
dv-processing

Release new-camera-capture-data_loss_cx3

iniVation AG

Jul 21, 2025

GETTING STARTED

1	Introduction	3
2	Getting started	5
2.1	Installation of dv-processing	5
2.2	Usage in a project	7
3	Basics	9
3.1	Event store	10
3.2	Event stream slicing	15
3.3	Event accumulation	27
3.4	Filtering events	46
3.5	Command-line utilities	64
3.6	Reading camera data	72
3.7	Writing camera data	97
3.8	Network streaming	106
4	Vision algorithms	113
4.1	Camera Geometry	113
4.2	Feature detection in events	121
4.3	Feature tracking	132
4.4	Kinematics primitives	148
4.5	Mean-shift clustering	151
5	Advanced applications	157
5.1	Depth estimation	157
5.2	Motion Compensation	169
5.3	Contrast Maximization	173
6	API	179
6.1	API	179
7	Help	649
	Index	651

Generic processing algorithms for event cameras.

INTRODUCTION

This documentation describes the API and the algorithms available in the `dv-processing` library. The documentation covers the basic usage of the library for event camera data processing. The library builds on top of C++20 coding standard, provides state-of-the-art algorithms to process event data streams from iniVation cameras with high efficiency. The library also provides Python bindings that allow users to develop high performance event processing application in Python as well. Since the library is built on top of modern C++, it extensively uses template metaprogramming to provide extensible and performant implementations of event processing algorithms. Python bindings have some limitations due to the use of templates, but most applications can be developed using Python alone. Extensive code samples in both C++ and Python are provided next to algorithm description, the samples also follow modern coding style conventions. More information about the coding style convention can be found [here](https://github.com/isocpp/CppCoreGuidelines/blob/master/CppCoreGuidelines.md)¹. Code samples apply the [constant and immutability rules](https://github.com/isocpp/CppCoreGuidelines/blob/master/CppCoreGuidelines.md#S-const)² from the previously mentioned document.

¹ <https://github.com/isocpp/CppCoreGuidelines/blob/master/CppCoreGuidelines.md>

² <https://github.com/isocpp/CppCoreGuidelines/blob/master/CppCoreGuidelines.md#S-const>

GETTING STARTED

This section covers installation and usage of the library in Linux , Windows , and MacOS . Usage in CMake projects is covered for the C++ API, and pip installation is preferred for Python projects.

2.1 Installation of dv-processing

The dv-processing library can be installed using different package management tools as well as installations from source. Since it is a header-only library, it can be used as a submodule in other CMake C++ projects.

2.1.1 C++

Installation of dv-processing headers is available on Linux, MacOS, and Windows platforms.

Linux

Ubuntu 20.04 / 22.04 / 24.04 (apt)

```
sudo add-apt-repository ppa:ubuntu-toolchain-r/test
sudo add-apt-repository ppa:inivation-ppa/inivation
sudo apt update
sudo apt install dv-processing
```

Fedora Linux

```
sudo dnf copr enable inivation/inivation
sudo dnf install dv-processing
```

Arch Linux

You can find dv-processing in the AUR repository, install the package 'dv-processing'.

The standard packages in the AUR repository already include all development files.

Gentoo Linux

A valid Gentoo ebuild repository is available [here](https://gitlab.com/inivation/gentoo-inivation/)³ over Git. The package to install is 'dev-libs/dv-processing'.

The standard packages in the Gentoo ebuild repository already include all development files.

³ <https://gitlab.com/inivation/gentoo-inivation/>

MacOS (brew)

Please notice that MacOS installation requires a recent compiler version (Apple XCode >= 16.0 or LLVM clang >= 18).

```
brew tap inivation/inivation
brew install dv-processing
```

Windows (VCPKG)

Windows installation is supported using the VCPKG package manager. This requires cloning of the VCPKG repository, you can follow the official [quick start guide to get started](#)⁴. The library can be installed using `.\vcpkg.exe install dv-processing` after VCPKG has been downloaded and bootstrapped.

A short tutorial for installation, create a directory at `C:\src`, execute these commands there:

```
git clone https://github.com/microsoft/vcpkg
.\vcpkg\bootstrap-vcpkg.bat
.\vcpkg\vcpkg install dv-processing
```

Installing `dv-processing[tools]` will also compile and install the command-line utilities, such as `dv-filestat`.

Installing as a git submodule

The library can be used in a CMake project by adding it as a git submodule. To add the library using git, call the following command from your project directory:

```
git submodule https://gitlab.com/inivation/dv/dv-processing.git thirdparty/dv-
↳processing
```

This will add the source code of the latest released version of `dv-processing` to a directory `thirdparty/dv-processing`. Now you can enable it in your `CMakeLists.txt`:

```
ADD_SUBDIRECTORY(thirdparty/dv-processing EXCLUDE_FROM_ALL)

# link your targets against the library
TARGET_LINK_LIBRARIES(your_target
    dv::processing
    ...)
```

Build and install from source

Manual build and installation from source is also possible. The library requires a C++20 compatible compiler and installed dependencies.

Please follow the installation instructions available in the [source code repository](#)⁵ to install the library from source.

2.1.2 Python

The `dv-processing` library is also available in Python using Pybind11 bindings. Since the underlying implementation remains C++ and only exposes the API through Python bindings, the performance of the provided methods remains high.

Installation using `pip install` is recommended, as it works on all supported operating systems, as well as VirtualEnv and Conda environments. On Linux systems, installation can also be performed through the package manager for the system Python installation.

⁴ <https://github.com/microsoft/vcpkg#quick-start-windows>

⁵ <https://gitlab.com/inivation/dv/dv-processing/>

Pip

```
python3 -m pip install dv-processing
```

Note

Pip installation is the only supported method for installation in a virtual Python environment (VirtualEnv / Conda).

Linux

Ubuntu 20.04 / 22.04 / 24.04 (apt)

```
sudo add-apt-repository ppa:inivation-ppa/inivation
sudo apt-get update
sudo apt-get install dv-processing-python
```

Fedora Linux

```
sudo dnf copr enable inivation/inivation
sudo dnf install dv-processing-python
```

Arch Linux

You can find dv-processing in the AUR repository, install the package ‘dv-processing’, which includes the Python bindings.

Gentoo Linux

A valid Gentoo ebuild repository is available [here](https://gitlab.com/inivation/gentoo-inivation/)⁶ over Git. The package to install is ‘dev-libs/dv-processing’ with the USE flag ‘python’ enabled.

2.2 Usage in a project

The library can be used in C++ CMake projects after a successful installation. Python projects support conda and venv environments using pip installation; pip installation is the preferred method for use in Python, although system-wide installation packages are also provided for some Linux distributions.

2.2.1 C++ (CMake)

To use the installed version of dv-processing in your project, add these lines to your CMakeLists.txt:

```
# Find installed dv-processing.
FIND_PACKAGE(dv-processing)

# Link your targets against the library
TARGET_LINK_LIBRARIES(your_target
    dv::processing
    ...)
```

⁶ <https://gitlab.com/inivation/gentoo-inivation/>

2.2.2 Python

After installing the dv-processing Python bindings either using pip or system-wide installation, the library is included using a simple import statement:

```
import dv_processing as dv  
  
print(dv.__version__)
```

Note

Pip installation is the only supported method for installation in a virtual Python environment (conda / venv).

BASICS

This section covers the very basic features of this library:

- Storing events in memory and efficiently accessing them;
- Frame image accumulation algorithms from events;
- Noise filtering in events and efficient subsampling;
- Input/Output of event data: live camera access, as well as reading / writing data from a file.



Fig. 1: Frame generated using `dv::EdgeMapAccumulator` class.

3.1 Event store

The dv-processing library provides an easy to use and an efficient data structure to store and manage incoming event data from the camera - the `dv::EventStore` class. It is implemented as a shallow, shared ownership data structure which only holds pointers to actual memory locations containing event data. Similarly to how OpenCV handles image data, the `dv::EventStore` does not copy the data, instead it only holds pointers to received packets of events and uses metadata of these packets to efficiently slice the data given time-intervals.

3.1.1 Create and store events

The following sample code shows how to create an empty event store and fill it with software generated events:

C++

Python

```

1  #include <dv-processing/core/core.hpp>
2  #include <dv-processing/core/utils.hpp>
3
4  int main() {
5      // Initialize an empty store
6      dv::EventStore store;
7
8      // Get the current timestamp
9      const int64_t timestamp = dv::now();
10
11     // Add some events into the event store
12     // This allocates and inserts events at the back, the function arguments are:
13     // timestamp, x, y, polarity
14     store.emplace_back(timestamp, 0, 0, true);
15     store.emplace_back(timestamp + 1000, 1, 1, false);
16     store.emplace_back(timestamp + 2000, 2, 2, false);
17     store.emplace_back(timestamp + 3000, 3, 3, true);
18
19     // Perform time-based slicing of event store, the output event store "sliced"
    ↪ will contain
20     // the second and third events from above. The end timestamp (second argument) is
    ↪ 2001, since start
21     // timestamp (first argument) is inclusive and timestamp is exclusive, so 1 is
    ↪ added.
22     const dv::EventStore sliced = store.sliceTime(timestamp + 1000, timestamp + 2001);
23
24     // This should print two events
25     for (const dv::Event &ev : sliced) {
26         std::cout << fmt::format("Sliced event [{}, {}, {}, {}]", ev.timestamp(), ev.
    ↪ x(), ev.y(), ev.polarity())
27         << std::endl;
28     }
29
30     return 0;
31 }

```

```

1  import dv_processing as dv
2

```

(continues on next page)

(continued from previous page)

```

3  # Initialize an empty store
4  store = dv.EventStore()
5
6  # Get the current timestamp
7  timestamp = dv.now()
8
9  # Add some events into the event store
10 # This allocates and inserts events at the back, the function arguments are:
11 # timestamp, x, y, polarity
12 store.push_back(timestamp, 0, 0, True)
13 store.push_back(timestamp + 1000, 1, 1, False)
14 store.push_back(timestamp + 2000, 2, 2, False)
15 store.push_back(timestamp + 3000, 3, 3, True)
16
17 # Perform time-based slicing of event store, the output event store "sliced" will
18   ↳ contain
19 # the second and third events from above. The end timestamp (second argument) is 2001,
20   ↳ since start
21 # timestamp (first argument) is inclusive and timestamp is exclusive, so 1 is added.
22 sliced = store.sliceTime(timestamp + 1000, timestamp + 2001)
23
24 # This should print two events
25 for ev in store:
26     print(f"Sliced event [{ev.timestamp()}], {ev.x()}, {ev.y()}, {ev.polarity()}]")

```

3.1.2 Slicing Event Stores

`dv::EventStore` can be sliced by time or by the number of events. Slicing is a shallow operation, it does not copy any data. Slicing returns a new `dv::EventStore` that only references the data requested. The original store is unaffected.

Slicing by time

`dv::EventStore` implements event data slicing by time in an efficient way. By reusing the underlying packet structure, the slicing is performed with $O(\log n)$ time complexity. The following sample shows the usage of time based slicing functions:

C++

Python

```

1  #include <dv-processing/core/core.hpp>
2  #include <dv-processing/data/generate.hpp>
3
4  int main() {
5      using namespace std::chrono_literals;
6
7      // Generate 10 events with time range [10000; 20000]
8      const auto store = dv::data::generate::uniformEventsWithinTimeRange(10000, 10ms,
9   ↳ cv::Size(100, 100), 10);
10
11     // Get all events with timestamp above 12500, it will be 13000 and up
12     dv::EventStore eventsAfterTimestamp = store.sliceTime(12'500);

```

(continues on next page)

(continued from previous page)

```

13 // Print the timestamp ranges
14 std::cout << "1. " << eventsAfterTimestamp << std::endl;
15
16 // Slice event within time range [12000; 16000); the end time is exclusive
17 const dv::EventStore eventsInRange = store.sliceTime(12'000, 16'000);
18
19 // Print the timestamp ranges; It will print that range is [12000; 15000] since
  ↳end time is exclusive and
20 // event at timestamp 16000 is not going to be included.
21 std::cout << "2. " << eventsInRange << std::endl;
22
23 return 0;
24 }

```

```

1 import dv_processing as dv
2 from datetime import timedelta
3
4 # Generate 10 events with time range [10000; 20000]
5 store = dv.data.generate.uniformEventsWithinTimeRange(10000,
  ↳timedelta(milliseconds=10), (100, 100), 10)
6
7 # Get all events with timestamp above 12500, it will be 13000 and up
8 eventsAfterTimestamp = store.sliceTime(12500)
9
10 # Print the timestamp ranges
11 print(f"1. {eventsAfterTimestamp}")
12
13 # Slice event within time range [12000; 16000); the end time is exclusive
14 eventsInRange = store.sliceTime(12000, 16000)
15
16 # Print the timestamp ranges; It will print that range is [12000; 15000] since end
  ↳time is exclusive and
17 # event at timestamp 16000 is not going to be included.
18 print(f"2. {eventsInRange}")

```

Slicing by number of events

`dv::EventStore` provides slicing capabilities by an index and number of events. The first argument to the `dv::EventStore::slice()` method is a starting index from which the output slice starts, the second optional argument is the number of events to be sliced. If the number argument is not provided, the slice will contain all events from given index.

Following sample shows how to slice an event store within given indices of the underlying events:

C++

Python

```

1 #include <dv-processing/core/core.hpp>
2 #include <dv-processing/data/generate.hpp>
3
4 int main() {
5     using namespace std::chrono_literals;

```

(continues on next page)

(continued from previous page)

```

6      // Add 10 event with timestamps in range [10000; 20000]
7      const auto store = dv::data::generate::uniformEventsWithinTimeRange(10000, 10ms,
8      ↪cv::Size(100, 100), 10);
9
10     // Get all events beyond and including index 5
11     dv::EventStore eventsAfterIndex = store.slice(5);
12     std::cout << "1. " << eventsAfterIndex << std::endl;
13
14     // Get 3 events starting with index 2
15     dv::EventStore eventsInRange = store.slice(2, 3);
16     std::cout << "2. " << eventsInRange << std::endl;
17
18     // Use sliceBack to retrieve event from the end; this call will retrieve last 3
19     ↪events
20     dv::EventStore lastEvents = store.sliceBack(3);
21     std::cout << "3. " << lastEvents << std::endl;
22
23     return 0;
24 }

```

```

1  import dv_processing as dv
2  from datetime import timedelta
3
4  # Generate 10 events with time range [10000; 20000]
5  store = dv.data.generate.uniformEventsWithinTimeRange(10000,
6  ↪timedelta(milliseconds=10), (100, 100), 10)
7
8  # Get all events beyond and including index 5
9  events_after_index = store.slice(5)
10 print(f"1. {events_after_index}")
11
12 # Get 3 events starting with index 2
13 events_in_range = store.slice(2, 3)
14 print(f"2. {events_in_range}")
15
16 # Use sliceBack to retrieve event from the end; this call will retrieve last 3 events
17 last_events = store.sliceBack(3)
18 print(f"3. {last_events}")

```

Combining multiple EventStores

Multiple instances of `dv::EventStore` can be added together to have a single object to access and manage it. Since `dv::EventStore` uses pointers to underlying event packet data, combining does not involve any deep data copies, the implementation takes over shared memory ownership instead. Following sample show how to efficiently combine multiple event stores into one:

C++

Python

```

1  #include <dv-processing/core/core.hpp>
2  #include <dv-processing/data/generate.hpp>

```

(continues on next page)

(continued from previous page)

```

3
4 int main() {
5     using namespace std::chrono_literals;
6
7     // Generate 10 events with timestamps in range [10000; 20000]
8     const auto store1 = dv::data::generate::uniformEventsWithinTimeRange(10000, 10ms,
↳cv::Size(100, 100), 10);
9
10    // Generate second event store with 10 events with timestamps in range [20000;
↳29000] to the second store
11    const auto store2 = dv::data::generate::uniformEventsWithinTimeRange(20000, 10ms,
↳cv::Size(100, 100), 10);
12
13    // Final event store which will contain all events
14    dv::EventStore finalStore;
15
16    // Add the events into the final store; this operation is shallow, so no data
↳copies
17    // are performed, but the underlying data has shared ownership between all stores
18    finalStore.add(store1);
19    finalStore.add(store2);
20
21    // Print specific information on what we contain in the event store
22    std::cout << finalStore << std::endl;
23
24    return 0;
25 }

```

```

1 import dv_processing as dv
2 from datetime import timedelta
3
4 # Generate 10 events with timestamps in range [10000; 20000]
5 store1 = dv.data.generate.uniformEventsWithinTimeRange(10000,
↳timedelta(milliseconds=10), (100, 100), 10)
6
7 # Generate second event store with 10 events with timestamps in range [20000; 29000]
↳to the second store
8 store2 = dv.data.generate.uniformEventsWithinTimeRange(20000,
↳timedelta(milliseconds=10), (100, 100), 10)
9
10 # Final event store which will contain all events
11 final_store = dv.EventStore()
12
13 # Add the events into the final store; this operation is shallow, so no data copies
14 # are performed, but the underlying data has shared ownership between all stores
15 final_store.add(store1)
16 final_store.add(store2)
17
18 # Print specific information on what we contain in the final event store
19 print(f"{final_store}")

```

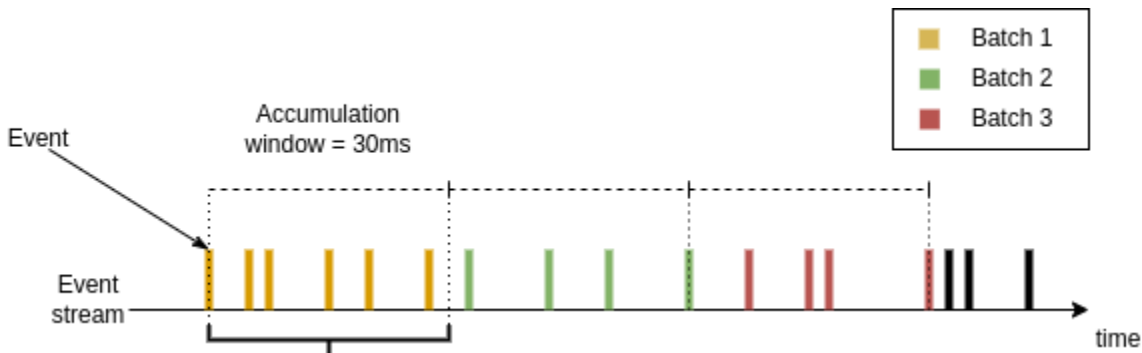
3.2 Event stream slicing

Events are a sparse representation for brightness changes registered by an event camera. It is useful to group incoming events from a stream in certain time-windows or by a certain number. The dv-processing library provides an efficient, yet powerful tools to approach continuous slicing of events. Following chapters will provide samples on how to use available event stream slicer implementations to group events in specific batches.

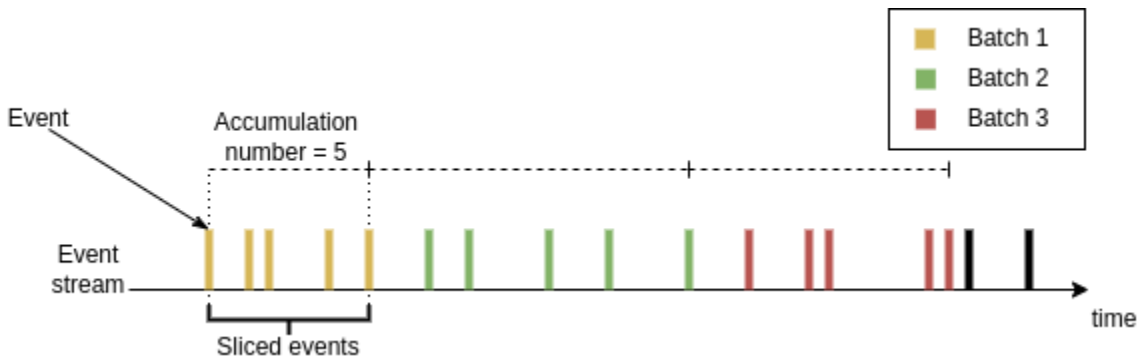
3.2.1 EventStreamSlicer

`dv::EventStreamSlicer` implements event slicing for a single event stream. Incoming events are passed into the slicer using the `accept()` method, the underlying implementation applies required slicing approach and resulting sliced events are passed into registered callback methods. The slicing can be performed using fixed size time-windows or fixed size number of events. According callback function can be registered using `dv::EventStreamSlicer::doEveryNumberOfEvents()` or `dv::EventStreamSlicer::doEveryTimeInterval()` methods. The registered callbacks and their calling parameters can also be modified later on.

The approach can be visualized by looking at a stream of events coming from a camera as a time series. The slicing can be performed by splitting the stream by a fixed time window as shown in a diagram below:



The other approach is to slice by a fixed number of events instead, as shown in the diagram below:



Slicing a stream

Following sample shows how to perform time-based slicing of an event stream:

C++

Python

```
1 #include <dv-processing/core/core.hpp>
2 #include <dv-processing/data/generate.hpp>
3
```

(continues on next page)

(continued from previous page)

```

4 int main() {
5     // Use this namespace to enable literal time expression from the chrono library
6     using namespace std::chrono_literals;
7
8     // Initialize slicer, it will have no jobs at this time
9     dv::EventStreamSlicer slicer;
10
11     // Register this method to be called every 33 millisecond worth of event data
12     slicer.doEveryTimeInterval(33ms, [](const dv::EventStore &events) {
13         std::cout << "* Received events time-based slicing: " << events << std::endl;
14     });
15
16     // Register this method to be called every 100 events
17     slicer.doEveryNumberOfElements(100, [](const dv::EventStore &events) {
18         std::cout << "# Received events in number-based slicing: " << events <<
19         std::endl;
20     });
21
22     // Generate 1000 events within 2 second interval. These will be sliced correctly
23     // by the slicer.
24     const dv::EventStore store = dv::data::generate::uniformEventsWithinTimeRange(0,
25     2s, cv::Size(100, 100), 1000);
26
27     // Now push the store into the slicer, the data contents within the store
28     // can be arbitrary, the slicer implementation takes care of correct slicing
29     // algorithm and calls the previously registered callbacks accordingly.
30     slicer.accept(store);
31
32     return 0;
33 }

```

```

1 import dv_processing as dv
2 from datetime import timedelta
3
4 # Initialize slicer, it will have no jobs at this time
5 slicer = dv.EventStreamSlicer()
6
7
8 def print_time_interval(events: dv.EventStore):
9     # Print the time duration received by this method
10    print(f"* Received events time-based slicing: {events}")
11
12
13 # Register this method to be called every 33 millisecond worth of event data
14 slicer.doEveryTimeInterval(timedelta(milliseconds=33), print_time_interval)
15
16
17 def print_event_number(events: dv.EventStore):
18     # Print the number of events received here
19     print(f"# Received events in number-based slicing: {events}")
20
21

```

(continues on next page)

(continued from previous page)

```

22 # Register this method to be called every 100 events
23 slicer.doEveryNumberOfElements(100, print_event_number)
24
25 # Generate 1000 events within 2 second interval. These will be sliced correctly by
26 ↪the slicer.
27 store = dv.data.generate.uniformEventsWithinTimeRange(0, timedelta(seconds=2), (100,
28 ↪100), 1000)
29
30 # Now push the store into the slicer, the data contents within the store
31 # can be arbitrary, the slicer implementation takes care of correct slicing
32 # algorithm and calls the previously registered callbacks accordingly.
33 slicer.accept(store)

```

Modifying slicing parameters

Slicing parameters can also be modified during runtime if needed. Following code sample is a modified version of the sample from previous chapter, which shows how to use the parameter modification methods:

C++

Python

```

1 #include <dv-processing/core/core.hpp>
2 #include <dv-processing/data/generate.hpp>
3
4 int main() {
5     // Use this namespace to enable literal time expression from the chrono library
6     using namespace std::chrono_literals;
7
8     // Initialize slicer, it will have no jobs at this time
9     dv::EventStreamSlicer slicer;
10
11     // Register this method to be called every 33 millisecond worth of event data
12     int timeJobId = slicer.doEveryTimeInterval(33ms, [] (const dv::EventStore &events)
13 ↪{
14         std::cout << "* Received events time-based slicing: " << events << std::endl;
15     });
16
17     // Register this method to be called every 100 events
18     int numberJobId = slicer.doEveryNumberOfElements(100, [] (const dv::EventStore &
19 ↪events) {
20         std::cout << "# Received events in number-based slicing: " << events <<
21 ↪std::endl;
22     });
23
24     // Implement data generation; The following loop will generate 10 packets of
25 ↪events,
26     // each containing 100 events within 20 millisecond duration.
27     for (int i = 0; i < 10; i++) {
28         // Generate 100 events within 20 millisecond interval. These will be sliced
29 ↪correctly by the slicer.
30         const auto store = dv::data::generate::uniformEventsWithinTimeRange(i * 20
31 ↪'000, 20ms, cv::Size(100, 100), 100);

```

(continues on next page)

(continued from previous page)

```

26
27     // Now push the store into the slicer, the data contents within the store
28     // can be arbitrary, the slicer implementation takes care of correct slicing
29     // algorithm and calls the previously registered callbacks accordingly.
30     slicer.accept(store);
31
32     // When a packet with index 5 is reached, modify the parameters
33     if (i == 5) {
34         // Modify time range to 10 milliseconds instead of 33
35         slicer.modifyTimeInterval(timeJobId, 10ms);
36         // Modify number to 200 instead of 100
37         slicer.modifyNumberInterval(numberJobId, 200);
38     }
39 }
40

```

```

1  import dv_processing as dv
2  from datetime import timedelta
3
4  # Initialize slicer, it will have no jobs at this time
5  slicer = dv.EventStreamSlicer()
6
7
8  def print_time_interval(events: dv.EventStore):
9      # Print the time duration received by this method
10     print(f"* Received event with {events.duration()} duration in time-based slicing")
11
12
13     # Register this method to be called every 33 millisecond worth of event data
14     time_job_id = slicer.doEveryTimeInterval(timedelta(milliseconds=33), print_time_
15     ↪interval)
16
17     def print_event_number(events: dv.EventStore):
18         # Print the number of events received here
19         print(f"# Received {events.size()} events in number-based slicing")
20
21
22     # Register this method to be called every 100 events
23     number_job_id = slicer.doEveryNumberOfElements(100, print_event_number)
24
25     # Implement data generation; The following loop will generate 10 stores
26     # of events, each containing 100 events within 10 millisecond duration.
27     for i in range(10):
28         # Generate 100 events within 20 millisecond interval. These will be sliced
29         ↪correctly by the slicer.
30         store = dv.data.generate.uniformEventsWithinTimeRange(i * 20000,
31         ↪timedelta(milliseconds=20), (100, 100), 100)
32
33         # Now push the store into the slicer, the data contents within the store
34         # can be arbitrary, the slicer implementation takes care of correct slicing
35         # algorithm and calls the previously registered callbacks accordingly.

```

(continues on next page)

(continued from previous page)

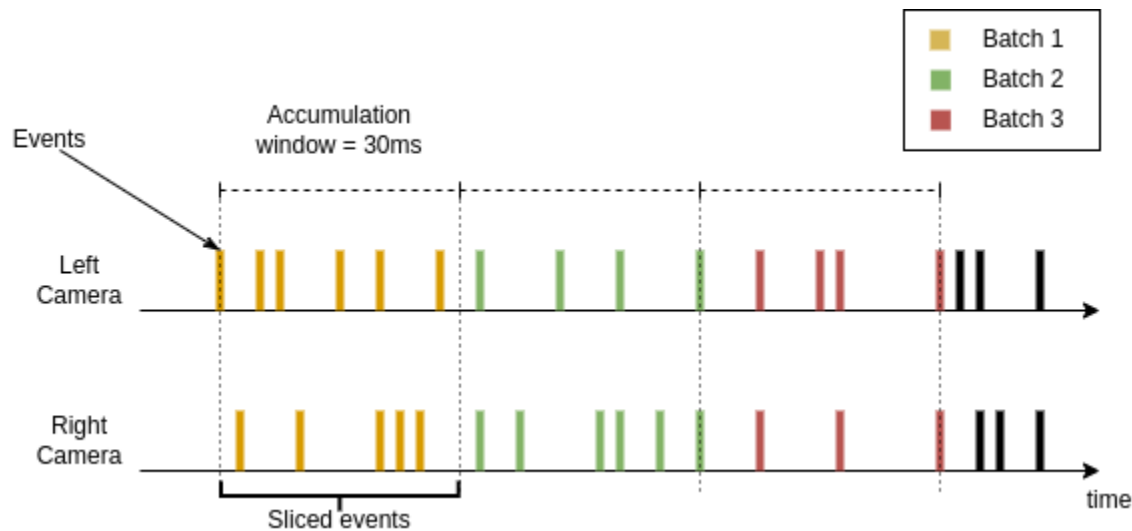
```

34 slicer.accept(store)
35
36 # When a packet with index 5 is reached, modify the parameters
37 if i == 5:
38     # Modify time range to 10 milliseconds instead of 33
39     slicer.modifyTimeInterval(time_job_id, timedelta(milliseconds=10))
40     # Modify number to 200 instead of 100
41     slicer.modifyNumberInterval(number_job_id, 200)

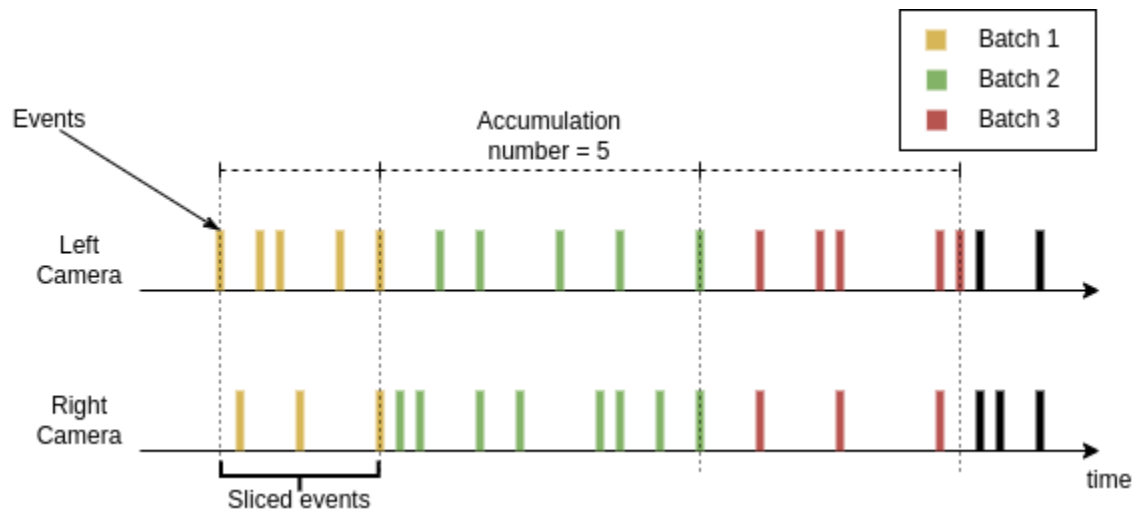
```

3.2.2 Stereo stream slicer

A dual stream slicer is required to perform synchronized time slicing from a stereo camera setup. This is implemented in the `dv::StereoEventStreamSlicer` class. It implements stereo event stream slicing by applying regular stream slicing on one of the input streams and performing time-based slicing within the same time-window. An illustration of the approach for slicing by time is displayed below:



Since the input data can be coming at different rates, in case of slicing by number, the according event from right camera are selected within the same time window. This case is shown below:



In the image above, the slicer is assumed to have a setting to slice every 5 events, these events are sliced from the left

camera and their exact timestamps are used to slice events from the right camera.

A sample code on how to use stereo event stream slicer with a live camera is shown below:

C++

Python

```

1  #include <dv-processing/core/stereo_event_stream_slicer.hpp>
2  #include <dv-processing/io/camera/discovery.hpp>
3
4  int main() {
5      using namespace std::chrono_literals;
6
7      // Discover connected camera to the system
8      const auto cameras = dv::io::camera::discover();
9      if (cameras.size() < 2) {
10         throw dv::exceptions::RuntimeError("Unable to discover two cameras");
11     }
12
13     // Open the cameras, just use first detected cameras
14     auto leftCamera = dv::io::camera::openSync(cameras[0]);
15     auto rightCamera = dv::io::camera::openSync(cameras[1]);
16
17     dv::io::camera::synchronizeAnyTwo(leftCamera, rightCamera);
18
19     // Initialize a stereo stream slicer
20     dv::StereoEventStreamSlicer slicer;
21
22     // Register a job to be performed every 33 milliseconds
23     slicer.doEveryTimeInterval(33ms, [] (const dv::EventStore &leftEvents, const
↪ dv::EventStore &rightEvents) {
24         // Print durations for time-based slicing callback
25         std::cout << fmt::format(
26             "* Received events with duration: left[{}] - right[{}]", leftEvents.
↪ duration(), rightEvents.duration())
27             << std::endl;
28     });
29
30     // Register a job to be performed every 1000 events
31     slicer.doEveryNumberOfEvents(
32         // Here we receive events from two camera, time-synchronized
33         1000, [] (const dv::EventStore &leftEvents, const dv::EventStore &rightEvents)
↪ {
34         // Print event store sizes for number based slicing callback
35         std::cout << fmt::format("# Received events in number-based slicing,
↪ counts: left[{}] - right[{}]",
36             leftEvents.size(), rightEvents.size())
37             << std::endl;
38     });
39
40     // Continue the loop while both cameras are connected
41     while (leftCamera->isRunning() && rightCamera->isRunning()) {
42         // Initialize
43         dv::EventStore left, right;

```

(continues on next page)

(continued from previous page)

```

44
45     // Handle left camera events
46     if (const auto batch = leftCamera->getNextEventBatch(); batch.has_value()) {
47         left = *batch;
48     }
49
50     // Handle right camera events
51     if (const auto batch = rightCamera->getNextEventBatch(); batch.has_value()) {
52         right = *batch;
53     }
54
55     // Pass all events into the slicer
56     slicer.accept(left, right);
57 }
58
59 return 0;
60 }

```

```

1  from datetime import timedelta
2
3  import dv_processing as dv
4
5  # Discover connected camera to the system
6  cameras = dv.io.camera.discover()
7  if len(cameras) < 2:
8      raise RuntimeError("Unable to discover two cameras")
9
10 # Open the cameras
11 leftCamera = dv.io.camera.openSync(cameras[0])
12 rightCamera = dv.io.camera.openSync(cameras[1])
13
14 dv.io.camera.synchronizeAnyTwo(leftCamera, rightCamera)
15
16 # Initialize a stereo stream slicer
17 slicer = dv.io.StereoEventStreamSlicer()
18
19
20 # Callback method for time based slicing
21 def print_time_interval(left_events: dv.EventStore, right_events: dv.EventStore):
22     # Print the time duration received by this method
23     print(f"* Received events with duration: left[{left_events.duration()}] - right[
    ↳ {right_events.duration()}]")
24
25
26 # Register a job to be performed every 33 milliseconds
27 slicer.doEveryTimeInterval(timedelta(milliseconds=33), print_time_interval)
28
29
30 # Callback method for number based slicing
31 def print_event_number(left_events: dv.EventStore, right_events: dv.EventStore):
32     # Print the number of events received here
33     print(

```

(continues on next page)

(continued from previous page)

```

34         f"# Received events in number-based slicing, counts: left[{left_events.size()}
    ↪] - right[{right_events.size()}]"
35
36
37     # Register this method to be called every 1000 events
38     slicer.doEveryNumberOfEvents(1000, print_event_number)
39
40     # Continue the loop while both cameras are connected
41     while leftCamera.isRunning() and rightCamera.isRunning():
42         # Initialize empty stores
43         left = dv.EventStore()
44         right = dv.EventStore()
45
46         # Receive packet from left camera
47         leftPacket = leftCamera.getNextEventBatch()
48         # Assign the packet if some data was received
49         if leftPacket is not None:
50             left = leftPacket
51
52         # Receive packet from right camera
53         rightPacket = rightCamera.getNextEventBatch()
54         # Assign the packet if some data was received
55         if rightPacket is not None:
56             right = rightPacket
57
58         # Pass all events into the slicer
59         slicer.accept(left, right)

```

3.2.3 Generic data stream slicing

Event stream slicers described previously are intended for efficient and specific slicing approach to event stream data. The dv-processing library also provides a generic time-series data stream slicer class `dv::MultiStreamSlicer` that applies the same approach, but supports arbitrary number of streams and arbitrary data-types. The only requirement for data types is the timestamp data, that can be accessed through a predefined API. The predefined API requirement is implemented using C++20 concepts and templates, so the requirement can be satisfied by any type that can provide a microsecond timestamp expressed in a signed 64-bit integer.

The `dv::MultiStreamSlicer` class provides same slicing capabilities with methods allowing to slice by time `dv::MultiStreamSlicer::doEveryTimeInterval()` or by number `dv::MultiStreamSlicer::doEveryNumberOfElements()`. Since the `dv::MultiStreamSlicer` supports an arbitrary number of streams, the streams are managed by assigning a unique string name to each stream by using the `dv::MultiStreamSlicer::addStream()` method. By default, the slicer accepts the timestamped data types from the DV flatbuffer type system, such as: `dv::EventPacket`, `dv::IMUPacket`, and other packet data types. If a data type is provided without a container, it can be used with STL containers, such as `std::vector`. In a case an image frame `dv::Frame` stream needs to be sliced, it has to be wrapped in a container, e.g. using `std::vector<std::vector<dv::Frame>>`. Such a contained and timestamped data can be sliced alongside other data type streams.

To be clear about how the `MultiStreamSlicer` manages data slicing, it uses a convention of “main stream” and “secondary streams”. Main stream is declared in the constructor of the class and all other stream are added using the `MultiStreamSlicer::addStream()` method. The main stream is the driving data stream slicing; while secondary streams are following the time-ranges that are sliced from the main stream, similar to how `dv::StereoEventStreamSlicer` works.

Following sample shows the use of the `MultiStreamSlicer` to synchronously slice incoming frame and event streams

from a DAVIS346 camera and show a preview. The preview below

```

1  #include <dv-processing/core/multi_stream_slicer.hpp>
2  #include <dv-processing/io/camera/davis.hpp>
3  #include <dv-processing/visualization/event_visualizer.hpp>
4
5  #include <opencv2/highgui.hpp>
6  #include <opencv2/imgproc.hpp>
7
8  int main() {
9      // Open a DAVIS camera, the sample will slice incoming frames and events.
10     ↪synchronously using the generic data
11     // stream slicer. The camera name is an empty string to open any DAVIS camera.
12     ↪detected on the system
13     auto camera = dv::io::camera::DAVIS();
14
15     // Declare the main stream type to be std::vector<dv::Frame> with stream name
16     ↪"frames"
17     dv::MultiStreamSlicer<std::vector<dv::Frame>> slicer("frames");
18
19     // Add a secondary stream of events
20     slicer.addStream<dv::EventStore>("events");
21
22     // It's possible to add additional secondary streams for slicing here, e.g.
23     ↪adding a trigger stream
24     // would look like:
25     // slicer.addStream<dv::TriggerPacket>("triggers");
26
27     // Use visualizer to overlay events on a frame
28     dv::visualization::EventVisualizer visualizer(camera.getEventResolution().value(),
29     ↪ dv::visualization::colors::white,
30     dv::visualization::colors::green, dv::visualization::colors::red);
31
32     // Declare display windows for frames
33     cv::namedWindow("Preview", cv::WINDOW_NORMAL);
34
35     // Register a job to be performed every frame, the event will be sliced in-
36     ↪between the main stream frames
37     slicer.doEveryNumberOfElements(
38     // Here we receive time-synchronized frames and events; sliced data is
39     ↪provided in a std::unordered_map
40     // for easy access using the stream name.
41     1, [&visualizer](const auto &data) {
42     // Extract events and pass them to the slicer, data is retrieved using
43     ↪the helper method `get()`
44     // which accepts the stream name and the type. Stream name and type has
45     ↪to match, otherwise
46     // an exception will be thrown
47     const auto events = data.template get<dv::EventStore>("events");
48
49     // Retrieve frames, although we get one frame per slice, it is stored in
50     ↪the configured container
51     const auto frames = data.template get<std::vector<dv::Frame>>("frames");

```

(continues on next page)

(continued from previous page)

```

42
43     // The container is non-empty, we expect only one frame in the container,
↳so just display
44     // the first frame in the container
45     const dv::Frame &frame = frames.at(0);
46
47     // Convert the frame into a grayscale image for overlay preview
48     cv::Mat preview;
49     if (frame.image.channels() == 3) {
50         // The image is already grayscale, no conversion is needed
51         preview = frame.image;
52     }
53     else {
54         // The image coming from the camera is multichannel image, so we can
↳safely assume
55         // it's a grayscale image, convert it into BGR for overlay drawing
56         cv::cvtColor(frame.image, preview, cv::COLOR_GRAY2BGR);
57     }
58
59     // Overlay events on top of preview image
60     visualizer.generateImage(events, preview);
61
62     // Display the overlayed image
63     cv::imshow("Preview", preview);
64
65     // If escape button is pressed (code 27 is escape key), exit the program
↳cleanly
66     if (cv::waitKey(2) == 27) {
67         exit(0);
68     }
69     });
70
71     // Continue the loop while both cameras are connected
72     while (camera.isRunning()) {
73         // Handle events
74         if (const auto events = camera.getNextEventBatch(); events.has_value()) {
75             slicer.accept("events", *events);
76         }
77
78         // Handle frames
79         if (const auto frame = camera.getNextFrame(); frame.has_value()) {
80             slicer.accept("frames", *frame);
81         }
82     }
83
84     return 0;
85 }

```

The multi stream slicer is also available in python, but due to missing template functionality in python, the python version is more limited. `dv.EventMultiStreamSlicer` class is provided in python, which only supports events as main stream and has slightly different API, but it supports arbitrary number of secondary streams, which can be of any type that is built-in in dv-processing. Secondary streams can be added using named methods for types.

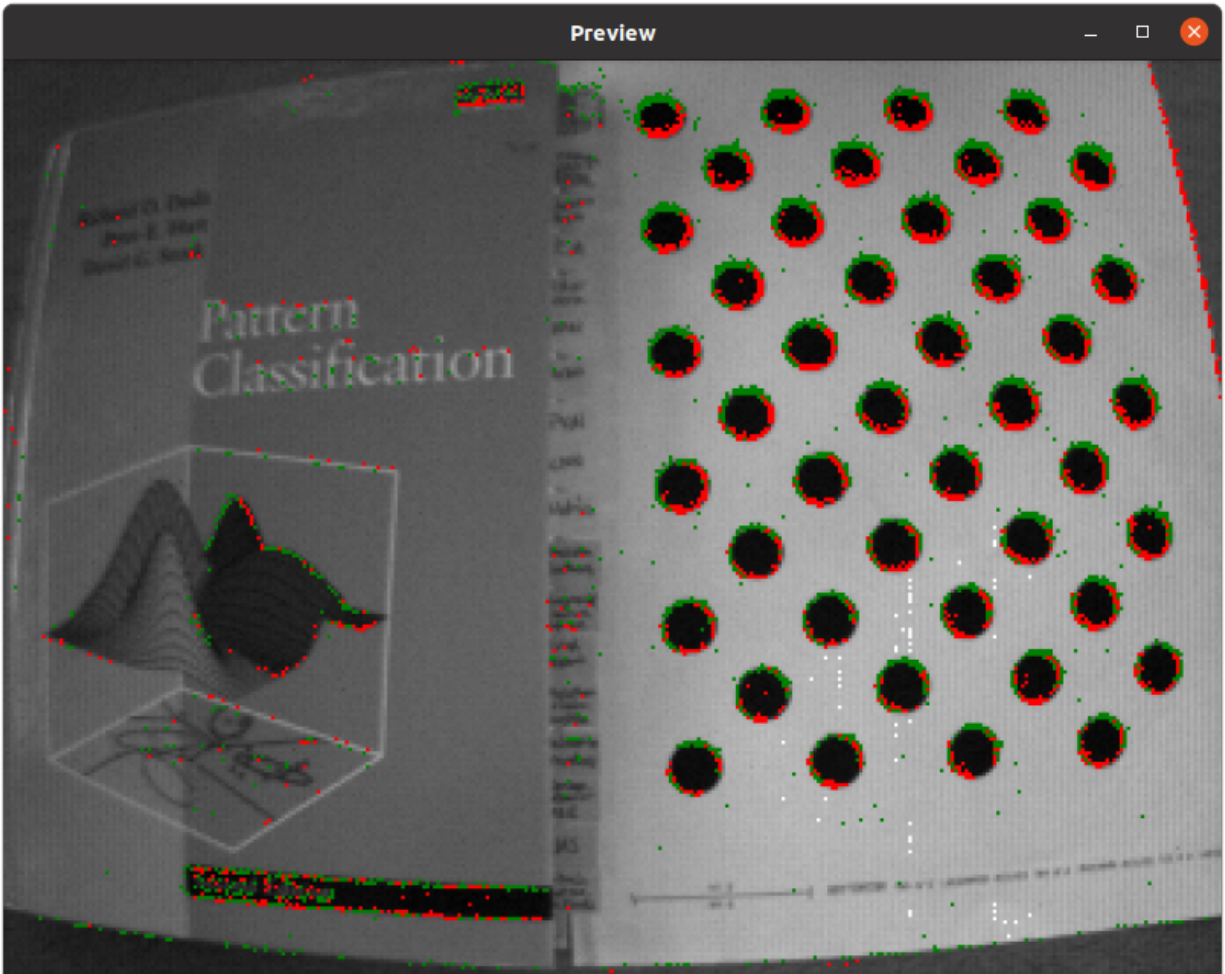


Fig. 2: Output from the multi stream sample usage - frames with synchronized events overlayed in the frame image.

The example below shows the use of the multi-stream slicer in python which uses events as main stream and slices frames as secondary stream:

```

1  from datetime import timedelta
2
3  import cv2 as cv
4  import dv_processing as dv
5
6  # Open the camera, just use first detected DAVIS camera
7  camera = dv.io.camera.DAVIS()
8
9  # Initialize a multi-stream slicer
10 slicer = dv.EventMultiStreamSlicer("events")
11
12 # Add a frame stream to the slicer
13 slicer.addFrameStream("frames")
14
15 # Initialize a visualizer for the overlay
16 visualizer = dv.visualization.EventVisualizer(camera.getEventResolution(), dv.
    ↪ visualization.colors.white(),
17                                         dv.visualization.colors.green(), dv.
    ↪ visualization.colors.red())
18
19 # Create a window for image display
20 cv.namedWindow("Preview", cv.WINDOW_NORMAL)
21
22
23 # Callback method for time based slicing
24 def display_preview(data):
25     # Retrieve frame data using the named method and stream name
26     frames = data.getFrames("frames")
27
28     # Retrieve event data
29     events = data.getEvents("events")
30
31     # Retrieve and color convert the latest frame of retrieved frames
32     latest_image = None
33     if len(frames) > 0:
34         if len(frames[-1].image.shape) == 3:
35             # We already have colored image, no conversion
36             latest_image = frames[-1].image
37         else:
38             # Image is grayscale, convert to color (BGR image)
39             latest_image = cv.cvtColor(frames[-1].image, cv.COLOR_GRAY2BGR)
40     else:
41         return
42
43     # Generate a preview and show the final image
44     cv.imshow("Preview", visualizer.generateImage(events, latest_image))
45
46     # If escape button is pressed (code 27 is escape key), exit the program cleanly
47     if cv.waitKey(2) == 27:
48         exit(0)

```

(continues on next page)

(continued from previous page)

```

49
50
51 # Register a job to be performed every 33 milliseconds
52 slicer.doEveryTimeInterval(timedelta(milliseconds=33), display_preview)
53
54 # Continue the loop while both cameras are connected
55 while camera.isRunning():
56     events = camera.getNextEventBatch()
57     if events is not None:
58         slicer.accept("events", events)
59
60     frame = camera.getNextFrame()
61     if frame is not None:
62         slicer.accept("frames", [frame])

```

This sample will result in similar output to the C++ version, but it rather synchronizes events to frames, so output of the samples are comparable, but not exact.

3.3 Event accumulation

Events are a sparse representation of brightness changes measured by the pixels on a camera sensor. Accumulation is a method applied to events to generate a frame representation of events. The dv-processing library provides a few highly optimized algorithm implementations to perform event accumulation and achieve different frame-like representation of the events.

3.3.1 Definitions

Certain definitions are used in this chapter and within accumulator API. Following is a list of specific definitions and their meaning within context of the accumulator:

- Potential - normalized pixel brightness value in some floating point range; it is an internal representation range for brightness that can be scaled into other brightness representations;
- Contribution - a numeric value that an event contributes to the brightness of pixel;
- Decay - pixel brightness correction applied over time when no events contribute to its brightness;
- Neutral potential - a default pixel brightness without any contribution;
- Minimum / maximum potential - limits for potential value representation.

3.3.2 Accumulator

The `dv::Accumulator` is a generalized implementation of a few accumulation algorithms that can be configured using class methods. Following chapter will describe the configuration options available for the accumulator. Final chapter of this section will provide a code sample that shows the use of all these configuration options for accumulation of events from a live camera.

Decay function and decay param

One of *None*, *Linear*, *Exponential*, *Step*. Defines the data degradation function that should be applied to the image. For each function, the *Decay param* setting assumes a different function:

Function	Enum value	Decay param function	Explanation
None	Decay::NONE	No function	Does not apply any decay
Linear	Decay::LINEAR	The slope a of the linear function, in <i>intensity per microsecond</i>	Assume intensity of a pixel is I_0 at time 0, this function applies a linear decay of the form of $I_0 - (t * \text{decayparam})$ or $I_0 + (t * \text{decayparam})$ until the value hits the value specified in <i>neutral potential</i>
Exponential	Decay::EXPONENTIAL	The time constant τ of the exponential function in <i>microseconds</i>	Assume intensity of a pixel is I_0 at time 0, this function applies an exponential decay of the form $I_0 * \exp(-(t/\text{decayparam}))$. The decay approaches a value of <i>neutralPotential</i> over time.
Step	Decay::STEP	No function	Set all pixel values to <i>neutral potential</i> after a frame is extracted.

The decay function can be set using `dv::Accumulator::setDecayFunction()` method.

Event Contribution

The contribution an event has onto the image. If an event arrives at a position x, y , the pixel value in the frame at x, y gets increased / decreased by the value of *Event contribution*, based on the events polarity.

Except:

- The resulting pixel value would be higher than *Max potential*, the value gets set to *Max potential* instead
- The resulting pixel value would be lower than *Min potential*, the value gets set to *Min potential* instead
- The event polarity is negative, and *Ignore polarity* is enabled, then the event is counted positively

Event contribution can be set using the `dv::Accumulator::setEventContribution()` method.

Min potential / Max potential

Sets the minimum and maximum values a pixel can achieve. If the value of the pixel would reach higher or lower, it is capped at these values. These values are also used for normalization at the output. The frame the module generates is an unsigned 8-bit grayscale image, normalized between *Min potential* and *Max potential*. A pixel with the value *Min potential* corresponds to a pixel with the value 0 in the output frame. A pixel with the value *Max potential* corresponds to a pixel with the value 255 in the output frame.

Min potential and *Max potential* can be set using `dv::Accumulator::setMinPotential()` and `dv::Accumulator::setMaxPotential()` methods.

Neutral potential

This setting has different effects depending on the decay function:

Function	Neutral potential function
None	No function.
Linear	Pixel brightness value decays linearly into the <i>neutral potential</i> value.
Exponential	No function.
Step	Each pixel value is set to <i>neutral potential</i> after generating a frame.

Neutral potential can be set using `dv::Accumulator::setNeutralPotential()`.

Ignore polarity

If this value is set, all events act as if they had positive polarity. In this case, *Event contribution* is always taken positively. This can be used to generate edge images instead of an actual image reconstruction. This can be done by setting *Neutral potential* and *Min potential* to zeros.

This feature can be enabled using the `dv::Accumulator::setIgnorePolarity()` method.

Synchronous Decay

If this value is set, decay happens in continuous time for all pixels. In every frame, each pixel will be eagerly decayed to the time the image gets generated. If this value is not set, decay at the individual pixel only happens when the pixel receives an event. Decay is lazily evaluated at the pixel.

Note

Both decay regimes yield the same overall decay over time, just the time at which it is applied changes. This parameter does not have an effect for Step decay. Step decay is always synchronous at generation time.

Synchronous decay can be enabled using the `dv::Accumulator::setSynchronousDecay()` method.

Accumulating event from a camera

The following sample code show how to use `dv::Accumulator` together with `dv::EventStreamSlicer` and `dv::io::camera::open()` to implement a pipeline that generates a continuous stream of accumulated frames:

C++

Python

```

1  #include <dv-processing/core/frame.hpp>
2  #include <dv-processing/io/camera/discovery.hpp>
3
4  #include <opencv2/highgui.hpp>
5
6  int main() {
7      using namespace std::chrono_literals;
8
9      // Open any camera
10     auto capture = dv::io::camera::open();
11
12     // Make sure it supports event stream output, throw an error otherwise
13     if (!capture->isEventStreamAvailable()) {
14         throw dv::exceptions::RuntimeError("Input camera does not provide an event_
↪ stream.");
15     }
16
17     // Initialize an accumulator with some resolution
18     dv::Accumulator accumulator(*capture->getEventResolution());
19
20     // Apply configuration, these values can be modified to taste
21     accumulator.setMinPotential(0.f);
22     accumulator.setMaxPotential(1.f);
23     accumulator.setNeutralPotential(0.5f);
24     accumulator.setEventContribution(0.15f);

```

(continues on next page)

(continued from previous page)

```

25     accumulator.setDecayFunction(dv::Accumulator::Decay::EXPONENTIAL);
26     accumulator.setDecayParam(1e+6);
27     accumulator.setIgnorePolarity(false);
28     accumulator.setSynchronousDecay(false);
29
30     // Initialize a preview window
31     cv::namedWindow("Preview", cv::WINDOW_NORMAL);
32
33     // Initialize a slicer
34     dv::EventStreamSlicer slicer;
35
36     // Register a callback every 33 milliseconds
37     slicer.doEveryTimeInterval(33ms, [&accumulator](const dv::EventStore &events) {
38         // Pass events into the accumulator and generate a preview frame
39         accumulator.accept(events);
40         dv::Frame frame = accumulator.generateFrame();
41
42         // Show the accumulated image
43         cv::imshow("Preview", frame.image);
44         cv::waitKey(2);
45     });
46
47     // Run the event processing while the camera is connected
48     while (capture->isRunning()) {
49         // Receive events, check if anything was received
50         if (const auto events = capture->getNextEventBatch()) {
51             // If so, pass the events into the slicer to handle them
52             slicer.accept(*events);
53         }
54     }
55
56     return 0;
57 }

```

```

1  from datetime import timedelta
2
3  import cv2 as cv
4  import dv_processing as dv
5
6  # Open any camera
7  capture = dv.io.camera.open()
8
9  # Make sure it supports event stream output, throw an error otherwise
10 if not capture.isEventStreamAvailable():
11     raise RuntimeError("Input camera does not provide an event stream.")
12
13 # Initialize an accumulator with some resolution
14 accumulator = dv.Accumulator(capture.getEventResolution())
15
16 # Apply configuration, these values can be modified to taste
17 accumulator.setMinPotential(0.0)
18 accumulator.setMaxPotential(1.0)

```

(continues on next page)

(continued from previous page)

```

19 accumulator.setNeutralPotential(0.5)
20 accumulator.setEventContribution(0.15)
21 accumulator.setDecayFunction(dv.Accumulator.Decay.EXPONENTIAL)
22 accumulator.setDecayParam(1e+6)
23 accumulator.setIgnorePolarity(False)
24 accumulator.setSynchronousDecay(False)
25
26 # Initialize preview window
27 cv.namedWindow("Preview", cv.WINDOW_NORMAL)
28
29 # Initialize a slicer
30 slicer = dv.EventStreamSlicer()
31
32
33 # Declare the callback method for slicer
34 def slicing_callback(events: dv.EventStore):
35     # Pass events into the accumulator and generate a preview frame
36     accumulator.accept(events)
37     frame = accumulator.generateFrame()
38
39     # Show the accumulated image
40     cv.imshow("Preview", frame.image)
41     cv.waitKey(2)
42
43
44 # Register a callback every 33 milliseconds
45 slicer.doEveryTimeInterval(timedelta(milliseconds=33), slicing_callback)
46
47 # Run the event processing while the camera is connected
48 while capture.isRunning():
49     # Receive events
50     events = capture.getNextEventBatch()
51
52     # Check if anything was received
53     if events is not None:
54         # If so, pass the events into the slicer to handle them
55         slicer.accept(events)

```

3.3.3 Edge accumulation

The dv-processing library provides a highly optimized variant of accumulator for generating edge maps - `dv:EdgeMapAccumulator`. It was specifically optimized for speed of execution, so it has only a minimal set of settings and supported features compared to `dv:Accumulator`.

Below is a table providing available parameters for the `dv:EdgeMapAccumulator`:

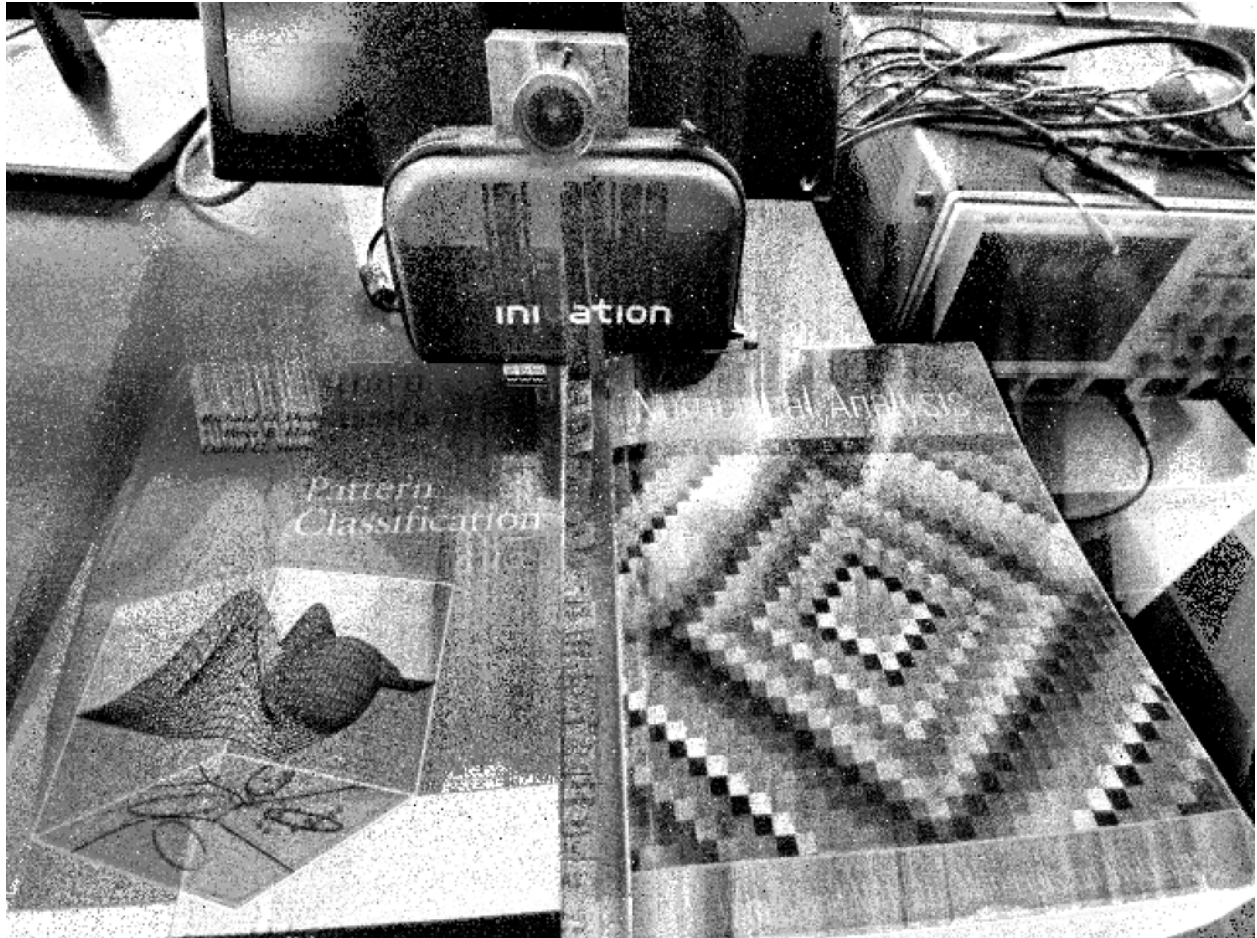


Fig. 3: Frame generated using `dv::Accumulator` class.

Parameter	Default value	Accepted values	Comment
Contribution	0.25	[0.0; 1.0]	Contribution potential for a single event.
Ignore polarity	true	boolean	All events are considered positive if enabled.
Neutral potential	0.0	[0.0; 1.0]	Neutral potential is the default pixel value when decay is disabled and the value that pixels decay into when decay is enabled.
Decay parameter	1.0	[0.0; 1.0]	This value defines how fast pixel values decay to neutral value. The bigger the value the faster the pixel value will reach neutral value. Decay is applied before each frame generation. The range for decay value is [0.0; 1.0], where 0.0 will not apply any decay and 1.0 will apply maximum decay value resetting a pixel to neutral potential at each generation (default behavior).

Following sample show the use of `dv::EdgeMapAccumulator` with `dv::EventStreamSlicer` and `dv::io::camera::open()` to generate stream of edge images:

C++

Python

```

1  #include <dv-processing/core/frame.hpp>
2  #include <dv-processing/io/camera/discovery.hpp>
3
4  #include <opencv2/highgui.hpp>
5
6  int main() {
7      using namespace std::chrono_literals;
8
9      // Open any camera
10     auto capture = dv::io::camera::open();
11
12     // Make sure it supports event stream output, throw an error otherwise
13     if (!capture->isEventStreamAvailable()) {
14         throw dv::exceptions::RuntimeError("Input camera does not provide an event_
↪ stream.");
15     }
16
17     // Initialize an accumulator with some resolution
18     dv::EdgeMapAccumulator accumulator(*capture->getEventResolution());
19
20     // Apply configuration, these values can be modified to taste
21     accumulator.setNeutralPotential(0.0f);
22     accumulator.setEventContribution(0.25f);

```

(continues on next page)

(continued from previous page)

```

23     accumulator.setDecay(1.0);
24     accumulator.setIgnorePolarity(true);
25
26     // Initialize a preview window
27     cv::namedWindow("Preview", cv::WINDOW_NORMAL);
28
29     // Initialize a slicer
30     dv::EventStreamSlicer slicer;
31
32     // Register a callback every 33 milliseconds
33     slicer.doEveryTimeInterval(33ms, [&accumulator](const dv::EventStore &events) {
34         // Pass events into the accumulator and generate a preview frame
35         accumulator.accept(events);
36         dv::Frame frame = accumulator.generateFrame();
37
38         // Show the accumulated image
39         cv::imshow("Preview", frame.image);
40         cv::waitKey(2);
41     });
42
43     // Run the event processing while the camera is connected
44     while (capture->isRunning()) {
45         // Receive events, check if anything was received
46         if (const auto events = capture->getNextEventBatch()) {
47             // If so, pass the events into the slicer to handle them
48             slicer.accept(*events);
49         }
50     }
51
52     return 0;
53 }

```

```

1  from datetime import timedelta
2
3  import cv2 as cv
4  import dv_processing as dv
5
6  # Open any camera
7  capture = dv.io.camera.open()
8
9  # Make sure it supports event stream output, throw an error otherwise
10 if not capture.isEventStreamAvailable():
11     raise RuntimeError("Input camera does not provide an event stream.")
12
13 # Initialize an accumulator with some resolution
14 accumulator = dv.EdgeMapAccumulator(capture.getEventResolution())
15
16 # Apply configuration, these values can be modified to taste
17 accumulator.setNeutralPotential(0.0)
18 accumulator.setEventContribution(0.25)
19 accumulator.setNeutralPotential(1.0)
20 accumulator.setIgnorePolarity(True)

```

(continues on next page)

(continued from previous page)

```

21
22 # Initialize a preview window
23 cv.namedWindow("Preview", cv.WINDOW_NORMAL)
24
25 # Initialize a slicer
26 slicer = dv.EventStreamSlicer()
27
28
29 # Declare the callback method for slicer
30 def slicing_callback(events: dv.EventStore):
31     # Pass events into the accumulator and generate a preview frame
32     accumulator.accept(events)
33     frame = accumulator.generateFrame()
34
35     # Show the accumulated image
36     cv.imshow("Preview", frame.image)
37     cv.waitKey(2)
38
39
40 # Register callback to be performed every 33 milliseconds
41 slicer.doEveryTimeInterval(timedelta(milliseconds=33), slicing_callback)
42
43 # Run the event processing while the camera is connected
44 while capture.isRunning():
45     # Receive events
46     events = capture.getNextEventBatch()
47
48     # Check if anything was received
49     if events is not None:
50         # If so, pass the events into the slicer to handle them
51         slicer.accept(events)

```

3.3.4 Event visualization

Accumulators, described in previous chapters, are useful when image or edge representation of events is needed, and they are mostly useful to process events using typical image processing algorithms. `dv::visualization::EventVisualizer` class serves a purpose to perform simple event visualization. Instead of increasing or decreasing pixel brightness, the `dv::visualization::EventVisualizer` just performs color coding of pixel coordinates where an event was registered.

The following sample shows the use of `dv::visualization::EventVisualizer` class to generate colored previews of events using `dv::EventStreamSlicer` and `dv::io::camera::open()`:

C++

Python

```

1 #include <dv-processing/io/camera/discovery.hpp>
2 #include <dv-processing/visualization/event_visualizer.hpp>
3
4 #include <opencv2/highgui.hpp>
5
6 int main() {

```

(continues on next page)



Fig. 4: Frame generated using `dv::EdgeMapAccumulator` class.

(continued from previous page)

```

7   using namespace std::chrono_literals;
8
9   // Open any camera
10  auto capture = dv::io::camera::open();
11
12  // Make sure it supports event stream output, throw an error otherwise
13  if (!capture->isEventStreamAvailable()) {
14      throw dv::exceptions::RuntimeError("Input camera does not provide an event_
15  ↪ stream.");
16  }
17
18  // Initialize an accumulator with some resolution
19  dv::visualization::EventVisualizer visualizer(*capture->getEventResolution());
20
21  // Apply color scheme configuration, these values can be modified to taste
22  visualizer.setBackgroundColor(dv::visualization::colors::white);
23  visualizer.setPositiveColor(dv::visualization::colors::iniBlue);
24  visualizer.setNegativeColor(dv::visualization::colors::darkGray);
25
26  // Initialize a preview window
27  cv::namedWindow("Preview", cv::WINDOW_NORMAL);
28
29  // Initialize a slicer
30  dv::EventStreamSlicer slicer;
31
32  // Register a callback every 33 milliseconds
33  slicer.doEveryTimeInterval(33ms, [&visualizer](const dv::EventStore &events) {
34      // Generate a preview frame
35      cv::Mat image = visualizer.generateImage(events);
36
37      // Show the accumulated image
38      cv::imshow("Preview", image);
39      cv::waitKey(2);
40  });
41
42  // Run the event processing while the camera is connected
43  while (capture->isRunning()) {
44      // Receive events, check if anything was received
45      if (const auto events = capture->getNextEventBatch()) {
46          // If so, pass the events into the slicer to handle them
47          slicer.accept(*events);
48      }
49  }
50
51  return 0;

```

```

1  from datetime import timedelta
2
3  import cv2 as cv
4  import dv_processing as dv
5

```

(continues on next page)

(continued from previous page)

```

6  # Open any camera
7  capture = dv.io.camera.open()
8
9  # Make sure it supports event stream output, throw an error otherwise
10 if not capture.isEventStreamAvailable():
11     raise RuntimeError("Input camera does not provide an event stream.")
12
13 # Initialize an accumulator with some resolution
14 visualizer = dv.visualization.EventVisualizer(capture.getEventResolution())
15
16 # Apply color scheme configuration, these values can be modified to taste
17 visualizer.setBackgroundColor(dv.visualization.colors.white())
18 visualizer.setPositiveColor(dv.visualization.colors.iniBlue())
19 visualizer.setNegativeColor(dv.visualization.colors.darkGray())
20
21 # Initialize a preview window
22 cv.namedWindow("Preview", cv.WINDOW_NORMAL)
23
24 # Initialize a slicer
25 slicer = dv.EventStreamSlicer()
26
27
28 # Declare the callback method for slicer
29 def slicing_callback(events: dv.EventStore):
30     # Generate a preview frame
31     frame = visualizer.generateImage(events)
32
33     # Show the accumulated image
34     cv.imshow("Preview", frame)
35     cv.waitKey(2)
36
37
38 # Register callback to be performed every 33 milliseconds
39 slicer.doEveryTimeInterval(timedelta(milliseconds=33), slicing_callback)
40
41 # Run the event processing while the camera is connected
42 while capture.isRunning():
43     # Receive events
44     events = capture.getNextEventBatch()
45
46     # Check if anything was received
47     if events is not None:
48         # If so, pass the events into the slicer to handle them
49         slicer.accept(events)

```

3.3.5 Time surface

A time surface is an event representation as an image frame, except instead of representing pixel brightness, it represents an event's timestamp.

The timestamp representations can be normalized to retrieve an image representation of the time surface. It will represent the latest timestamps with the brightest pixel values.

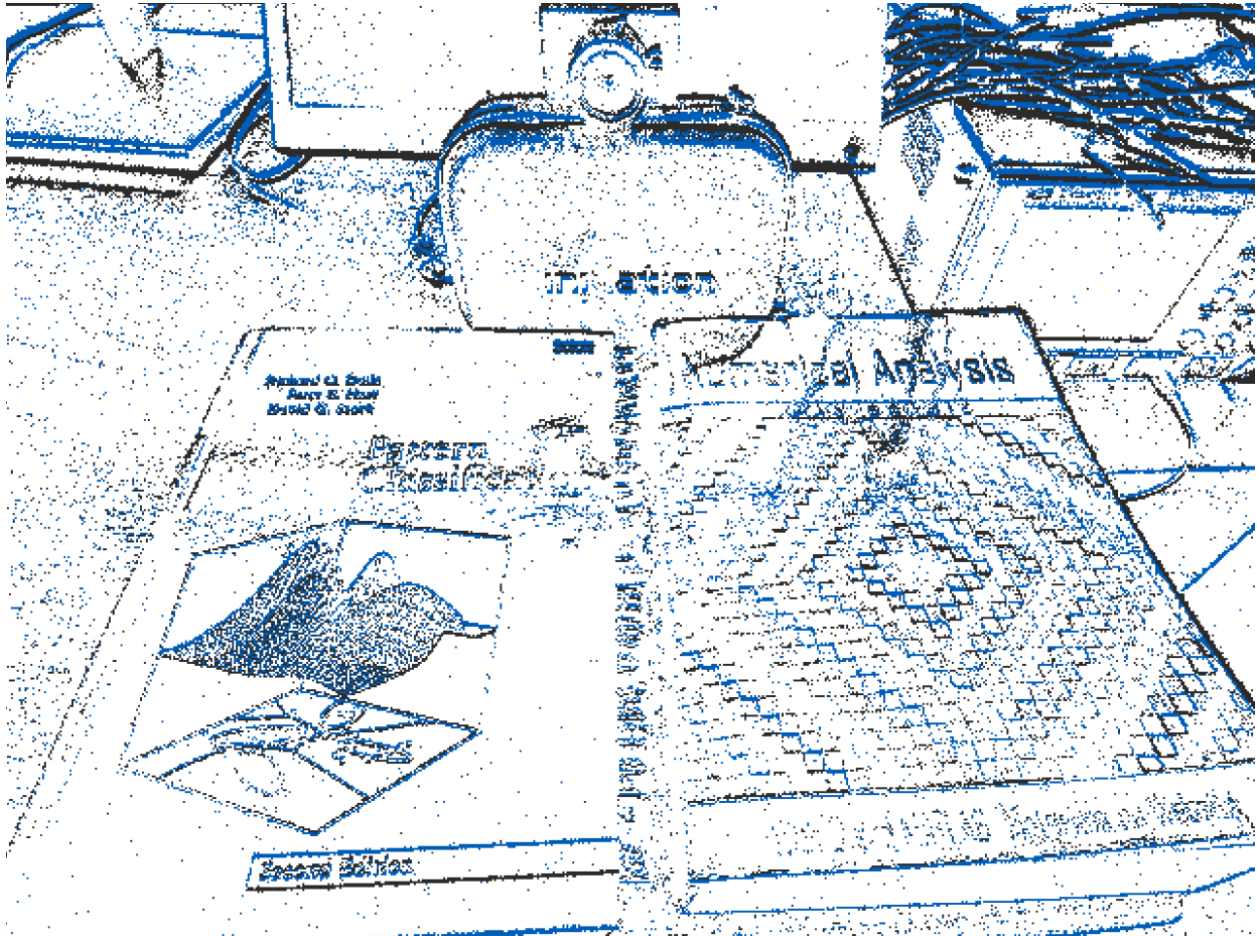


Fig. 5: Frame generated using `dv::visualization::EventVisualizer` class.

Following sample show the use of `dv::TimeSurface` class to generate time surface previews of events using `dv::EventStreamSlicer` and `dv::io::camera::open()`:

C++

Python

```

1  #include <dv-processing/core/frame.hpp>
2  #include <dv-processing/io/camera/discovery.hpp>
3
4  #include <opencv2/highgui.hpp>
5
6  int main() {
7      using namespace std::chrono_literals;
8
9      // Open any camera
10     auto capture = dv::io::camera::open();
11
12     // Make sure it supports event stream output, throw an error otherwise
13     if (!capture->isEventStreamAvailable()) {
14         throw dv::exceptions::RuntimeError("Input camera does not provide an event_
15         ↪stream.");
16     }
17
18     // Initialize an accumulator with camera sensor resolution
19     dv::TimeSurface surface(*capture->getEventResolution());
20
21     // Initialize a preview window
22     cv::namedWindow("Preview", cv::WINDOW_NORMAL);
23
24     // Initialize a slicer
25     dv::EventStreamSlicer slicer;
26
27     // Register a callback every 33 milliseconds
28     slicer.doEveryTimeInterval(33ms, [&surface](const dv::EventStore &events) {
29         // Pass the events to update the time surface
30         surface.accept(events);
31
32         // Generate a preview frame
33         dv::Frame frame = surface.generateFrame();
34
35         // Show the accumulated image
36         cv::imshow("Preview", frame.image);
37         cv::waitKey(2);
38     });
39
40     // Run the event processing while the camera is connected
41     while (capture->isRunning()) {
42         // Receive events, check if anything was received
43         if (const auto events = capture->getNextEventBatch()) {
44             // If so, pass the events into the slicer to handle them
45             slicer.accept(*events);
46         }
47     }
48 }

```

(continues on next page)

(continued from previous page)

```

47     return 0;
48 }
49

```

```

1  from datetime import timedelta
2
3  import cv2 as cv
4  import dv_processing as dv
5
6  # Open any camera
7  capture = dv.io.camera.open()
8
9  # Make sure it supports event stream output, throw an error otherwise
10 if not capture.isEventStreamAvailable():
11     raise RuntimeError("Input camera does not provide an event stream.")
12
13 # Initialize an accumulator with camera sensor resolution
14 surface = dv.TimeSurface(capture.getEventResolution())
15
16 # Initialize a preview window
17 cv.namedWindow("Preview", cv.WINDOW_NORMAL)
18
19 # Initialize a slicer
20 slicer = dv.EventStreamSlicer()
21
22
23 # Declare the callback method for slicer
24 def slicing_callback(events: dv.EventStore):
25     # Pass the events to update the time surface
26     surface.accept(events)
27
28     # Generate a preview frame
29     frame = surface.generateFrame()
30
31     # Show the accumulated image
32     cv.imshow("Preview", frame.image)
33     cv.waitKey(2)
34
35
36 # Register callback to be performed every 33 milliseconds
37 slicer.doEveryTimeInterval(timedelta(milliseconds=33), slicing_callback)
38
39 # Run the event processing while the camera is connected
40 while capture.isRunning():
41     # Receive events
42     events = capture.getNextEventBatch()
43
44     # Check if anything was received
45     if events is not None:
46         # If so, pass the events into the slicer to handle them
47         slicer.accept(events)

```



Fig. 6: Frame generated using `dv:TimeSurface` class.

3.3.6 Speed invariant time surface

Speed invariant time surface is a specific time surface variant that is more suitable for feature extraction, the implementation follows [this paper](#)⁷.

Following sample show the use of `dv::SpeedInvariantTimeSurface` class to generate time surface previews of events using `dv::EventStreamSlicer` and `dv::io::camera::open()`:

C++

Python

```

1  #include <dv-processing/io/camera/discovery.hpp>
2
3  #include <opencv2/highgui.hpp>
4
5  int main() {
6      using namespace std::chrono_literals;
7
8      // Open any camera
9      auto capture = dv::io::camera::open();
10
11     // Make sure it supports event stream output, throw an error otherwise
12     if (!capture->isEventStreamAvailable()) {
13         throw dv::exceptions::RuntimeError("Input camera does not provide an event_
14         stream.");
15     }
16
17     // Initialize an accumulator with camera sensor resolution
18     dv::SpeedInvariantTimeSurface surface(*capture->getEventResolution());
19
20     // Initialize a preview window
21     cv::namedWindow("Preview", cv::WINDOW_NORMAL);
22
23     // Initialize a slicer
24     dv::EventStreamSlicer slicer;
25
26     // Register a callback every 33 milliseconds
27     slicer.doEveryTimeInterval(33ms, [&surface](const dv::EventStore &events) {
28         // Pass the events to update the time surface
29         surface.accept(events);
30
31         // Generate a preview frame
32         dv::Frame frame = surface.generateFrame();
33
34         // Show the accumulated image
35         cv::imshow("Preview", frame.image);
36         cv::waitKey(2);
37     });
38
39     // Run the event processing while the camera is connected
40     while (capture->isRunning()) {
41         // Receive events, check if anything was received
42         if (const auto events = capture->getNextEventBatch()) {

```

(continues on next page)

⁷ <https://arxiv.org/pdf/1903.11332.pdf>

(continued from previous page)

```

42         // If so, pass the events into the slicer to handle them
43         slicer.accept(*events);
44     }
45 }
46
47 return 0;
48 }

```

```

1  from datetime import timedelta
2
3  import cv2 as cv
4  import dv_processing as dv
5
6  # Open any camera
7  capture = dv.io.camera.open()
8
9  # Make sure it supports event stream output, throw an error otherwise
10 if not capture.isEventStreamAvailable():
11     raise RuntimeError("Input camera does not provide an event stream.")
12
13 # Initialize an accumulator with camera sensor resolution
14 surface = dv.SpeedInvariantTimeSurface(capture.getEventResolution())
15
16 # Initialize a preview window
17 cv.namedWindow("Preview", cv.WINDOW_NORMAL)
18
19 # Initialize a slicer
20 slicer = dv.EventStreamSlicer()
21
22
23 # Declare the callback method for slicer
24 def slicing_callback(events: dv.EventStore):
25     # Pass the events to update the time surface
26     surface.accept(events)
27
28     # Generate a preview frame
29     frame = surface.generateFrame()
30
31     # Show the accumulated image
32     cv.imshow("Preview", frame.image)
33     cv.waitKey(2)
34
35
36 # Register callback to be performed every 33 milliseconds
37 slicer.doEveryTimeInterval(timedelta(milliseconds=33), slicing_callback)
38
39 # Run the event processing while the camera is connected
40 while capture.isRunning():
41     # Receive events
42     events = capture.getNextEventBatch()
43
44     # Check if anything was received

```

(continues on next page)

(continued from previous page)

```

45  if events is not None:
46      # If so, pass the events into the slicer to handle them
47      slicer.accept(events)

```

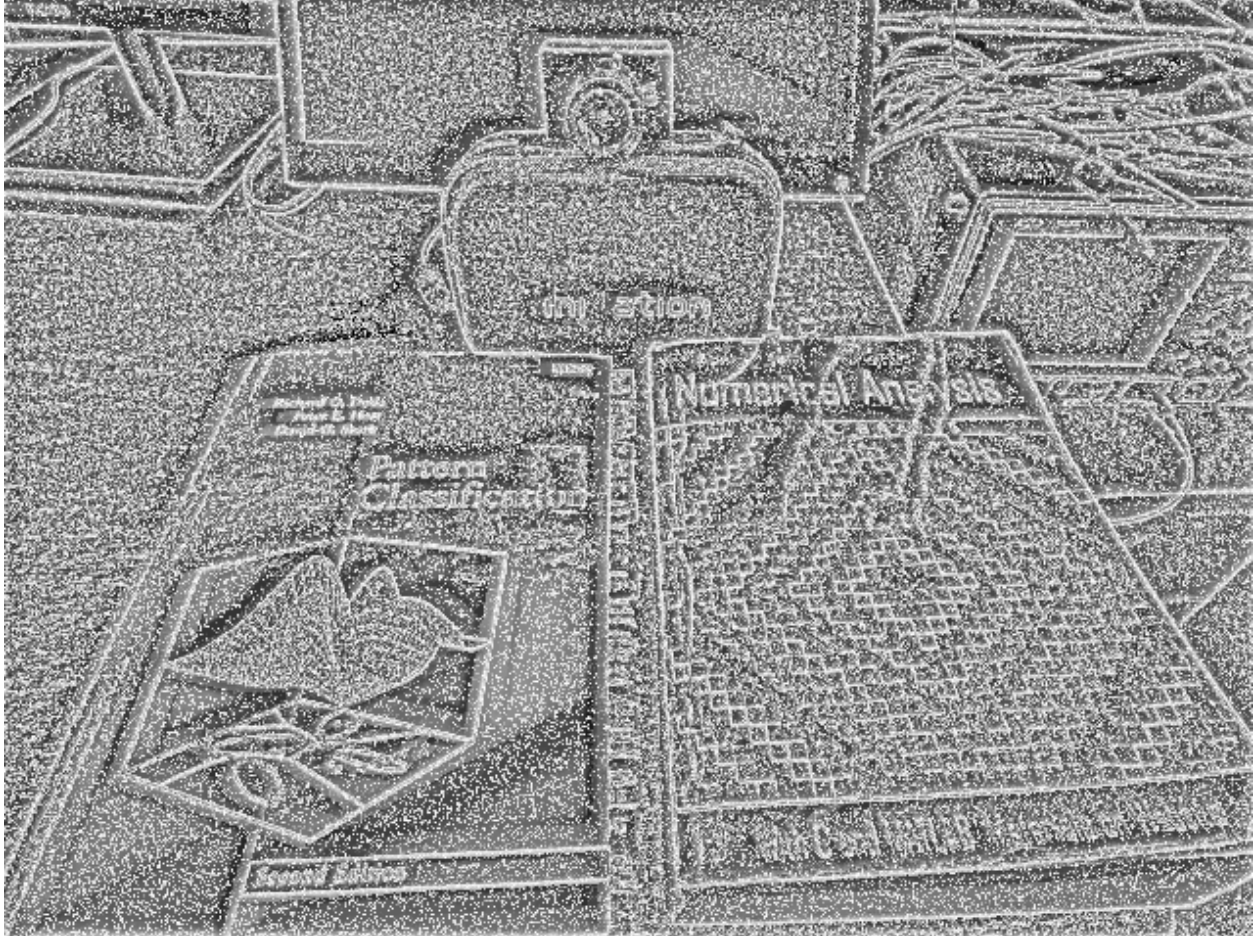


Fig. 7: Frame generated using `dv::SpeedInvariantTimeSurface` class.

3.3.7 Performance of available accumulators

The library performs benchmarking of available accumulation algorithms to ensure their best performance. Accumulators are also benchmarked on two metrics:

- Event throughput - measured in millions of (mega) events per second;
- Framerate - measured in generated frames per second.

The benchmarks are performed by generating a batch of events at uniformly random pixel coordinates on a VGA (640x480) resolution. Below are the results of running the benchmark on AMD Ryzen 7 3800X 8-Core Processor:

Accumulator type	Framerate (FPS)	Throughput (MegaEvent/s)
Accumulator	668	66.5
EdgeMapAccumulator	1767	149.1
EventVisualizer	785	78.3
TimeSurface	910	91.5
SpeedInvariantTimeSurface	370	36.8

3.4 Filtering events

Event stream coming from a camera can contain noise, which is caused by analog electrical circuitry used to compare brightness on each pixel. The dv-processing library provides algorithms that filter noise efficiently on an event stream. Additionally, filtering can be used to subsample events in region of interest, filter by polarity, apply masks to events. This tutorial covers available filter implementations and how to use them efficiently.

3.4.1 Implementation of filters

The library provides two main types of filters - noise and subsampling. Subsampling filters include polarity filters, region of interest, mask filters. The library provides two algorithms for filtering noise that can be found under namespace `dv::noise`. Below is a class hierarchy diagram for available noise filters:

All filter have a common programming pattern:

- Events are added to the filter instance using overloaded `dv::EventFilterBase::accept()` method.
- Input events are filtered and returned by calling overloaded `dv::EventFilterBase::generateEvents()` method.
- Internally events are filtered by calling a filter `retain()` method on each event and discarding events if a `false` is returned.

3.4.2 Noise filtering

This chapter describes the available event noise filters in the library.

Background activity noise filter

`dv::noise::BackgroundActivityNoiseFilter` - events are filtered based on short-term local neighborhood activity. If an event is “supported” by another event registered at local pixel neighborhood, that event is not considered noise and is added to the output events.

Following sample code shows the usage of the `dv::noise::BackgroundActivityNoiseFilter` to filter noise:

C++

Python

```
1 #include <dv-processing/data/generate.hpp>
2 #include <dv-processing/noise/background_activity_noise_filter.hpp>
3 #include <dv-processing/visualization/event_visualizer.hpp>
4
5 #include <opencv2/highgui.hpp>
6
7 int main() {
8     using namespace std::chrono_literals;
```

(continues on next page)

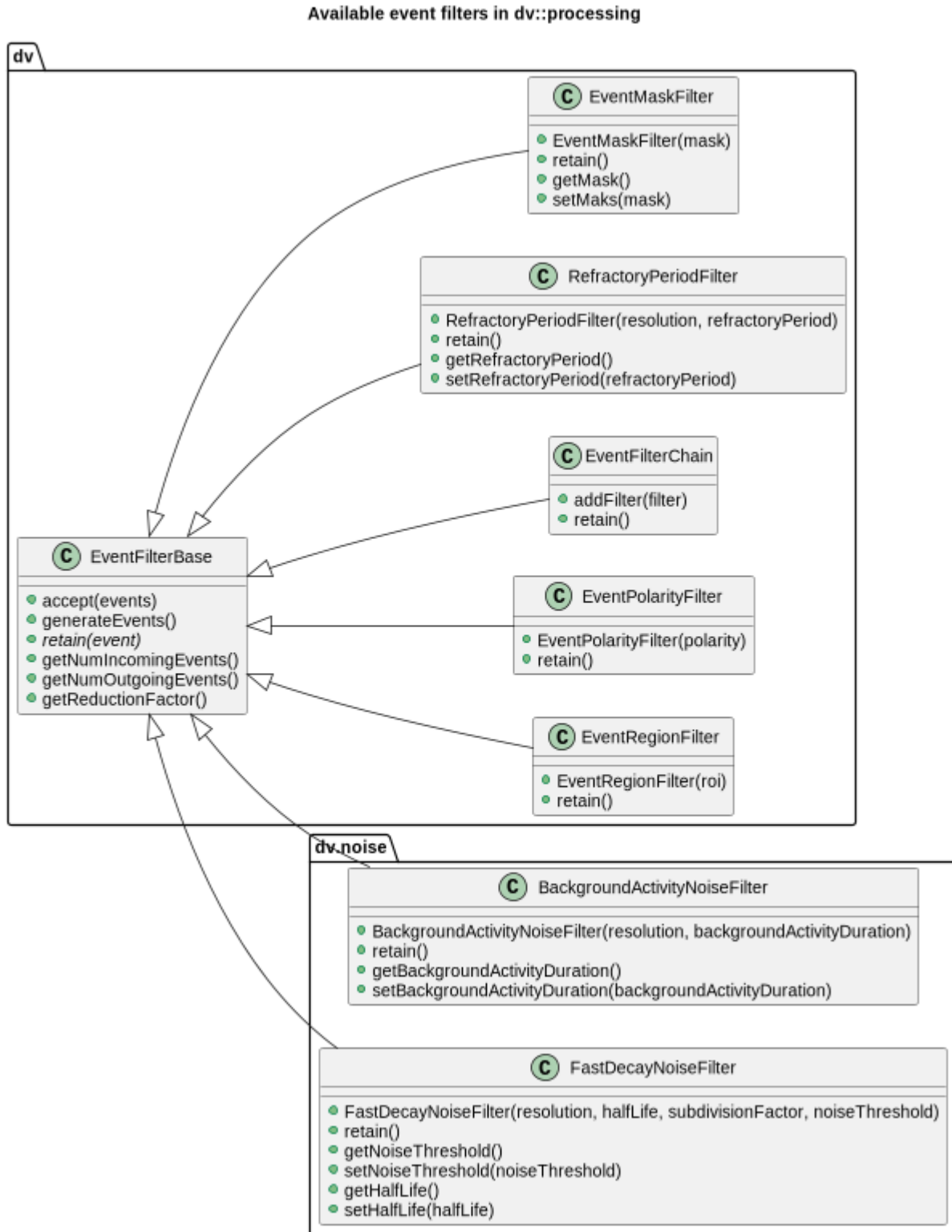


Fig. 8: Available filters and their hierarchy in the library.

(continued from previous page)

```

9
10 // Hardcoded VGA resolution
11 const cv::Size resolution(640, 480);
12
13 // Initializing input events with uniformly distributed events which represent
14 ↪noise
15     dv::EventStore events = dv::data::generate::uniformEventsWithinTimeRange(0, 10ms,
16 ↪resolution, 1000);
17
18 // Adding additional data for drawing, this will give an idea whether the filter
19 ↪removes actual signal events
20     events.add(dv::data::generate::dvLogoAsEvents(10000, resolution));
21
22 // Initialize a background activity noise filter with 1-millisecond activity
23 ↪period
24     dv::noise::BackgroundActivityNoiseFilter filter(resolution, 1ms);
25
26 // Pass events to the filter
27     filter.accept(events);
28
29 // Call generate events to apply the noise filter
30     const dv::EventStore filtered = filter.generateEvents();
31
32 // Print out the reduction factor, which indicates the percentage of discarded
33 ↪events
34     std::cout << "Filter reduced number of events by a factor of " << filter.
35 ↪getReductionFactor() << std::endl;
36
37 // Use a visualizer instance to preview the events
38     dv::visualization::EventVisualizer visualizer(resolution);
39
40 // Generate preview images of data input and output
41     const cv::Mat input = visualizer.generateImage(events);
42     const cv::Mat output = visualizer.generateImage(filtered);
43
44 // Concatenate the images into a single image for preview
45     cv::Mat preview;
46     cv::hconcat(input, output, preview);
47
48 // Display the input and output images
49     cv::namedWindow("preview", cv::WINDOW_NORMAL);
50     cv::imshow("preview", preview);
51     cv::waitKey();
52
53     return 0;
54 }

```

```

1 import dv_processing as dv
2 import cv2 as cv
3 from datetime import timedelta
4
5 # Hardcoded VGA resolution

```

(continues on next page)

(continued from previous page)

```

6 resolution = (640, 480)
7
8 # Initializing input events with uniformly distributed events which represent noise
9 events = dv.data.generate.uniformEventsWithinTimeRange(0, timedelta(milliseconds=10), ↵
    ↵resolution, 1000)
10
11 # Adding additional data for drawing, this will give an idea whether the filter ↵
    ↵removes actual signal events
12 events.add(dv.data.generate.dvLogoAsEvents(10000, resolution))
13
14 # Initialize a background activity noise filter with 1-millisecond activity period
15 filter = dv.noise.BackgroundActivityNoiseFilter(resolution, ↵
    ↵backgroundActivityDuration=timedelta(milliseconds=1))
16
17 # Pass events to the filter
18 filter.accept(events)
19
20 # Call generate events to apply the noise filter
21 filtered = filter.generateEvents()
22
23 # Print out the reduction factor, which indicates the percentage of discarded events
24 print(f"Filter reduced number of events by a factor of {filter.getReductionFactor()}")
25
26 # Use a visualizer instance to preview the events
27 visualizer = dv.visualization.EventVisualizer(resolution)
28
29 # Generate preview images of data input and output
30 input = visualizer.generateImage(events)
31 output = visualizer.generateImage(filtered)
32
33 # Concatenate the images into a single image for preview
34 preview = cv.hconcat([input, output])
35
36 # Display the input and output images
37 cv.namedWindow("preview", cv.WINDOW_NORMAL)
38 cv.imshow("preview", preview)
39 cv.waitKey()

```

The sample code outputs such images:

Note

The image above that the filter reduces the amount of speckles on white area, but maintains the logo preview, those events are not filtered out.

Fast decay noise filter

`dv::noise::FastDecayNoiseFilter` - events are filtered based on lower resolution fast-decaying representation of events. Events contribute to a low-resolution accumulated image with a fast decay, which also represents local activity. Unlike the `BackgroundActivityNoiseFilter`, this filter uses decay instead of a hard time threshold, although the approach is very similar - an event needs to be supported by another event in a local pixel neighborhood. This filter has a lower memory footprint since the neighborhood is represented in a low resolution accumulated image.

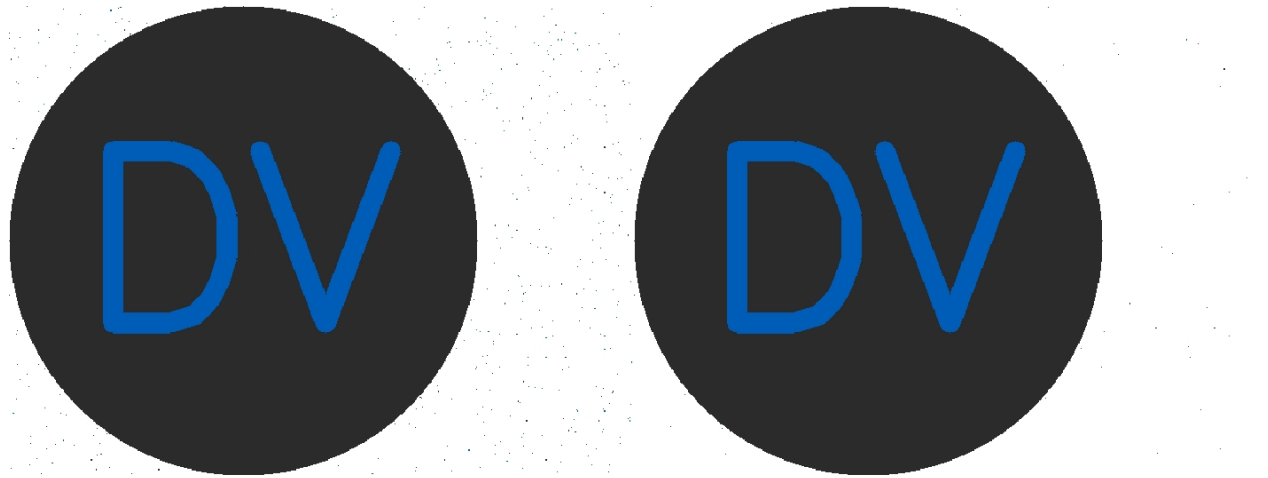


Fig. 9: Output of the sample use of background activity filter. Left is a preview of input events, right is a preview of filtered events.

Following sample code shows the usage of the `dv::noise::FastDecayNoiseFilter` to filter noise:

C++

Python

```

1  #include <dv-processing/data/generate.hpp>
2  #include <dv-processing/noise/fast_decay_noise_filter.hpp>
3  #include <dv-processing/visualization/event_visualizer.hpp>
4
5  #include <opencv2/highgui.hpp>
6
7  int main() {
8      using namespace std::chrono_literals;
9
10     // Hardcoded VGA resolution
11     const cv::Size resolution(640, 480);
12
13     // Initializing input events with uniformly distributed events which represent_
    ↪ noise
14     dv::EventStore events = dv::data::generate::uniformEventsWithinTimeRange(0, 10ms, _
    ↪ resolution, 1000);
15
16     // Adding additional data for drawing, this will give an idea whether the filter_
    ↪ removes actual signal events
17     events.add(dv::data::generate::dvLogoAsEvents(10000, resolution));
18
19     // Initialize a background activity noise filter with 10-millisecond half life_
    ↪ decay, resolution subdivision
20     // factor of 4 and noise threshold of 1. Half life decay and noise threshold_
    ↪ values controls the quality of
21     // filtering, while subdivision factor is used for resolution downsizing for_
    ↪ internal event representation.
22     dv::noise::FastDecayNoiseFilter filter(resolution, 10ms, 4, 1.f);
23

```

(continues on next page)

(continued from previous page)

```

24 // Pass events to the filter
25 filter.accept(events);
26
27 // Call generate events to apply the noise filter
28 const dv::EventStore filtered = filter.generateEvents();
29
30 // Print out the reduction factor, which indicates the percentage of discarded
  ↪ events
31 std::cout << "Filter reduced number of events by a factor of " << filter.
  ↪ getReductionFactor() << std::endl;
32
33 // Use a visualizer instance to preview the events
34 dv::visualization::EventVisualizer visualizer(resolution);
35
36 // Generate preview images of data input and output
37 const cv::Mat input = visualizer.generateImage(events);
38 const cv::Mat output = visualizer.generateImage(filtered);
39
40 // Concatenate the images into a single image for preview
41 cv::Mat preview;
42 cv::hconcat(input, output, preview);
43
44 // Display the input and output images
45 cv::namedWindow("preview", cv::WINDOW_NORMAL);
46 cv::imshow("preview", preview);
47 cv::waitKey();
48
49 return 0;
50 }

```

```

1 import dv_processing as dv
2 import cv2 as cv
3 from datetime import timedelta
4
5 # Hardcoded VGA resolution
6 resolution = (640, 480)
7
8 # Initializing input events with uniformly distributed events which represent noise
9 events = dv.data.generate.uniformEventsWithinTimeRange(0, timedelta(milliseconds=10),
  ↪ resolution, 1000)
10
11 # Adding additional data for drawing, this will give an idea whether the filter
  ↪ removes actual signal events
12 events.add(dv.data.generate.dvLogoAsEvents(10000, resolution))
13
14 # Initialize a background activity noise filter with 10-millisecond half life decay,
  ↪ resolution subdivision
15 # factor of 4 and noise threshold of 1. Half life decay and noise threshold values
  ↪ controls the quality of
16 # filtering, while subdivision factor is used for resolution downsizing for internal
  ↪ event representation.
17 filter = dv.noise.FastDecayNoiseFilter(resolution,

```

(continues on next page)

(continued from previous page)

```

18         halfLife=timedelta(milliseconds=10),
19         subdivisionFactor=4,
20         noiseThreshold=1.0)
21
22     # Pass events to the filter
23     filter.accept(events)
24
25     # Call generate events to apply the noise filter
26     filtered = filter.generateEvents()
27
28     # Print out the reduction factor, which indicates the percentage of discarded events
29     print(f"Filter reduced number of events by a factor of {filter.getReductionFactor()}")
30
31     # Use a visualizer instance to preview the events
32     visualizer = dv.visualization.EventVisualizer(resolution)
33
34     # Generate preview images of data input and output
35     input = visualizer.generateImage(events)
36     output = visualizer.generateImage(filtered)
37
38     # Concatenate the images into a single image for preview
39     preview = cv.hconcat([input, output])
40
41     # Display the input and output images
42     cv.namedWindow("preview", cv.WINDOW_NORMAL)
43     cv.imshow("preview", preview)
44     cv.waitKey()

```

The sample code outputs such images:

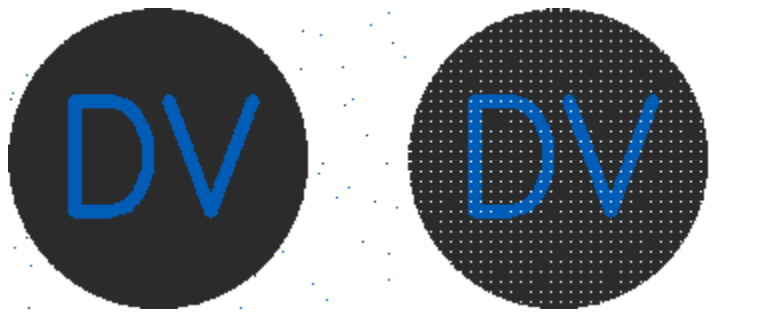


Fig. 10: Output of the sample use of fast decay noise filter. Left is a preview of input events, right is a preview of filtered events.

Note

The image above that the filter drastically the amount of speckles on white area, although the logo image is also affected and some true-signal events contributing to the logo are also filtered out.

3.4.3 Event subsampling

The same filtering approach is used to subsampling events based on their pixel location, polarity or other properties. This chapter describes the available event subsampling filters in the library.

Mask filter

`dv::EventMaskFilter` - filters events based on a pixel mask. Events are discarded in pixel locations where mask has zero values.

Following sample code shows the usage of the `dv::EventMaskFilter` to filter out selected regions of events:

C++

Python

```

1  #include <dv-processing/core/filters.hpp>
2  #include <dv-processing/data/generate.hpp>
3  #include <dv-processing/visualization/event_visualizer.hpp>
4
5  #include <opencv2/highgui.hpp>
6
7  int main() {
8      using namespace std::chrono_literals;
9
10     // Smaller resolution for previews
11     const cv::Size resolution(200, 200);
12
13     // Initializing input events with events that represent a logo
14     const dv::EventStore events = dv::data::generate::dvLogoAsEvents(0, resolution);
15
16     // Initialize a mask with all zero values
17     cv::Mat mask(resolution, CV_8UC1, cv::Scalar(0));
18
19     // Draw two rectangles to generate a similar to checkerboard mask pattern
20     cv::rectangle(
21         mask, cv::Point(0, 0), cv::Point(resolution.width / 2, resolution.height / 2),
22         cv::Scalar(255), cv::FILLED);
23     cv::rectangle(mask, cv::Point(resolution.width / 2, resolution.height / 2),
24         cv::Point(resolution.width, resolution.height), cv::Scalar(255), cv::FILLED);
25
26     // Initialize the mask filter with the generated mask
27     dv::EventMaskFilter filter(mask);
28
29     // Pass events to the filter
30     filter.accept(events);
31
32     // Call generate events to apply the filter
33     const dv::EventStore filtered = filter.generateEvents();
34
35     // Print out the reduction factor, which indicates the percentage of discarded
36     // events
37     std::cout << "Filter reduced number of events by a factor of " << filter.
38     getReductionFactor() << std::endl;
39
40     // Use a visualizer instance to preview the events

```

(continues on next page)

(continued from previous page)

```

38     dv::visualization::EventVisualizer visualizer(resolution);
39
40     // Generate preview images of data input and output
41     const cv::Mat input = visualizer.generateImage(events);
42     const cv::Mat output = visualizer.generateImage(filtered);
43
44     // Concatenate the images into a single image for preview
45     cv::Mat preview, maskColored;
46     cv::cvtColor(mask, maskColored, cv::COLOR_GRAY2BGR);
47     cv::hconcat(std::vector<cv::Mat>({input, maskColored, output}), preview);
48
49     // Display the input and output images
50     cv::namedWindow("preview", cv::WINDOW_NORMAL);
51     cv::imshow("preview", preview);
52     cv::waitKey();
53
54     return 0;
55 }

```

```

1  import dv_processing as dv
2  import cv2 as cv
3  import numpy as np
4
5  # Smaller resolution for previews
6  resolution = (200, 200)
7
8  # Initializing input events with events that represent a logo
9  events = dv.data.generate.dvLogoAsEvents(0, resolution)
10
11 # Initialize a mask with all zero values
12 mask = np.full(resolution, fill_value=0, dtype=np.uint8)
13
14 # Draw two rectangles to generate a similar to checkerboard mask pattern
15 cv.rectangle(mask, [0, 0], [int(resolution[0] / 2), int(resolution[1] / 2)], (255, ), ↵
↵ cv.FILLED)
16 cv.rectangle(mask, [int(resolution[0] / 2), int(resolution[1] / 2)], resolution, (255,
↵ ), cv.FILLED)
17
18 # Initialize the mask filter with the generated mask
19 filter = dv.EventMaskFilter(mask)
20
21 # Pass events to the filter
22 filter.accept(events)
23
24 # Call generate events to apply the filter
25 filtered = filter.generateEvents()
26
27 # Print out the reduction factor, which indicates the percentage of discarded events
28 print(f"Filter reduced number of events by a factor of {filter.getReductionFactor()}")
29
30 # Use a visualizer instance to preview the events
31 visualizer = dv.visualization.EventVisualizer(resolution)

```

(continues on next page)

(continued from previous page)

```

32
33 # Generate preview images of data input and output
34 input = visualizer.generateImage(events)
35 output = visualizer.generateImage(filtered)
36
37 # Concatenate the images into a single image for preview
38 preview = cv.hconcat([input, cv.cvtColor(mask, cv.COLOR_GRAY2BGR), output])
39
40 # Display the input and output images
41 cv.namedWindow("preview", cv.WINDOW_NORMAL)
42 cv.imshow("preview", preview)
43 cv.waitKey()

```

The sample code outputs such images:



Fig. 11: Output of the sample use of event mask filter. Left is a preview of input events, middle is the mask used, and right is a preview of filtered events.

Frequency filters

`dv::noise::LowPassFilter` - low-pass frequency filter discards high frequency events at repeating pixel locations. This is also commonly referred to as refractory period filter, where the refractory period is the inverse of the cutoff frequency. Each event timestamp is compared against most recent event timestamp on the same pixel location, if the frequency is higher than the cutoff frequency, the event is discarded.

Similarly, `dv-processing` implements similar frequency filters such as `dv::noise::HighPassFilter`, `dv::noise::BandPassFilter`, and `dv::noise::BandCutFilter` to discard events below a given frequency, outside a frequency band, or inside a frequency band, respectively.

Following sample code shows the usage of the `dv::noise::LowPassFilter` to filter out high frequency events on the same coordinate location:

C++

Python

```

1 #include <dv-processing/data/generate.hpp>
2 #include <dv-processing/noise/frequency_filters.hpp>
3
4 #include <opencv2/highgui.hpp>
5
6 int main() {
7     using namespace std::chrono_literals;
8
9     const cv::Size resolution(200, 200);

```

(continues on next page)

(continued from previous page)

```

10  const float frequency = 2000.0f;
11
12  // Initializing 10000 events that are uniformly spaced in pixel area and time
13  dv::EventStore events = dv::data::generate::uniformEventsWithinTimeRange(
14      0, dv::Duration(static_cast<int64_t>(1e6f / frequency)), resolution, 10000);
15
16  // Initialize low-pass frequency filter with given cutoff frequency
17  dv::noise::LowPassFilter filter(resolution, frequency);
18
19  // Pass events to the filter
20  filter.accept(events);
21
22  // Call generate events to apply the filter
23  const dv::EventStore filtered = filter.generateEvents();
24
25  // Print out the number of events after filtering
26  std::cout << "Filtered [" << filtered.size() << "]" events out of [" << events.
27  ↪size() << "]" << std::endl;
28
29  return 0;

```

```

1  import dv_processing as dv
2  import cv2 as cv
3  from datetime import timedelta
4
5  resolution = (200, 200)
6  frequency = 2000
7  # Initializing 10000 events that are uniformly spaced in pixel area and time
8  events = dv.data.generate.uniformEventsWithinTimeRange(0, timedelta(milliseconds=1e3 /
9  ↪ frequency), resolution, 10000)
10
11 # Initialize low-pass frequency filter with given cutoff frequency
12 filter = dv.noise.LowPassFilter(resolution, frequency)
13
14 # Pass events to the filter
15 filter.accept(events)
16
17 # Call generate events to apply the filter
18 filtered = filter.generateEvents()
19
20 # Print out the number of events after filtering
21 print(f"Filtered [{len(filtered)}] events out of [{len(events)}]")

```

Polarity filter

`dv::EventPolarityFilter` - filter events based on polarity.

Following sample code shows the usage of the `dv::EventPolarityFilter` to filter out events based on polarity:

C++

Python

```

1  #include <dv-processing/core/filters.hpp>
2  #include <dv-processing/data/generate.hpp>
3  #include <dv-processing/visualization/event_visualizer.hpp>
4
5  #include <opencv2/highgui.hpp>
6
7  int main() {
8      using namespace std::chrono_literals;
9
10     const cv::Size resolution(200, 200);
11
12     // Initializing input events with events that represent a logo
13     dv::EventStore events = dv::data::generate::dvLogoAsEvents(0, resolution);
14
15     // Filter positive polarity events only
16     dv::EventPolarityFilter filter(true);
17
18     // Pass events to the filter
19     filter.accept(events);
20
21     // Call generate events to apply the filter
22     const dv::EventStore filtered = filter.generateEvents();
23
24     // Use a visualizer instance to preview the events
25     dv::visualization::EventVisualizer visualizer(resolution);
26
27     // Generate preview images of data input and output
28     const cv::Mat input = visualizer.generateImage(events);
29     const cv::Mat output = visualizer.generateImage(filtered);
30
31     // Concatenate the images into a single image for preview
32     cv::Mat preview;
33     cv::hconcat(input, output, preview);
34
35     // Display the input and output images
36     cv::namedWindow("preview", cv::WINDOW_NORMAL);
37     cv::imshow("preview", preview);
38     cv::waitKey();
39
40     return 0;
41 }

```

```

1  import dv_processing as dv
2  import cv2 as cv
3  from datetime import timedelta
4
5  resolution = (200, 200)
6
7  # Initializing input events with events that represent a logo
8  events = dv.data.generate.dvLogoAsEvents(0, resolution)
9
10 # Filter positive polarity events only

```

(continues on next page)

(continued from previous page)

```
11 filter = dv.EventPolarityFilter(True)
12
13 # Pass events to the filter
14 filter.accept(events)
15
16 # Call generate events to apply the filter
17 filtered = filter.generateEvents()
18
19 # Use a visualizer instance to preview the events
20 visualizer = dv.visualization.EventVisualizer(resolution)
21
22 # Generate preview images of data input and output
23 input = visualizer.generateImage(events)
24 output = visualizer.generateImage(filtered)
25
26 # Concatenate the images into a single image for preview
27 preview = cv.hconcat([input, output])
28
29 # Display the input and output images
30 cv.namedWindow("preview", cv.WINDOW_NORMAL)
31 cv.imshow("preview", preview)
32 cv.waitKey()
```

The sample code outputs such images:



Fig. 12: Output of the sample use of polarity filter. Left is a preview of input events and right is a preview of filtered events.

Note

The generated DV logo image comes from event representation, white area means no events are there, dark gray area are negative events and pixels are blue on coordinates where positive polarity events are provided. The filtered image only contains the letters “DV” since the background circle is represented by negative polarity events.

Event region filter

`dv::EventRegionFilter` - filter events based on given region of interest.

Following sample code shows the usage of the `dv::EventRegionFilter` to filter out specific area of events:

C++

Python

```

1  #include <dv-processing/core/filters.hpp>
2  #include <dv-processing/data/generate.hpp>
3  #include <dv-processing/visualization/event_visualizer.hpp>
4
5  #include <opencv2/highgui.hpp>
6
7  int main() {
8      using namespace std::chrono_literals;
9
10     const cv::Size resolution(200, 200);
11
12     // Initializing input events with events that represent a logo
13     dv::EventStore events = dv::data::generate::dvLogoAsEvents(0, resolution);
14
15     // Initialize region filter using hardcoded coordinates
16     dv::EventRegionFilter filter(cv::Rect(50, 50, 100, 100));
17
18     // Pass events to the filter
19     filter.accept(events);
20
21     // Call generate events to apply the filter
22     const dv::EventStore filtered = filter.generateEvents();
23
24     // Use a visualizer instance to preview the events
25     dv::visualization::EventVisualizer visualizer(resolution);
26
27     // Generate preview images of data input and output
28     const cv::Mat input = visualizer.generateImage(events);
29     const cv::Mat output = visualizer.generateImage(filtered);
30
31     // Concatenate the images into a single image for preview
32     cv::Mat preview;
33     cv::hconcat(input, output, preview);
34
35     // Display the input and output images
36     cv::namedWindow("preview", cv::WINDOW_NORMAL);
37     cv::imshow("preview", preview);
38     cv::waitKey();
39
40     return 0;
41 }

```

```

1  import dv_processing as dv
2  import cv2 as cv
3  from datetime import timedelta

```

(continues on next page)

(continued from previous page)

```
4
5 resolution = (200, 200)
6
7 # Initializing input events with events that represent a logo
8 events = dv.data.generate.dvLogoAsEvents(0, resolution)
9
10 # Initialize region filter using hardcoded coordinates
11 filter = dv.EventRegionFilter((50, 50, 100, 100))
12
13 # Pass events to the filter
14 filter.accept(events)
15
16 # Call generate events to apply the filter
17 filtered = filter.generateEvents()
18
19 # Use a visualizer instance to preview the events
20 visualizer = dv.visualization.EventVisualizer(resolution)
21
22 # Generate preview images of data input and output
23 input = visualizer.generateImage(events)
24 output = visualizer.generateImage(filtered)
25
26 # Concatenate the images into a single image for preview
27 preview = cv.hconcat([input, output])
28
29 # Display the input and output images
30 cv.namedWindow("preview", cv.WINDOW_NORMAL)
31 cv.imshow("preview", preview)
32 cv.waitKey()
```

The sample code outputs such images:



Fig. 13: Output of the sample use of region filter. Left is a preview of input events and right is a preview of filtered events.

Filter chain

Multiple filters can be combined into a single filter chain, which optimizes memory operations to increase the performance of applying multiple filters. This is achieved by using the `dv::EventFilterChain` class. Multiple filters can be added using `dv::EventFilterChain::addFilter()` method, it accepts filter wrapped in `std::shared_pointer`, the shared pointer is used to be able to modify the parameters of filters after they are added to the filter chain.

Following sample code shows the usage of the `dv::EventFilterChain` to apply multiple types of filters in a single chain:

C++

Python

```

1  #include <dv-processing/core/filters.hpp>
2  #include <dv-processing/data/generate.hpp>
3  #include <dv-processing/noise/background_activity_noise_filter.hpp>
4  #include <dv-processing/visualization/event_visualizer.hpp>
5
6  #include <opencv2/highgui.hpp>
7
8  int main() {
9      using namespace std::chrono_literals;
10
11     const cv::Size resolution(200, 200);
12
13     // Initializing input events with events that represent a logo
14     dv::EventStore events = dv::data::generate::dvLogoAsEvents(0, resolution);
15
16     // Initialize event filter chain, it contains no filters
17     dv::EventFilterChain filter;
18
19     // Now let's add filters
20     // First, add a region filter with hardcoded coordinates
21     filter.addFilter(std::make_shared<dv::EventRegionFilter<>>(cv::Rect(50, 50, 100, 100)));
22
23     // Second, add a positive polarity filter
24     filter.addFilter(std::make_shared<dv::EventPolarityFilter<>>(true));
25
26     // Third, add a background activity noise filter
27     filter.addFilter(std::make_shared<dv::noise::BackgroundActivityNoiseFilter<>>(resolution));
28
29     // Pass events to the filter
30     filter.accept(events);
31
32     // Call generate events to apply the filter chain, it will apply all three filters
33     const dv::EventStore filtered = filter.generateEvents();
34
35     // Use a visualizer instance to preview the events
36     dv::visualization::EventVisualizer visualizer(resolution);
37
38     // Generate preview images of data input and output
39     const cv::Mat input = visualizer.generateImage(events);

```

(continues on next page)

(continued from previous page)

```

40     const cv::Mat output = visualizer.generateImage(filtered);
41
42     // Concatenate the images into a single image for preview
43     cv::Mat preview;
44     cv::hconcat(input, output, preview);
45
46     // Display the input and output images
47     cv::namedWindow("preview", cv::WINDOW_NORMAL);
48     cv::imshow("preview", preview);
49     cv::waitKey();
50
51     return 0;
52 }

```

```

1  import dv_processing as dv
2  import cv2 as cv
3  from datetime import timedelta
4
5  resolution = (200, 200)
6
7  # Initializing input events with events that represent a logo
8  events = dv.data.generate.dvLogoAsEvents(0, resolution)
9
10 # Initialize event filter chain, it contains no filters
11 filter = dv.EventFilterChain()
12
13 # Now let's add filters
14 # First, add a region filter with hardcoded coordinates
15 filter.addFilter(dv.EventRegionFilter((50, 50, 100, 100)))
16
17 # Second, add a positive polarity filter
18 filter.addFilter(dv.EventPolarityFilter(True))
19
20 # Third, add a background activity noise filter
21 filter.addFilter(dv.noise.BackgroundActivityNoiseFilter(resolution))
22
23 # Pass events to the filter
24 filter.accept(events)
25
26 # Call generate events to apply the filter
27 filtered = filter.generateEvents()
28
29 # Use a visualizer instance to preview the events
30 visualizer = dv.visualization.EventVisualizer(resolution)
31
32 # Generate preview images of data input and output
33 input = visualizer.generateImage(events)
34 output = visualizer.generateImage(filtered)
35
36 # Concatenate the images into a single image for preview
37 preview = cv.hconcat([input, output])
38

```

(continues on next page)

(continued from previous page)

```

39 # Display the input and output images
40 cv.namedWindow("preview", cv.WINDOW_NORMAL)
41 cv.imshow("preview", preview)
42 cv.waitKey()

```

The sample code outputs such images:



Fig. 14: Output of the sample use of multiple filters in a filter chain. Left is a preview of input events and right is a preview of filtered events.

3.4.4 Filtering performance

The provided event filters performance is measured using benchmarks, the benchmarks for filters can be found under directory `benchmarks/noise` in the project repository. These are sample benchmarking results, performance of filters is measured in throughput of mega-events per second. These measurements were capture on an Intel i7-1165G7 8-Core processor.

Filter	Event count per iteration	Throughput, MegaEvents / second
FastDecayNoiseFilter	1000	86.9
FastDecayNoiseFilter	4096	90.5
FastDecayNoiseFilter	32768	89.2
FastDecayNoiseFilter	262144	89.0
FastDecayNoiseFilter	1000000	88.3
BackgroundActivityNoiseFilter	1000	282.9
BackgroundActivityNoiseFilter	4096	294.7
BackgroundActivityNoiseFilter	32768	226.0
BackgroundActivityNoiseFilter	262144	198.6
BackgroundActivityNoiseFilter	1000000	203.7
LowPassFilter	1000	432.1
LowPassFilter	4096	449.7
LowPassFilter	32768	374.3
LowPassFilter	262144	281.0
LowPassFilter	1000000	307.6
HighPassFilter	1000	287.6
HighPassFilter	4096	296.2
HighPassFilter	32768	244.6
HighPassFilter	262144	205.4

continues on next page

Table 1 – continued from previous page

Filter	Event count per iteration	Throughput, MegaEvents / second
HighPassFilter	1000000	185.4
BandPassFilter	1000	355.8
BandPassFilter	4096	342.4
BandPassFilter	32768	312.0
BandPassFilter	262144	233.7
BandPassFilter	1000000	265.4
BandCutFilter	1000	276.6
BandCutFilter	4096	334.7
BandCutFilter	32768	264.6
BandCutFilter	262144	156.2
BandCutFilter	1000000	133.2
PolarityFilter	1000	539.7
PolarityFilter	4096	302.7
PolarityFilter	32768	184.3
PolarityFilter	262144	177.6
PolarityFilter	1000000	172.9
RegionFilter	1000	526.3
RegionFilter	4096	481.5
RegionFilter	32768	185.7
RegionFilter	262144	175.1
RegionFilter	1000000	168.5
MaskFilter	1000	545.7
MaskFilter	4096	304.2
MaskFilter	32768	143.5
MaskFilter	262144	131.6
MaskFilter	1000000	132.1
ThreeFiltersNoChain	1000	195.5
ThreeFiltersNoChain	4096	112.2
ThreeFiltersNoChain	32768	64.0
ThreeFiltersNoChain	262144	48.6
ThreeFiltersNoChain	1000000	57.6
ThreeFiltersWithChain	1000	173.0
ThreeFiltersWithChain	4096	91.2
ThreeFiltersWithChain	32768	74.3
ThreeFiltersWithChain	262144	73.1
ThreeFiltersWithChain	1000000	62.5

3.5 Command-line utilities

The dv-processing library provides certain command-line utilities that are useful for troubleshooting basic issues or performing some basic tasks.

3.5.1 dv-filestat

The dv-filestat utility is useful for inspection of AEDAT4 files. It provides information on the content of a given file. Usage:

```
$ dv-filestat path/to/file.aedat4
```

Sample output from this utility:

```
$ dv-filestat ~/dvSave-2022_06_29_13_40_44.aedat4
File path (canonical): "/home/rokas/dvSave-2022_06_29_13_40_44.aedat4"
File size (OS): 25429285
File size (Parser): 25429285
Compression: LZ4
Timestamp lowest: 1656502844057775
Timestamp highest: 1656502852007717
Timestamp difference: 7949942
Timestamp shift: 1656502844057775
Stream 0: events - EVTS
Stream 2: imu - IMUS
Stream 3: triggers - TRIG
DataTable file position: 25377171
DataTable file size: 52114
DataTable elements: 1590
```

Here, the provided fields are:

- File path - absolute file path in the filesystem;
- File size - actual file size in bytes;
- Compression - compression type;
- Timestamp - timing information in Unix microsecond format;
- Stream - available streams (e.g. "Stream 0: events - EVTS"), where "0" - stream id, "events" - stream name, "EVTS" - stream type identifier;
- DataTable - internal data layout information.

It is possible to pass a verbose flag ("-v" or "-verbose") to obtain more information on the recorded data. Sample verbose output:

```
$ dv-filestat -v ~/dvSave-2022_06_29_13_40_44.aedat4
File path (canonical): "/home/rokas/dvSave-2022_06_29_13_40_44.aedat4"
File size (OS): 25429285
File size (Parser): 25429285
Compression: LZ4
Timestamp lowest: 1656502844057775
Timestamp highest: 1656502852007717
Timestamp difference: 7949942
Timestamp shift: 1656502844057775
Stream 0: events - EVTS
XML content:
0/
compression = LZ4
originalModuleName = capture
originalOutputName = events
typeDescription = Array of events (polarity ON/OFF).
typeIdentifier = EVTS
0/info/
sizeX = 640
sizeY = 480
source = DVXplorer_DXA00093
tsOffset = 1656502833725006
```

(continues on next page)

(continued from previous page)

```

Stream 2: imu - IMUS
XML content:
2/
compression = LZ4
originalModuleName = capture
originalOutputName = imu
typeDescription = Inertial Measurement Unit data samples.
typeIdentifier = IMUS
  2/info/
    source = DVXplorer_DXA00093
    tsOffset = 1656502833725006

Stream 3: triggers - TRIG
XML content:
3/
compression = LZ4
originalModuleName = capture
originalOutputName = triggers
typeDescription = External triggers and special signals.
typeIdentifier = TRIG
  3/info/
    source = DVXplorer_DXA00093
    tsOffset = 1656502833725006

DataTable file position: 25377171
DataTable file size: 52114
DataTable elements: 1590
Packet at 2342: StreamID 0 - Size 31375 - NumElements 4911 - TimestampStart↵
↵1656502844057775 - TimestampEnd 1656502844067773
Packet at 33725: StreamID 2 - Size 401 - NumElements 8 - TimestampStart↵
↵1656502844057878 - TimestampEnd 1656502844066774
Packet at 34134: StreamID 0 - Size 30591 - NumElements 4815 - TimestampStart↵
↵1656502844067783 - TimestampEnd 1656502844077744
Packet at 64733: StreamID 2 - Size 374 - NumElements 8 - TimestampStart↵
↵1656502844068045 - TimestampEnd 1656502844076941
Packet at 65115: StreamID 0 - Size 32736 - NumElements 5131 - TimestampStart↵
↵1656502844077777 - TimestampEnd 1656502844087761
Packet at 97859: StreamID 2 - Size 378 - NumElements 8 - TimestampStart↵
↵1656502844078212 - TimestampEnd 1656502844087107
Packet at 98245: StreamID 0 - Size 31442 - NumElements 4933 - TimestampStart↵
↵1656502844087779 - TimestampEnd 1656502844097756
Packet at 129695: StreamID 2 - Size 378 - NumElements 8 - TimestampStart↵
↵1656502844088378 - TimestampEnd 1656502844097274

```

The verbose output now prints each streams metadata, e.g.:

```

Stream 0: events - EVTS
XML content:
0/
compression = LZ4
originalModuleName = capture

```

(continues on next page)

(continued from previous page)

```
originalOutputName = events
typeDescription = Array of events (polarity ON/OFF) .
typeIdentifier = EVTS
  0/info/
  sizeX = 640
  sizeY = 480
  source = DVXplorer_DXA00093
  tsOffset = 1656502833725006
```

The XML metadata provide information on the compression type, some information on the type, as well as source camera name and the resolution of the data.

Additionally, packet information is printed, e.g.:

```
Packet at 2342: StreamID 0 - Size 31375 - NumElements 4911 - TimestampStart↵
↵1656502844057775 - TimestampEnd 1656502844067773
```

The information here represents:

- “Packet at 2342” - the 2342 is the byte start index of this packet in the file;
- “StreamID 0” - id of the stream, which matches stream ids from output above;
- “Size 31375” - size of the content of this packet in bytes;
- “NumElements 4911” - number of element in packet, since it’s an event packet, it contains 4911 events;
- “TimestampStart 1656502844057775 - TimestampEnd 1656502844067773” - this is the timing information of this packet, it contains data from time 1656502844057775 to 1656502844067773. These are Unix microsecond timestamps.

3.5.2 dv-list-devices

The `dv-list-devices` is a basic utility that allows inspection of connected iniVation cameras on the system. Sample output from this command:

```
$ dv-list-devices
Device discovery: found 1 devices.
Detected device [DVXPLOER DXA00093]
```

Running the command without any additional flags will just print the list of available cameras with their serial numbers. By adding a verbose flag (“-v” or “--verbose”), it will print additional information about the available cameras, a sample output:

```
$ dv-list-devices --verbose
Device discovery: found 1 devices.
Detected device [DVXPLOER DXA00093]
- DVXPLOER
  - USB busNumber:devAddress: 10:21
  - USB Device Type: FX3_RED
  - Serial Number: DXA00093
  - Firmware Version: 9
```

3.5.3 dv-imu-bias-estimation

The `dv-imu-bias-estimation` tool is used to estimate intrinsic measurement biases that are measured by the IMU device when there is no motion to be measured. The biases are internal offsets in the measuring device that come from imperfections during manufacturing process of the IMU device.

Usage:

```
Usage: dv-imu-bias-estimation [OPTIONS]

Options:
  -h, --help                Print this help message and exit
  -c, --camera-name TEXT    Provide camera name to open. Application will open
                             ↪first discovered camera if a name is not provided.
  -t, --variance-threshold FLOAT [0.1]
                             Maximum variance that can be measured on IMU data. This
                             ↪value is used to determine motion in the data.
  -d, --duration FLOAT [1]  Time duration for collecting sample data in seconds.
  -a, --calibration-file TEXT:FILE
                             Path to the calibration file to store the biases values.
```

To estimate the biases, first place the camera steady on a level surface and run the utility:

```
$ dv-imu-bias-estimation
Opened camera [DVXplorer_DXA00093]
Keep the camera steady on a level surface and press ENTER to start data collection ...

Collected 794 samples over 1.007546 seconds
Bias estimation successful!
Accelerometer biases [x, y, z] in m/s^2:
    [-0.4947727, -0.7454767, 0.12935007]
Gyroscope biases [x, y, z] in rad/s:
    [0.0028670905, 0.004207727, 0.0023824223]
```

When running without any parameters, the utility will open the first discovered camera and will perform bias estimation for it. It will prompt the user to place the camera on a level surface and press enter key to start collection. When the key is pressed, it will collect 1 second of IMU measurements and estimate the biases which are printed into the screen. These values can be subtracted from raw measurements of IMU to compensate for the intrinsic offsets the IMU is measuring.

The utility also provides additional parameters, below is a detailed explanation of these value:

- `-c, --camera-name` - provide a specific camera name to open. By default, the application will open first discovered device on the system, by providing a specific name, it will open the specified device.
- `-t, --variance-threshold` - to ensure that the camera is stationary while collecting data, the utility is measuring variance on the measured data. It will throw an exception if variance of at least one measurement is exceeding the given threshold. If this threshold is exceeded and the camera was not moved, this value can be increased to bypass this check.
- `-d, --duration` - duration of data collection in seconds. Usually one second is sufficient for a general case. The duration value can be tuned if estimated biases are inconsistent.
- `-a, --calibration-file` - the estimated bias values are only printed to the terminal output. It is also possible to save them in a calibration file for persistence. The utility will open the given file (the file needs to exist and contain a calibration) and update the IMU parameter values. If an IMU calibration for the camera already exists, it will overwrite previous settings, if IMU calibration for this camera does not exist, the calibration will be added.



Fig. 15: Camera placed on a level table surface for imu bias estimation.

3.5.4 dv-tcpstat

The `dv-tcpstat` utility is useful for inspection of remote TCP server streams. It provides information on the stream the remote server provides. It is compatible with TCP remote streaming instances created using `dv::io::NetworkWriter` class and with DV's [Output net tcp server module](#)⁸.

Usage:

```
$ dv-tcpstat
Connect to a TCP streaming server and print information about it
Usage: dv-tcpstat [OPTIONS]

Options:
  -h, --help                Print this help message and exit
  -v, --verbose              Print full packet table
  -r, --try-reading          Try reading and printing packet information
  -i, --ip TEXT:IPV4 REQUIRED IP address of the server to connect to
  -p, --port UINT:INT in [0 - 65535] REQUIRED
                             Server port number
```

As a sample demonstration of usage, let's take a DV project that outputs event stream into the output TCP module.

The module is configured to listen on IP address 127.0.0.1 and port 46581, we can inspect this server using the `dv-tcpstat` utility from the same computer. When running the utility with given IP, port, and verbose output flag, it prints the information about the available data:

```
$ dv-tcpstat -i 127.0.0.1 -p 46581 -v
Attempting to connect to [127.0.0.1:46581]...
Connected to [127.0.0.1:46581]!

Stream info on stream ID 0:
  Stream name: "tcpserver"
  Stream type identifier: "EVTS"
  Stream details:
    /0/
    originalOutputName = tcpserver
    typeDescription = Array of events (polarity ON/OFF).
    typeIdentifier = EVTS
    /0/info/
    sizeX = 640
    sizeY = 480
    source = DVXplorer_DXA00093
```

The executable will print this information and disconnect immediately. It is possible to try and read some data from the remote server by passing the `-r, --try-reading` flag, this is the expected with this option enabled:

```
$ dv-tcpstat -i 127.0.0.1 -p 46581 -r
Attempting to connect to [127.0.0.1:46581]...
Connected to [127.0.0.1:46581]!

Stream info on stream ID 0:
  Stream name: "tcpserver"
  Stream type identifier: "EVTS"
  Stream details:
```

(continues on next page)

⁸ <https://docs.inivation.com/software/dv/modules/built-in-modules/output-network.html#output-tcp-module>

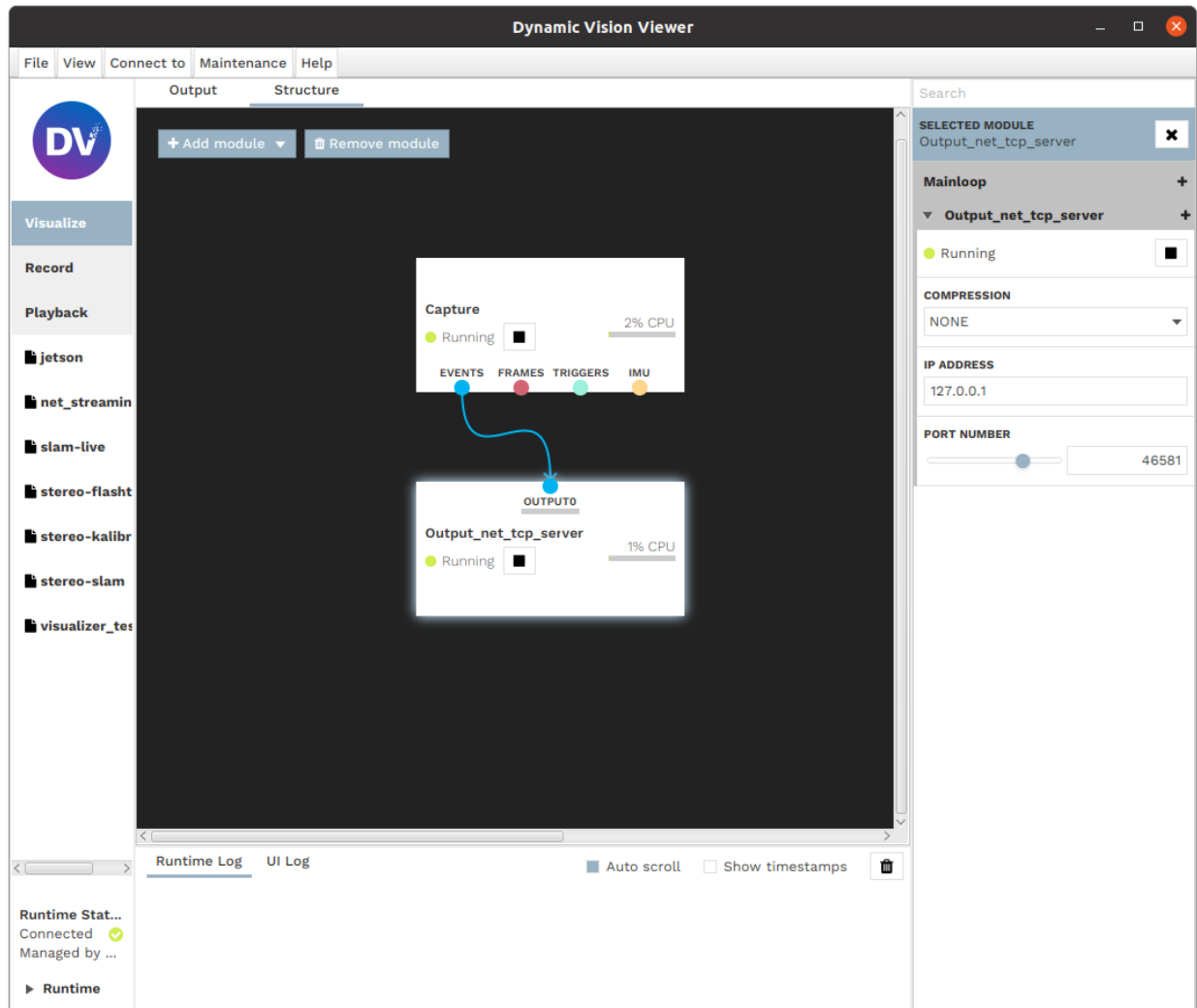


Fig. 16: A project in DV running a live camera connected to a TCP output server module.

(continued from previous page)

```

/0/
originalOutputName = tcpserver
typeDescription = Array of events (polarity ON/OFF).
typeIdentifier = EVTS
    /0/info/
    sizeX = 640
    sizeY = 480
    source = DVXplorer_DXA00093
[2022-11-25 09:31:33] Received: EventStore containing 3103 events within 29416µs
↳duration; time range within [1669365093787516; 1669365093816932]
[2022-11-25 09:31:33] Received: EventStore containing 3100 events within 29007µs
↳duration; time range within [1669365093816932; 1669365093845939]
[2022-11-25 09:31:33] Received: EventStore containing 3105 events within 29034µs
↳duration; time range within [1669365093845939; 1669365093874973]
[2022-11-25 09:31:33] Received: EventStore containing 3107 events within 28213µs
↳duration; time range within [1669365093874973; 1669365093903186]
[2022-11-25 09:31:33] Received: EventStore containing 3090 events within 29441µs
↳duration; time range within [1669365093903186; 1669365093932627]
[2022-11-25 09:31:33] Received: EventStore containing 3097 events within 29614µs
↳duration; time range within [1669365093932627; 1669365093962241]
[2022-11-25 09:31:33] Received: EventStore containing 3109 events within 27834µs
↳duration; time range within [1669365093962241; 1669365093990075]
[2022-11-25 09:31:33] Received: EventStore containing 3107 events within 27601µs
↳duration; time range within [1669365093990075; 1669365094017676]

```

The executable will print packet information until a interrupt signal is sent by the user.

3.6 Reading camera data

The dv-processing library provides a convenient way of reading camera data from live connected cameras and persistent files.

3.6.1 Camera name

Camera name is used to identify a unique camera produced by iniVation. Camera name consists of a camera model and a serial number, concatenated by an underscore (“_”) character. The library refers to camera name in multiple methods, this value can be consistently used across the library.

Some examples of camera names:

- DVXplorer: DVXplorer_DXA00093, DVXplorer_DXM00123
- DAVIS: DAVIS346_00000499

Note

This definition is valid for USB cameras, the camera name is also reported in network streaming sources. In that case, camera name can be manually set by the developer, so naming convention for the models might not be entirely followed.

3.6.2 From a camera

The easiest approach to access data from a live camera is to use the `dv::io::camera::open()` functions. This section provides in-depth explanations on their usage and code samples.

Discover connected cameras

The command-line utility `dv-list-devices` is available in the packages of `dv-processing` and can show information on the connected cameras. Sample output for the utility:

```
$ dv-list-devices
Device discovery: found 2 devices.
Detected device [DAVIS 00000499]
Detected device [DVXPLOER DXA00093]
```

Note

This utility reports type and serial number, not the full camera name described above. The `getCameraName()` function provides the full name once a device has been opened.

Device discovery is also possible with the use of library methods. Following is a sample on how to detect connected devices using discovery method:

C++

Python

```
1 #include <dv-processing/io/camera/discovery.hpp>
2
3 #include <iostream>
4
5 int main() {
6     // Call the discovery method
7     const auto cameras = dv::io::camera::discover();
8
9     std::cout << "Device discovery: found " << cameras.size() << " devices." << \
↪std::endl;
10
11     // Loop through detected camera names and print them
12     for (const auto &camera : cameras) {
13         std::cout << "Detected device [" << camera.cameraModel << " " << camera.
↪serialNumber << "]" << std::endl;
14     }
15
16     return 0;
17 }
```

```
1 import dv_processing as dv
2
3 cameras = dv.io.camera.discover()
4
5 print(f"Device discovery: found {len(cameras)} devices.")
6 for camera_name in cameras:
7     print(f"Detected device [{camera_name}]")
```

Opening a camera

The `dv::io::camera::DVS128`, `dv::io::camera::DAVIS`, `dv::io::camera::DVXplorer` and `dv::io::camera::DVXplorerM` classes follow the RAII⁹ pattern for resource management. Creating an instance of the class will open the camera connected on USB and starts reading data immediately; the camera is shut-down when the object instance is destroyed. These classes allow full interaction with the cameras, exposing all available functionality.

It is also possible to open cameras in a more generic way, using the `dv::io::camera::open()` functions. For cameras with multi-camera synchronization support, you can also use the `dv::io::camera::openSync()` functions. This allows access to a set of features common to all cameras. For access to all features, you will have to up-cast the generic camera to the class representing its full type.

C++

Python

```
// Open the first camera found
auto capture = dv::io::camera::open();
```

```
import dv_processing as dv

# Open the first camera found
capture = dv.io.camera.open()
```

It is also possible to open a specific camera on the system by providing the serial number:

C++

Python

```
// Open the specified camera
auto capture = dv::io::camera::open("DXA00093");
```

```
import dv_processing as dv

# Open the specified camera
capture = dv.io.camera.open("DXA00093")
```

To access all features, it's easiest to open the camera using its full class:

C++

Python

```
// Open any DAVIS camera (camera name not specified)
auto capture = dv::io::camera::DAVIS();
```

```
import dv_processing as dv

# Open any DAVIS camera (camera name not specified)
capture = dv.io.camera.DAVIS()
```

⁹ https://en.wikipedia.org/wiki/Resource_acquisition_is_initialization

Checking camera capabilities

Since some cameras provide different data types and capabilities, the base class `dv::io::camera::CameraInputBase` provides methods to test what data the camera can provide:

C++

Python

```

1  #include <dv-processing/io/camera/discovery.hpp>
2
3  int main() {
4      // Open any camera
5      auto capture = dv::io::camera::open();
6
7      // Print the camera name
8      std::cout << "Opened [" << capture->getCameraName() << "] camera, it provides:" <
    -> std::endl;
9
10     // Check whether event stream is available
11     if (capture->isEventStreamAvailable()) {
12         // Get the event stream resolution, the output is a std::optional, so the
    ->value() method is
13         // used to get the actual resolution value
14         const cv::Size resolution = capture->getEventResolution().value();
15
16         // Print the event stream capability with resolution value
17         std::cout << "* Events at " << resolution << " resolution" << std::endl;
18     }
19
20     // Check whether frame stream is available
21     if (capture->isFrameStreamAvailable()) {
22         // Get the frame stream resolution
23         const cv::Size resolution = capture->getFrameResolution().value();
24
25         // Print the frame stream capability with resolution value
26         std::cout << "* Frames at " << resolution << " resolution" << std::endl;
27     }
28
29     // Check whether the IMU stream is available
30     if (capture->isImuStreamAvailable()) {
31         // Print the imu data stream capability
32         std::cout << "* IMU measurements" << std::endl;
33     }
34
35     // Check whether the trigger stream is available
36     if (capture->isTriggerStreamAvailable()) {
37         // Print the trigger stream capability
38         std::cout << "* Triggers" << std::endl;
39     }
40
41     return 0;
42 }
```

```
1 import dv_processing as dv
2
3 # Open any camera
4 capture = dv.io.camera.open()
5
6 # Print the camera name
7 print(f"Opened [{capture.getCameraName()}] camera, it provides:")
8
9 # Check whether event stream is available
10 if capture.isEventStreamAvailable():
11     # Get the event stream resolution
12     resolution = capture.getEventResolution()
13
14     # Print the event stream capability with resolution value
15     print(f"* Events at ({resolution[0]}x{resolution[1]}) resolution")
16
17 # Check whether frame stream is available
18 if capture.isFrameStreamAvailable():
19     # Get the frame stream resolution
20     resolution = capture.getFrameResolution()
21
22     # Print the frame stream capability with resolution value
23     print(f"* Frames at ({resolution[0]}x{resolution[1]}) resolution")
24
25 # Check whether the IMU stream is available
26 if capture.isImuStreamAvailable():
27     # Print the imu data stream capability
28     print("* IMU measurements")
29
30 # Check whether the trigger stream is available
31 if capture.isTriggerStreamAvailable():
32     # Print the trigger stream capability
33     print("* Triggers")
```

Configuring camera options

Some advanced properties of our cameras can be configured by a number of functions. Some are listed here for reference, please check the detailed API documentation for more details.

DVXplorer camera advanced control options

Sensitivity and eFPS

C++

Python

```
1 #include <dv-processing/io/camera/dvxplorer.hpp>
2
3 int main() {
4     // Open a DVXplorer camera
5     dv::io::camera::DVXplorer capture{};
6 }
```

(continues on next page)

(continued from previous page)

```

7 // Configure event sensitivity. Range 0-17, default 9.
8 capture.setContrastThresholdOn(9);
9 capture.setContrastThresholdOff(9);
10
11 // Configure event-frame readouts per second (here variable 5000 FPS, the default_
↪value)
12 // See detailed API documentation for other available values
13 capture.setReadoutFPS(dv::io::camera::DVXplorer::ReadoutFPS::VARIABLE_5000);
14
15 // Enable global hold setting (already the default)
16 capture.setGlobalHold(true);
17 // Disable global reset setting (already the default)
18 capture.setGlobalReset(false);
19
20 return 0;
21 }

```

```

1 import dv_processing as dv
2
3 # Open a DVXplorer camera
4 capture = dv.io.camera.DVXplorer()
5
6 # Configure event sensitivity. Range 0-17, default 9.
7 capture.setContrastThresholdOn(9)
8 capture.setContrastThresholdOff(9)
9
10 # Configure event-frame readouts per second (here variable 5000 FPS, the default_
↪value)
11 # See detailed API documentation for other available values
12 capture.setReadoutFPS(dv.io.camera.DVXplorer.ReadoutFPS.VARIABLE_5000)
13
14 # Enable global hold setting (already the default)
15 capture.setGlobalHold(True)
16 # Disable global reset setting (already the default)
17 capture.setGlobalReset(False)

```

Read more about DVXplorer biases and details about the ReadoutFPS implementation in our [documentation page](#)¹⁰.

Note

On DVXplorer, setting global hold to false can help for certain applications containing repeating patterns observation, such as flickering LEDs.

Crop events to a region of interest

C++

Python

¹⁰ <https://docs.inivation.com/hardware/hardware-advanced-usage/biasing.html#dvxplorer-biases>

```
1 #include <dv-processing/io/camera/dvxplorer.hpp>
2
3 int main() {
4     // Open a DVXplorer camera
5     dv::io::camera::DVXplorer capture{};
6
7     // Set ROI area, automatically takes effect
8     capture.setCropArea({0, 0, 200, 200});
9
10    return 0;
11 }
```

```
1 import dv_processing as dv
2
3 # Open a DVXplorer camera
4 capture = dv.io.camera.DVXplorer()
5
6 # Set ROI area, automatically takes effect
7 capture.setCropArea((0, 0, 200, 200))
```

DAVIS camera advanced control options

General options

C++

Python

```
1 #include <dv-processing/io/camera/davis.hpp>
2
3 int main() {
4     // Open a Davis camera
5     dv::io::camera::DAVIS capture{};
6
7     // Setting camera readout to events and frames (default).
8     capture.setEventsRunning(true);
9     capture.setFramesRunning(true);
10
11     // Configure frame output mode to color (default), only on COLOR cameras. Other_
12     ↪mode available: GRAYSCALE
13     capture.setColorMode(dv::io::camera::parser::DAVIS::ColorMode::DEFAULT);
14
15     return 0;
16 }
```

```
1 import dv_processing as dv
2
3 # Open a Davis camera
4 capture = dv.io.camera.DAVIS()
5
6 # Setting camera readout to events and frames (default).
7 capture.setEventsRunning(True)
```

(continues on next page)

(continued from previous page)

```

8 capture.setFramesRunning(True)
9
10 # Configure frame output mode to color (default), only on COLOR cameras. Other mode_
   ↳ available: GRAYSCALE
11 capture.setColorMode(dv.io.camera.DAVIS.ColorMode.DEFAULT)

```

Frame options

C++

Python

```

1  #include <dv-processing/io/camera/davis.hpp>
2
3  #include <chrono>
4
5  int main() {
6      using namespace std::chrono_literals;
7
8      // Open a Davis camera
9      dv::io::camera::DAVIS capture{};
10
11     // Enable frame auto-exposure (default behavior)
12     capture.setAutoExposure(true);
13
14     // Disable auto-exposure, set frame exposure (here 10ms)
15     capture.setAutoExposure(false);
16     capture.setExposureDuration(10ms);
17
18     // Read current frame exposure duration value
19     std::chrono::microseconds duration = capture.getExposureDuration();
20
21     // Set frame interval duration (here 33ms for ~30FPS)
22     capture.setFrameInterval(33ms);
23
24     // Read current frame interval duration value
25     std::chrono::microseconds interval = capture.getFrameInterval();
26
27     return 0;
28 }

```

```

1  from datetime import timedelta
2
3  import dv_processing as dv
4
5  # Open a Davis camera
6  capture = dv.io.camera.DAVIS()
7
8  # Enable frame auto-exposure (default behavior)
9  capture.setAutoExposure(True)
10 # Disable auto-exposure, set frame exposure (here 10ms)
11 capture.setAutoExposure(False)

```

(continues on next page)

(continued from previous page)

```

12 capture.setExposureDuration(timedelta(milliseconds=10))
13 # Read current frame exposure duration value
14 duration = capture.getExposureDuration()
15 # Set frame interval duration (here 33ms for ~30FPS)
16 capture.setFrameInterval(timedelta(milliseconds=33))
17 # Read current frame interval duration value
18 interval = capture.getFrameInterval()

```

Event options (biases)

Warning

Before using biases, make sure that you absolutely need to change them and that you understand them by reading about biases on our [documentation page](#)¹¹.

C++

Python

```

1  #include <dv-processing/io/camera/davis.hpp>
2
3  int main() {
4      // Open a Davis346 camera
5      dv::io::camera::DAVIS capture{};
6
7      /// Access biases raw value
8      // Photoreceptor bias
9      auto biasPhotoreceptor = capture.
→getDavis346BiasCoarseFine(dv::io::camera::DAVIS::Davis346BiasCF::Photoreceptor);
10     // Source follower bias
11     auto biasPhotoreceptorSourceFollower
12         = capture.
→getDavis346BiasCoarseFine(dv::io::camera::DAVIS::Davis346BiasCF::PhotoreceptorSourceFollower);
13     // Differential bias
14     auto biasDiff = capture.
→getDavis346BiasCoarseFine(dv::io::camera::DAVIS::Davis346BiasCF::Diff);
15     // On threshold bias
16     auto biasOn = capture.
→getDavis346BiasCoarseFine(dv::io::camera::DAVIS::Davis346BiasCF::On);
17     // Off threshold bias
18     auto biasOff = capture.
→getDavis346BiasCoarseFine(dv::io::camera::DAVIS::Davis346BiasCF::Off);
19     // Refractory period bias
20     auto biasRefractory = capture.
→getDavis346BiasCoarseFine(dv::io::camera::DAVIS::Davis346BiasCF::Refractory);
21
22     // Add 1 on the log-scale coarse value, i.e. multiply bias value by 8_
→(approximately)

```

(continues on next page)

¹¹ <https://docs.inivation.com/hardware/hardware-advanced-usage/biasing.html>

(continued from previous page)

```

23     biasPhotoreceptor.first           += 1;
24     biasPhotoreceptorSourceFollower.first += 1;
25     biasDiff.first                   += 1;
26     biasOn.first                     += 1;
27     biasOff.first                    += 1;
28     biasRefractory.first              += 1;
29
30     /// Set biases raw value
31     // Setting photoreceptor bias
32     capture.setDavis346BiasCoarseFine(
33         dv::io::camera::DAVIS::Davis346BiasCF::Photoreceptor, biasPhotoreceptor.first,
34         ↪ biasPhotoreceptor.second);
35     // Setting source follower bias
36     capture.
37     ↪ setDavis346BiasCoarseFine(dv::io::camera::DAVIS::Davis346BiasCF::PhotoreceptorSourceFollower,
38     ↪
39     ↪ biasPhotoreceptorSourceFollower.first, biasPhotoreceptorSourceFollower.
40     ↪ second);
41     // Setting differential bias
42     capture.setDavis346BiasCoarseFine(dv::io::camera::DAVIS::Davis346BiasCF::Diff, ↪
43     ↪ biasDiff.first, biasDiff.second);
44     // Setting on threshold bias
45     capture.setDavis346BiasCoarseFine(dv::io::camera::DAVIS::Davis346BiasCF::On, ↪
46     ↪ biasOn.first, biasOn.second);
47     // Setting off threshold bias
48     capture.setDavis346BiasCoarseFine(dv::io::camera::DAVIS::Davis346BiasCF::Off, ↪
49     ↪ biasOff.first, biasOff.second);
50     // Setting refractory period bias
51     capture.setDavis346BiasCoarseFine(
52         dv::io::camera::DAVIS::Davis346BiasCF::Refractory, biasRefractory.first, ↪
53     ↪ biasRefractory.second);
54
55     return 0;
56 }

```

```

1  import dv_processing as dv
2
3  # Open a Davis camera
4  capture = dv.io.camera.DAVIS()
5
6  # - Access biases raw value
7  # Photoreceptor bias
8  defaultPrBpInt = capture.getDavis346BiasCurrent(dv.io.camera.DAVIS.Davis346BiasCF.
9  ↪ Photoreceptor)
10 # Source follower bias
11 defaultPrSfBpInt = capture.getDavis346BiasCurrent(dv.io.camera.DAVIS.Davis346BiasCF.
12 ↪ PhotoreceptorSourceFollower)
13 # Differential bias
14 defaultDiffBnInt = capture.getDavis346BiasCurrent(dv.io.camera.DAVIS.Davis346BiasCF.
15 ↪ Diff)
16 # On threshold bias
17 defaultOnBnInt = capture.getDavis346BiasCurrent(dv.io.camera.DAVIS.Davis346BiasCF.On)

```

(continues on next page)

(continued from previous page)

```

15 # Off threshold bias
16 defaultOffBnInt = capture.getDavis346BiasCurrent(dv.io.camera.DAVIS.Davis346BiasCF.
    ↪Off)
17 # Refractory period bias
18 defaultRefrBpInt = capture.getDavis346BiasCurrent(dv.io.camera.DAVIS.Davis346BiasCF.
    ↪Refractory)
19
20 # Change biases values by 1 milliAmpere (1 mio picoAmpere).
21 defaultPrBpInt += 1000000
22 defaultPrSfBpInt += 1000000
23 defaultDiffBnInt += 1000000
24 defaultOnBnInt += 1000000
25 defaultOffBnInt += 1000000
26 defaultRefrBpInt += 1000000
27
28 # Set biases raw value
29 # Setting photoreceptor bias
30 capture.setDavis346BiasCurrent(dv.io.camera.DAVIS.Davis346BiasCF.Photoreceptor, ↪
    ↪defaultPrBpInt)
31 # Setting source follower bias
32 capture.setDavis346BiasCurrent(dv.io.camera.DAVIS.Davis346BiasCF.
    ↪PhotoreceptorSourceFollower, defaultPrSfBpInt)
33 # Setting differential bias
34 capture.setDavis346BiasCurrent(dv.io.camera.DAVIS.Davis346BiasCF.Diff, ↪
    ↪defaultDiffBnInt)
35 # Setting on threshold bias
36 capture.setDavis346BiasCurrent(dv.io.camera.DAVIS.Davis346BiasCF.On, defaultOnBnInt)
37 # Setting off threshold bias
38 capture.setDavis346BiasCurrent(dv.io.camera.DAVIS.Davis346BiasCF.Off, defaultOffBnInt)
39 # Setting refractory period bias
40 capture.setDavis346BiasCurrent(dv.io.camera.DAVIS.Davis346BiasCF.Refractory, ↪
    ↪defaultRefrBpInt)

```

Crop events or frames to a region of interest

C++

Python

```

1 #include <dv-processing/io/camera/davis.hpp>
2
3 int main() {
4     // Open a Davis camera
5     dv::io::camera::DAVIS capture{};
6
7     // Set ROI for events
8     capture.setCropAreaEvents({0, 0, 200, 200});
9
10    // Set ROI for frames
11    capture.setCropAreaFrames({0, 0, 300, 150});
12
13    return 0;

```

(continues on next page)

(continued from previous page)

```

14 }

1 import dv_processing as dv
2
3 # Open a Davis camera
4 capture = dv.io.camera.DAVIS()
5
6 # Set ROI for events
7 capture.setCropAreaEvents((0, 0, 200, 200))
8
9 # Set ROI for frames
10 capture.setCropAreaFrames((0, 0, 300, 150))

```

Read events from a live camera

Incoming data from a camera can be read sequentially using the `dv::io::CameraInputBase::getNextEventBatch()`. Following is a minimal sample on how to read events sequentially from a camera:

C++

Python

```

1 #include <dv-processing/io/camera/discovery.hpp>
2
3 #include <chrono>
4
5 int main() {
6     using namespace std::chrono_literals;
7
8     // Open any camera
9     auto capture = dv::io::camera::open();
10
11     // Run the loop while camera is still connected
12     while (capture->isRunning()) {
13         // Read batch of events, check whether received data is correct.
14         // The method does not wait for data arrive, it returns immediately with
15         // the latest available data or if no data is available, returns a
16         ↪ `std::nullopt`.
17         if (const auto events = capture->getNextEventBatch(); events.has_value()) {
18             // Print received packet information
19             std::cout << *events << std::endl;
20         }
21         else {
22             // No data has arrived yet, short sleep to reduce CPU load.
23             std::this_thread::sleep_for(1ms);
24         }
25     }
26
27     return 0;
28 }

```

```

1 import time

```

(continues on next page)

(continued from previous page)

```

2
3 import dv_processing as dv
4
5 # Open any camera
6 capture = dv.io.camera.open()
7
8 # Run the loop while camera is still connected
9 while capture.isRunning():
10     # Read batch of events
11     events = capture.getNextEventBatch()
12
13     # The method does not wait for data arrive, it returns immediately with
14     # latest available data or if no data is available, returns a `None`
15     if events is not None:
16         # Print received packet time range
17         print(f"Received events within time range [{events.getLowestTime()}; {events.
18 ↪getHighestTime()}]")
19     else:
20         # No data has arrived yet, short sleep to reduce CPU load
21         time.sleep(0.001)

```

Read frames from a live camera

Incoming frames from a camera can be read sequentially frame-by-frame using the `dv::io::CameraInputBase::getNextFrame()`. Following is a minimal sample on how to read frames sequentially from a camera:

C++

Python

```

1 #include <dv-processing/io/camera/discovery.hpp>
2
3 #include <opencv2/highgui.hpp>
4
5 int main() {
6     // Open any camera
7     auto capture = dv::io::camera::open();
8
9     // Initiate a preview window
10    cv::namedWindow("Preview", cv::WINDOW_NORMAL);
11
12    // Run the loop while camera is still connected
13    while (capture->isRunning()) {
14        // Read a frame, check whether it is correct.
15        // The method does not wait for frame arrive, it returns immediately with
16        // the latest available frame or if no data is available, returns a
17        ↪`std::nullopt`.
18        if (const auto frame = capture->getNextFrame(); frame.has_value()) {
19            std::cout << *frame << std::endl;
20
21            // Show a preview of the image
22            cv::imshow("Preview", frame->image);
23        }
24    }
25 }

```

(continues on next page)

(continued from previous page)

```

23     cv::waitKey(2);
24 }
25
26 return 0;
27 }

```

```

1  import cv2 as cv
2  import dv_processing as dv
3
4  # Open any camera
5  capture = dv.io.camera.open()
6
7  # Initiate a preview window
8  cv.namedWindow("Preview", cv.WINDOW_NORMAL)
9
10 # Run the loop while camera is still connected
11 while capture.isRunning():
12     # Read a frame from the camera
13     frame = capture.getNextFrame()
14
15     # The method does not wait for frame arrive, it returns immediately with
16     # latest available frame or if no data is available, returns a `None`
17     if frame is not None:
18         # Print received packet time range
19         print(f"Received a frame at time [{frame.timestamp}]")
20
21         # Show a preview of the image
22         cv.imshow("Preview", frame.image)
23     cv.waitKey(2)

```

Read IMU data from a live camera

Incoming imu data from a camera can be read sequentially using the {cpp:func} `dv::io::CameraInputBase::getNextImuBatch()`. Following is a minimal sample on how to read imu data sequentially from a camera:

C++

Python

```

1  #include <dv-processing/io/camera/discovery.hpp>
2
3  #include <chrono>
4
5  int main() {
6      using namespace std::chrono_literals;
7
8      // Open any camera
9      auto capture = dv::io::camera::open();
10
11     // Run the loop while camera is still connected
12     while (capture->isRunning()) {
13         // Read IMU measurement batch, check whether it is correct.
14         // The method does not wait for data to arrive, it returns immediately with

```

(continues on next page)

(continued from previous page)

```

15     // the latest available imu data or if no data is available, returns a
↳ `std::nullopt`.
16     if (const auto imuBatch = capture->getNextImuBatch(); imuBatch.has_value() &&
↳ !imuBatch->empty()) {
17         std::cout << "Received " << imuBatch->size() << " IMU measurements" <<
↳ std::endl;
18     }
19     else {
20         // No data has arrived yet, short sleep to reduce CPU load.
21         std::this_thread::sleep_for(1ms);
22     }
23 }
24
25 return 0;
26 }

```

```

1 import time
2
3 import dv_processing as dv
4
5 # Open any camera
6 capture = dv.io.camera.open()
7
8 # Run the loop while camera is still connected
9 while capture.isRunning():
10     # Read a batch of IMU data from the camera
11     imu_batch = capture.getNextImuBatch()
12
13     # The method does not wait for data to arrive, it returns immediately with
14     # latest available data or if no data is available, returns a `None`
15     if imu_batch is not None and len(imu_batch) > 0:
16         # Print the time range of imu data
17         print(f"Received imu data within time range [{imu_batch[0].timestamp}; {imu_
↳ batch[-1].timestamp}]")
18     else:
19         time.sleep(0.001)

```

Read triggers from a live camera

Note

To understand what triggers are and where they come from, read more about them on our [documentation page](https://docs.inivation.com/hardware/hardware-advanced-usage/external-camera-sync.html)¹².

Incoming trigger data from a camera can be read sequentially using the `dv::io::CameraInputBase::getNextTriggerBatch()`. Following is a minimal sample on how to read trigger data sequentially from a camera:

C++

Python

¹² <https://docs.inivation.com/hardware/hardware-advanced-usage/external-camera-sync.html>

```

1  #include <dv-processing/io/camera/davis.hpp>
2
3  #include <chrono>
4
5  int main() {
6      using namespace std::chrono_literals;
7
8      // Open a DAVIS camera (DVXplorer also has triggers)
9      dv::io::camera::DAVIS capture{};
10
11     // Depending on the incoming signal, enable the detection of the desired type of
    ↳ pattern, here we enable everything.
12     // Enable rising edge detection
13     capture.setDetectorRisingEdges(true);
14     // Enable falling edge detection
15     capture.setDetectorFallingEdges(true);
16     // Enable detector
17     capture.setDetectorRunning(true);
18
19     // Run the loop while camera is still connected
20     while (capture.isRunning()) {
21         // Read trigger batch, check whether it is correct.
22         // The method does not wait for data to arrive, it returns immediately with
23         // the latest available data or if no data is available, returns a
    ↳ `std::nullopt`.
24         if (const auto triggers = capture.getNextTriggerBatch(); triggers.has_value()
    ↳ && !triggers->empty()) {
25             std::cout << "Received " << triggers->size() << " Triggers" << std::endl;
26         }
27         else {
28             // No data has arrived yet, short sleep to reduce CPU load.
29             std::this_thread::sleep_for(1ms);
30         }
31     }
32
33     return 0;
34 }

```

```

1  import time
2
3  import dv_processing as dv
4
5  # Open any camera
6  capture = dv.io.camera.open()
7
8  # Depending on the incoming signal, enable the detection of the desired type of
    ↳ pattern, here we enable everything.
9  # Enable rising edge detection
10 capture.setDetectorRisingEdges(True)
11 # Enable falling edge detection
12 capture.setDetectorFallingEdges(True)
13 # Enable detector

```

(continues on next page)

(continued from previous page)

```

14 capture.setDetectorRunning(True)
15
16 # Run the loop while camera is still connected
17 while capture.isRunning():
18     # Read a batch of triggers from the camera
19     triggers = capture.getNextTriggerBatch()
20
21     # The method does not wait for data arrive, it returns immediately with
22     # latest available data or if no data is available, returns a `None`
23     if triggers is not None and len(triggers) > 0:
24         # Print the time range of trigger data
25         print(f"Received trigger data within time range [{triggers[0].timestamp};
↪ {triggers[-1].timestamp}]")
26     else:
27         time.sleep(0.001)

```

Sample application - reading data from a live camera

An application reading multiple types of data from a live camera can be found among the code samples in the source code repository of the dv-processing library:

- Camera capture in C++¹³
- Camera capture in Python¹⁴

3.6.3 From a file

Data from iniVation cameras are usually recorded using the AEDAT4 file format. The dv-processing library provide tools for reading such files. This section contains explanations and samples on how data can be read from AEDAT4 files. More detailed information on the AEDAT4 file format can be found [here](#)¹⁵.

Inspecting AEDAT4 files

AEDAT4 file format supports recording of different data streams into single file, multiple cameras are also supported. The library provides a command-line utility for inspection of AEDAT4 files *dv-filestat*, it provides information on available streams recorded in it.

The utility provides information on the size, timestamp information, duration. More information about the utility can be found [here](#).

Opening a file

AEDAT4 files can be opened and read using the *dv::io::MonoCameraRecording* class. This class assumes that the recording was performed using a single camera. Following is a minimal sample code on opening a recording and printing information about it.

A file can be opened by providing its path in the filesystem:

C++

Python

¹³ <https://gitlab.com/inivation/dv/dv-processing/-/blob/master/samples/io/camera-capture>

¹⁴ https://gitlab.com/inivation/dv/dv-processing/-/blob/master/python/samples/camera_capture.py

¹⁵ <https://docs.inivation.com/software/software-advanced-usage/file-formats/aedat-4.0.html>

```

1  #include <dv-processing/io/mono_camera_recording.hpp>
2
3  int main() {
4      // Open a file
5      dv::io::MonoCameraRecording reader("path/to/file.aedat4");
6
7      // Get and print the camera name that data from recorded from
8      std::cout << "Opened an AEDAT4 file which contains data from [" << reader.
    ↪getCameraName() << "]" camera"
9          << std::endl;
10
11     return 0;
12 }

```

```

1  import dv_processing as dv
2
3  # Open a file
4  reader = dv.io.MonoCameraRecording("path/to/file.aedat4")
5
6  # Get and print the camera name that data from recorded from
7  print(f"Opened an AEDAT4 file which contains data from [{reader.getCameraName()}]_
    ↪camera")

```

Checking available streams

Data recordings might contain various data streams. The `dv::io::MonoCameraRecording` provides easy-to-use methods to inspect what data streams are available. Following sample code shows how to check for existence of various data streams:

C++

Python

```

1  #include <dv-processing/io/mono_camera_recording.hpp>
2
3  int main() {
4      // Store the file path
5      const std::string pathToFile = "path/to/file.aedat4";
6
7      // Open a file
8      dv::io::MonoCameraRecording reader(pathToFile);
9
10     // Print file path and camera name
11     std::cout << "Available streams in [" << pathToFile << "]:" << std::endl;
12
13     // Check if event stream is available
14     if (reader.isEventStreamAvailable()) {
15         // Check the resolution of event stream. Since the getEventResolution()_
    ↪method returns
16         // a std::optional, we use *operator to get the value. The method returns_
    ↪std::nullopt
17         // only in case the stream is unavailable, which is already checked.
18         const cv::Size resolution = *reader.getEventResolution();

```

(continues on next page)

(continued from previous page)

```

19
20     // Print that the stream is present and its resolution
21     std::cout << " * Event stream with resolution " << resolution << std::endl;
22 }
23
24 // Check if frame stream is available
25 if (reader.isFrameStreamAvailable()) {
26     // Check the resolution of frame stream. Since the getFrameResolution()
    ↪method returns
27     // a std::optional, we use *operator to get the value. The method returns
    ↪std::nullopt
28     // only in case the stream is unavailable, which is already checked.
29     const cv::Size resolution = *reader.getFrameResolution();
30
31     // Print that the stream is available and its resolution
32     std::cout << " * Frame stream with resolution " << resolution << std::endl;
33 }
34
35 // Check if IMU stream is available
36 if (reader.isImuStreamAvailable()) {
37     // Print that the IMU stream is available
38     std::cout << " * IMU stream" << std::endl;
39 }
40
41 // Check if trigger stream is available
42 if (reader.isTriggerStreamAvailable()) {
43     // Print that the trigger stream is available
44     std::cout << " * Trigger stream " << std::endl;
45 }
46
47 return 0;
48 }

```

```

1 import dv_processing as dv
2
3 # Store the file path
4 path_to_file = "path/to/file.aedat4"
5
6 # Open a file
7 reader = dv.io.MonoCameraRecording(path_to_file)
8
9 # Print file path and camera name
10 print(f"Checking available streams in [{path_to_file}] for camera name [{reader.
    ↪getCameraName()}]:")
11
12 # Check if event stream is available
13 if reader.isEventStreamAvailable():
14     # Check the resolution of event stream
15     resolution = reader.getEventResolution()
16
17     # Print that the stream is present and its resolution
18     print(f" * Event stream with resolution [{resolution[0]}x{resolution[1]}]")

```

(continues on next page)

(continued from previous page)

```

19
20 # Check if frame stream is available
21 if reader.isFrameStreamAvailable():
22     # Check the resolution of frame stream
23     resolution = reader.getFrameResolution()
24
25     # Print that the stream is available and its resolution
26     print(f" * Frame stream with resolution [{resolution[0]}x{resolution[1]}]")
27
28 # Check if IMU stream is available
29 if reader.isImuStreamAvailable():
30     # Print that the IMU stream is available
31     print(" * IMU stream")
32
33 # Check if trigger stream is available
34 if reader.isTriggerStreamAvailable():
35     # Print that the trigger stream is available
36     print(" * Trigger stream")

```

The dv-processing library also supports recording of other types, type agnostic methods are available as templated methods in C++, while Python only contains a limited set of named methods (since templating is unavailable in Python). The following sample shows the use of generic methods for checking the availability of certain streams with a name and a type:

```

1 #include <dv-processing/io/mono_camera_recording.hpp>
2
3 int main() {
4     // Open a file
5     dv::io::MonoCameraRecording reader("path/to/file.aedat4");
6
7     // Store some stream name
8     const std::string streamName = "poses";
9
10    // Check if such stream name is available and validate the data type of this
    ↪ stream
11    if (reader.isStreamAvailable("streamName") && reader.isStreamOfDataType<dv::Pose>(
    ↪ "streamName")) {
12        std::cout << "The file contains a stream named [" << streamName << "] and of
    ↪ data type [dv::Pose]" << std::endl;
13    }
14
15    return 0;
16 }

```

Read events from a file

Following sample reads events in batches while the stream has available data to read. While reading from a file, the `dv::io::MonoCameraRecording::getNextEventBatch()` will return data until the end of stream is reached, the `dv::io::MonoCameraRecording::isRunning()` method will return a false boolean when the end-of-file is reached for any stream. You can also use the `dv::io::MonoCameraRecording::isRunning(streamName)()` method to check for specific streams.

C++

Python

```
1 #include <dv-processing/io/mono_camera_recording.hpp>
2
3 int main() {
4     // Open a file
5     dv::io::MonoCameraRecording reader("path/to/file.aedat4");
6
7     // Run the loop while data is available
8     while (reader.isRunning()) {
9         // Read batch of events, check whether received data is correct.
10        if (const auto events = reader.getNextEventBatch(); events.has_value()) {
11            // Print received event packet information
12            std::cout << *events << std::endl;
13        }
14    }
15
16    return 0;
17 }
```

```
1 import dv_processing as dv
2
3 # Open any camera
4 reader = dv.io.MonoCameraRecording("path/to/file.aedat4")
5
6 # Run the loop while camera is still connected
7 while reader.isRunning():
8     # Read batch of events
9     events = reader.getNextEventBatch()
10    if events is not None:
11        # Print received packet time range
12        print(f"{events}")
```

Read frames from a file

Following sample reads frames in batches while the stream has available data to read. While reading from a file, the `dv::io::MonoCameraRecording::getNextFrame()` will return a frame until the end of stream is reached, the `dv::io::MonoCameraRecording::isRunning()` method will return a false boolean when the end-of-file is reached for any stream. You can also use the `dv::io::MonoCameraRecording::isRunning(streamName)()` method to check for specific streams.

C++

Python

```
1 #include <dv-processing/io/mono_camera_recording.hpp>
2
3 #include <opencv2/highgui.hpp>
4
5 int main() {
6     // Open a file
7     dv::io::MonoCameraRecording reader("path/to/file.aedat4");
8
9     // Initiate a preview window
```

(continues on next page)

(continued from previous page)

```

10 cv::namedWindow("Preview", cv::WINDOW_NORMAL);
11
12 // Variable to store the previous frame timestamp for correct playback
13 std::optional<int64_t> lastTimestamp = std::nullopt;
14
15 // Run the loop while data is available
16 while (reader.isRunning()) {
17     // Read a frame, check whether it is correct.
18     // The method does not wait for frame arrive, it returns immediately with
19     // latest available frame or if no data is available, returns a
    ↳ `std::nullopt`.
20     if (const auto frame = reader.getNextFrame(); frame.has_value()) {
21         // Print information about received frame
22         std::cout << *frame << std::endl;
23
24         // Show a preview of the image
25         cv::imshow("Preview", frame->image);
26
27         // Calculate the delay between last and current frame, divide by 1000 to
    ↳ convert microseconds
28         // to milliseconds
29         const int delay = lastTimestamp.has_value() ? (frame->timestamp -
    ↳ *lastTimestamp) / 1000 : 2;
30
31         // Perform the sleep
32         cv::waitKey(delay);
33
34         // Store timestamp for the next frame
35         lastTimestamp = frame->timestamp;
36     }
37 }
38
39 return 0;
40 }

```

```

1 import dv_processing as dv
2 import cv2 as cv
3
4 # Open a file
5 reader = dv.io.MonoCameraRecording("path/to/file.aedat4")
6
7 # Initiate a preview window
8 cv.namedWindow("Preview", cv.WINDOW_NORMAL)
9
10 # Variable to store the previous frame timestamp for correct playback
11 lastTimestamp = None
12
13 # Run the loop while camera is still connected
14 while reader.isRunning():
15     # Read a frame from the camera
16     frame = reader.getNextFrame()
17

```

(continues on next page)

(continued from previous page)

```

18     if frame is not None:
19         # Print the timestamp of the received frame
20         print(f"Received a frame at time [{frame.timestamp}]")
21
22         # Show a preview of the image
23         cv.imshow("Preview", frame.image)
24
25         # Calculate the delay between last and current frame, divide by 1000 to
↳convert microseconds
26         # to milliseconds
27         delay = (2 if lastTimestamp is None else (frame.timestamp - lastTimestamp) /
↳1000)
28
29         # Perform the sleep
30         cv.waitKey(delay)
31
32         # Store timestamp for the next frame
33         lastTimestamp = frame.timestamp

```

Read IMU data from a file

Following sample reads imu data in batches while the stream has available data to read. While reading from a file, the `dv::io::MonoCameraRecording::getNextImuBatch()` will return an IMU measurement batch until the end of stream is reached, the `dv::io::MonoCameraRecording::isRunning()` method will return a false boolean when the end-of-file is reached for any stream. You can also use the `dv::io::MonoCameraRecording::isRunning(streamName)()` method to check for specific streams.

C++

Python

```

1  #include <dv-processing/io/mono_camera_recording.hpp>
2
3  int main() {
4      // Open a file
5      dv::io::MonoCameraRecording reader("path/to/file.aedat4");
6
7      // Run the loop while data is available
8      while (reader.isRunning()) {
9          // Read IMU measurement batch, check whether it is correct.
10         if (const auto imuBatch = reader.getNextImuBatch(); imuBatch.has_value() && !
↳imuBatch->empty()) {
11             // Print IMU batch information
12             std::cout << "Received " << imuBatch->size() << " IMU measurements" <<
↳std::endl;
13         }
14     }
15
16     return 0;
17 }

```

```

1  import dv_processing as dv
2

```

(continues on next page)

(continued from previous page)

```

3  # Open a file
4  reader = dv.io.MonoCameraRecording("path/to/file.aedat4")
5
6  # Run the loop while stream contains data
7  while reader.isRunning():
8      # Read a batch of IMU data from the camera
9      imu_batch = reader.getNextImuBatch()
10     if imu_batch is not None and len(imu_batch) > 0:
11         # Print the info of the imu data
12         print(f"Received {len(imu_batch)} IMU measurements")

```

Read triggers from a file

Following sample reads triggers in batches while the stream has available data to read. While reading from a file, the `dv::io::MonoCameraRecording::getNextTriggerBatch()` will return a batch of triggers until the end of stream is reached, the `dv::io::MonoCameraRecording::isRunning()` method will return a false boolean when the end-of-file is reached for any stream. You can also use the `dv::io::MonoCameraRecording::isRunning(streamName)()` method to check for specific streams.

C++

Python

```

1  #include <dv-processing/io/mono_camera_recording.hpp>
2
3  int main() {
4      // Open a file
5      dv::io::MonoCameraRecording reader("path/to/file.aedat4");
6
7      // Run the loop while data is available
8      while (reader.isRunning()) {
9          // Read trigger batch, check whether it is correct.
10         if (const auto triggers = reader.getNextTriggerBatch(); triggers.has_value() &
11             !triggers->empty()) {
12             // Print the trigger batch information
13             std::cout << "Received " << triggers->size() << " triggers" << std::endl;
14         }
15     }
16
17     return 0;
18 }

```

```

1  import dv_processing as dv
2
3  # Open a file
4  reader = dv.io.MonoCameraRecording("path/to/file.aedat4")
5
6  # Run the loop while camera is still connected
7  while reader.isRunning():
8      # Read a a batch of triggers from the camera
9      triggers = reader.getNextTriggerBatch()
10
11     # Check whether batch is valid and contains data

```

(continues on next page)

(continued from previous page)

```

12     if triggers is not None and len(triggers) > 0:
13         # Print the trigger batch information
14         print(f"Received {len(triggers)} triggers")

```

[Advanced] Reading custom data types

The previous samples show how to use named functions to read different data types. The C++ API provides templated methods to read any type of data. Below is a sample that shows how to read data using the generic templated API:

Note

Since templated methods are only available in C++, the generic writing methods are only available in the C++ API.

```

1  #include <dv-processing/data/timed_keypoint_base.hpp>
2  #include <dv-processing/io/mono_camera_recording.hpp>
3
4  int main() {
5      // Open a file
6      dv::io::MonoCameraRecording reader("path/to/file.aedat4");
7
8      // Define and contain a stream name in a variable
9      const std::string stream = "keypoints";
10
11     // Check whether a timed-keypoint stream is available
12     if (!reader.isStreamAvailable(stream) || !reader.isStreamOfDataType
↪ <dv::TimedKeyPointPacket>(stream)) {
13         throw dv::exceptions::RuntimeError("Stream named 'keypoints' not found");
14     }
15
16     // Run the loop while data is available
17     while (reader.isRunning()) {
18         // Read timed keypoint batch, check whether it is correct.
19         if (const auto keypoints = reader.getNextStreamPacket<dv::TimedKeyPointPacket>
↪ (stream); keypoints.has_value()) {
20             // Print the number of keypoints read
21             std::cout << "Read " << keypoints->elements.size() << " timed keypoints" <
↪ < std::endl;
22         }
23     }
24
25     return 0;
26 }

```

Sample application - reading data from a recorded AEDAT4 file

An application reading multiple types of data from an AEDAT4 file can be found in the source code repository of the dv-processing library:

- AEDAT4 player in C++¹⁶

¹⁶ <https://gitlab.com/inivation/dv/dv-processing/-/blob/master/samples/io/aedat4-reader>

- AEDAT4 to CSV converter in Python¹⁷

3.7 Writing camera data

The dv-processing library provides a convenient way of recording camera data into AEDAT4 files. These files can record multiple data-types and can be read using existing tools in the library. The DV GUI software also provides ways to play back these recordings.

3.7.1 Writing a file

Files can be written using the `dv::io::MonoCameraWriter` class. Since the writer supports multiple data type configuration, it needs to know apriori what data streams are needed for writing. They are declared using a `dv::io::MonoCameraWriter::Config` structure and is passed to the constructor of the writer class to correctly initialize the required streams in the file header.

Defining output streams from an existing camera reader handle

As mentioned in the previous paragraph, the number of output streams and their data-types needs to be known for the writer. Output streams definitions can be generated by an input camera capture. In this case, the writer class will inspect what streams the capture can output and will create output streams for them. Following is an example on how to initialize a writer with a camera capture class:

C++

Python

```

1  #include <dv-processing/io/camera/discovery.hpp>
2  #include <dv-processing/io/mono_camera_writer.hpp>
3
4  int main() {
5      // Open any camera
6      auto capture = dv::io::camera::open();
7
8      // Create the writer instance, writer will inspect the capture capabilities and
9      ↪ create output
10     // streams for all available data streams from the capture instance.
11     dv::io::MonoCameraWriter writer("mono_writer_sample.aedat4", *capture);
12
13     // Print which streams were configured
14     std::cout << std::boolalpha;
15     std::cout << "Is event stream available? " << writer.isEventStreamConfigured() <<
16     ↪ std::endl;
17     std::cout << "Is frame stream available? " << writer.isFrameStreamConfigured() <<
18     ↪ std::endl;
19     std::cout << "Is imu stream available? " << writer.isImuStreamConfigured() <<
20     ↪ std::endl;
21     std::cout << "Is trigger stream available? " << writer.
22     ↪ isTriggerStreamConfigured() << std::endl;
23
24     return 0;
25 }
```

¹⁷ https://gitlab.com/inivation/dv/dv-processing/-/blob/master/python/samples/aedat4_reader.py

```

1 import dv_processing as dv
2
3 capture = dv.io.camera.open()
4
5 writer = dv.io.MonoCameraWriter("mono_writer_sample.aedat4", capture)
6
7 print(f"Is event stream available? {str(writer.isEventStreamConfigured())}")
8 print(f"Is frame stream available? {str(writer.isFrameStreamConfigured())}")
9 print(f"Is imu stream available? {str(writer.isImuStreamConfigured())}")
10 print(f"Is trigger stream available? {str(writer.isTriggerStreamConfigured())}")

```

Defining output streams manually

Output streams definition can be manually defined in the config structure, following is a sample that manually defines event, frame, IMU, and trigger data-type streams and creates a writer instance:

C++

Python

```

1 #include <dv-processing/io/mono_camera_writer.hpp>
2
3 int main() {
4     dv::io::MonoCameraWriter::Config config("DVXplorer_sample");
5
6     // Sample VGA resolution, same as the DVXplorer camera
7     const cv::Size resolution(640, 480);
8
9     // Add an event stream with a resolution
10    config.addEventStream(resolution);
11
12    // Add frame stream with a resolution
13    config.addFrameStream(resolution);
14
15    // Add IMU stream
16    config.addImuStream();
17
18    // Add trigger stream
19    config.addTriggerStream();
20
21    // Create the writer instance with the configuration structure
22    dv::io::MonoCameraWriter writer("mono_writer_sample.aedat4", config);
23
24    // Print which streams were configured
25    std::cout << std::boolalpha;
26    std::cout << "Is event stream available? " << writer.isEventStreamConfigured() <<
↪std::endl;
27    std::cout << "Is frame stream available? " << writer.isFrameStreamConfigured() <<
↪std::endl;
28    std::cout << "Is imu stream available? " << writer.isImuStreamConfigured() <<
↪std::endl;
29    std::cout << "Is trigger stream available? " << writer.
↪isTriggerStreamConfigured() << std::endl;
30

```

(continues on next page)

(continued from previous page)

```

31     return 0;
32 }

```

```

1  import dv_processing as dv
2
3  config = dv.io.MonoCameraWriter.Config("DVXplorer_sample")
4
5  # Define VGA resolution for this camera
6  resolution = (640, 480)
7
8  # Add an event stream with a resolution
9  config.addEventStream(resolution)
10
11 # Add frame stream with a resolution
12 config.addFrameStream(resolution)
13
14 # Add IMU stream
15 config.addImuStream()
16
17 # Add trigger stream
18 config.addTriggerStream()
19
20 # Create the writer instance with the configuration structure
21 writer = dv.io.MonoCameraWriter("mono_writer_sample.aedat4", config)
22
23 print(f"Is event stream available? {str(writer.isEventStreamConfigured())}")
24 print(f"Is frame stream available? {str(writer.isFrameStreamConfigured())}")
25 print(f"Is imu stream available? {str(writer.isImuStreamConfigured())}")
26 print(f"Is trigger stream available? {str(writer.isTriggerStreamConfigured())}")

```

Defining output streams using predefined templates

The `dv::io::MonoCameraWriter` class provides some templates for configuration to reduce the code lines. Following is a sample how to use a named method that generates a config for writing data:

C++

Python

```

1  #include <dv-processing/io/mono_camera_writer.hpp>
2
3  int main() {
4      // Create a DVS config - events, imu, and triggers are going to be enabled
5      const auto config = dv::io::MonoCameraWriter::DVSConfig("DVXplorer_sample",
6      ↪ cv::Size(640, 480));
7
8      // Create the writer instance with the configuration structure
9      dv::io::MonoCameraWriter writer("mono_writer_sample.aedat4", config);
10
11     // Print which streams were configured
12     std::cout << std::boolalpha;
13     std::cout << "Is event stream available? " << writer.isEventStreamConfigured() <<
14     ↪ std::endl;

```

(continues on next page)

(continued from previous page)

```

13     std::cout << "Is frame stream available? " << writer.isFrameStreamConfigured() << "\n";
14     std::cout << "Is imu stream available? " << writer.isImuStreamConfigured() << "\n";
15     std::cout << "Is trigger stream available? " << writer.isTriggerStreamConfigured() << "\n";
16
17     return 0;
18 }

```

```

1 import dv_processing as dv
2
3 # Create a DVS config - events, imu, and triggers are going to be enabled
4 config = dv.io.MonoCameraWriter.DVSConfig("DVXplorer_sample", (640, 480))
5
6 # Create the writer instance with the configuration structure
7 writer = dv.io.MonoCameraWriter("mono_writer_sample.aedat4", config)
8
9 # Print which streams were configured
10 print(f"Is event stream available? {str(writer.isEventStreamConfigured())}")
11 print(f"Is frame stream available? {str(writer.isFrameStreamConfigured())}")
12 print(f"Is imu stream available? {str(writer.isImuStreamConfigured())}")
13 print(f"Is trigger stream available? {str(writer.isTriggerStreamConfigured())}")

```

A list of available output stream configurations is available under the `dv::io::MonoCameraWriter` class:

- EventOnlyConfig - a single output stream “events”
- FrameOnlyConfig - a single output stream “frames”
- DVSConfig - three output streams: “events”, “imu”, and “trigger”
- DAVISConfig - four output streams: “events”, “frames”, “imu”, and “trigger”

3.7.2 Writing data to the file

The writer instance can be used to write data with the output streams configured. The following sections provide sample code on how to write individual data types using the writer class.

Writing events

The following sample shows an example on how to write event data using the `dv::io::MonoCameraWriter` class:

C++

Python

```

1 #include <dv-processing/data/generate.hpp>
2 #include <dv-processing/io/mono_camera_writer.hpp>
3
4 int main() {
5     // Sample VGA resolution, same as the DVXplorer camera
6     const cv::Size resolution(640, 480);
7
8     // Event only configuration

```

(continues on next page)

(continued from previous page)

```

9     const auto config = dv::io::MonoCameraWriter::EventOnlyConfig("DVXplorer_sample",
↪resolution);
10
11     // Create the writer instance, it will only have a single event output stream.
12     dv::io::MonoCameraWriter writer("mono_writer_sample.aedat4", config);
13
14     // Write 100 packet of event data
15     for (int i = 0; i < 100; i++) {
16         // EventStore requires strictly monotonically increasing data, generate
17         // a timestamp from the iteration counter value
18         const int64_t timestamp = i * 10000;
19
20         // Generate sample event batch
21         dv::EventStore events = dv::data::generate::dvLogoAsEvents(timestamp,
↪resolution);
22
23         // Write the packet using the writer, the data is not going be written at the
↪exact
24         // time of the call to this function, it is only guaranteed to be written
↪after
25         // the writer instance is destroyed (destructor has completed)
26         writer.writeEvents(events);
27     }
28
29     return 0;
30 }

```

```

1 import dv_processing as dv
2
3 # Sample VGA resolution, same as the DVXplorer camera
4 resolution = (640, 480)
5
6 # Event only configuration
7 config = dv.io.MonoCameraWriter.EventOnlyConfig("DVXplorer_sample", resolution)
8
9 # Create the writer instance, it will only have a single event output stream.
10 writer = dv.io.MonoCameraWriter("mono_writer_sample.aedat4", config)
11
12 # Write 100 packet of event data
13 for i in range(100):
14     # EventStore requires strictly monotonically increasing data, generate
15     # a timestamp from the iteration counter value
16     timestamp = i * 1000
17
18     # Empty event store
19     events = dv.data.generate.dvLogoAsEvents(timestamp, resolution)
20
21     # Write the packet using the writer, the data is not going be written at the exact
22     # time of the call to this function, it is only guaranteed to be written after
23     # the writer instance is destroyed (destructor has completed)
24     writer.writeEvents(events)

```

Writing frames

The following sample shows an example on how to write image frames using the `dv::io::MonoCameraWriter` class:

C++

Python

```
1 #include <dv-processing/io/mono_camera_writer.hpp>
2
3 int main() {
4     // Sample VGA resolution, same as the DVXplorer camera
5     const cv::Size resolution(640, 480);
6
7     // Frame only configuration
8     const auto config = dv::io::MonoCameraWriter::FrameOnlyConfig("DVXplorer_sample",
9 ↪resolution);
10
11     // Create the writer instance, it will only have a single frame output stream.
12     dv::io::MonoCameraWriter writer("mono_writer_sample.aedat4", config);
13
14     // Write 10 image frames
15     for (int i = 0; i < 10; i++) {
16         // Initialize a white image
17         cv::Mat image(resolution, CV_8UC3, cv::Scalar(255, 255, 255));
18
19         // Generate some monotonically increasing timestamp
20         const int64_t timestamp = i * 1000;
21
22         // Encapsulate the image in a frame that has a timestamp, this does not copy
23 ↪the pixel data
24         dv::Frame frame(timestamp, image);
25
26         // Write the frame
27         writer.writeFrame(frame);
28     }
29     return 0;
30 }
```

```
1 import dv_processing as dv
2 import numpy as np
3
4 # Frame only configuration
5 config = dv.io.MonoCameraWriter.FrameOnlyConfig("DVXplorer_sample", (640, 480))
6
7 # Create the writer instance, it will only have a single frame output stream
8 writer = dv.io.MonoCameraWriter("mono_writer_sample.aedat4", config)
9
10 # Write 10 image frames
11 for i in range(10):
12     # Initialize a white image
13     image = np.full((480, 640, 3), fill_value=255, dtype=np.uint8)
14
15     # Generate some monotonically increasing timestamp
```

(continues on next page)

(continued from previous page)

```

16     timestamp = i * 1000
17
18     # Encapsulate the image in a frame that has a timestamp, this does not copy the
19     ↪ pixel data
20     frame = dv.Frame(timestamp, image)
21
22     # Write the frame
23     writer.writeFrame(frame)

```

Writing IMU data

The following sample shows an example on how to write IMU measurement data using the `dv::io::MonoCameraWriter` class:

C++

Python

```

1  #include <dv-processing/data/generate.hpp>
2  #include <dv-processing/io/mono_camera_writer.hpp>
3
4  int main() {
5      // IMU data only configuration
6      auto config = dv::io::MonoCameraWriter::Config("DVXplorer_sample");
7      config.addImuStream();
8
9      // Create the writer instance, it will only have a single IMU data output stream
10     dv::io::MonoCameraWriter writer("mono_writer_sample.aedat4", config);
11
12     // Write 100 IMU measurements
13     for (int i = 0; i < 100; i++) {
14         // Generate some monotonically increasing timestamp
15         const int64_t timestamp = i * 1000;
16
17         // Some sample measurements - no rotation on gyro and a one-G gravity on Y
18         ↪ axis of accelerometer
19         const dv::IMU measurement = dv::data::generate::levelImuWithNoise(timestamp);
20
21         // Write the measurement
22         writer.writeImu(measurement);
23     }
24
25     return 0;

```

```

1  import dv_processing as dv
2
3  # IMU data only configuration
4  config = dv.io.MonoCameraWriter.Config("DVXplorer_sample")
5  config.addImuStream()
6
7  # Create the writer instance, it will only have a single IMU data output stream
8  writer = dv.io.MonoCameraWriter("mono_writer_sample.aedat4", config)

```

(continues on next page)

(continued from previous page)

```

9
10 # Write 100 IMU measurements
11 for i in range(100):
12     # Generate some monotonically increasing timestamp
13     timestamp = i * 1000
14
15     # Single IMU measurement instance
16     measurement = dv.data.generate.levelImuWithNoise(timestamp)
17
18     # Write the measurement
19     writer.writeImu(measurement)

```

Writing trigger data

The following sample shows an example on how to write triggers using the `dv::io::MonoCameraWriter` class:

C++

Python

```

1 #include <dv-processing/io/mono_camera_writer.hpp>
2
3 int main() {
4     // Trigger data only configuration
5     auto config = dv::io::MonoCameraWriter::Config("DVXplorer_sample");
6     config.addTriggerStream();
7
8     // Create the writer instance, it will only have a single trigger output stream
9     dv::io::MonoCameraWriter writer("mono_writer_sample.aedat4", config);
10
11     // Write 100 triggers
12     for (int i = 0; i < 100; i++) {
13         // Generate some monotonically increasing timestamp
14         const int64_t timestamp = i * 1000;
15
16         // Single trigger instance, let's say this is some signal from external source
17         dv::Trigger trigger(timestamp, dv::TriggerType::EXTERNAL_GENERATOR_RISING_
↪EDGE);
18
19         // Write the trigger value
20         writer.writeTrigger(trigger);
21     }
22
23     return 0;
24 }

```

```

1 import dv_processing as dv
2
3 # Trigger data only configuration
4 config = dv.io.MonoCameraWriter.Config("DVXplorer_sample")
5 config.addTriggerStream()
6
7 # Create the writer instance, it will only have a single trigger output stream

```

(continues on next page)

(continued from previous page)

```

8  writer = dv.io.MonoCameraWriter("mono_writer_sample.aedat4", config)
9
10 # Write 100 triggers
11 for i in range(100):
12     # Generate some monotonically increasing timestamp
13     timestamp = i * 1000
14
15     # Single trigger instance, let's say this is some signal from external source
16     trigger = dv.Trigger(timestamp, dv.TriggerType.EXTERNAL_GENERATOR_RISING_EDGE)
17
18     # Write the trigger value
19     writer.writeTrigger(trigger)

```

[Advanced] Writing custom data types

The previous samples show how to use named functions for writing different data types into a file. C++ API provides templated methods to write any type of data. Below is sample that shows how to write data using the generic templated API:

Note

Since templated methods are only available in C++, the generic writing methods are only available in the C++ API.

```

1  #include <dv-processing/io/mono_camera_writer.hpp>
2
3  int main() {
4      // Create a stream named "keypoints" for a data type of TimedKeyPoint packet
5      auto config = dv::io::MonoCameraWriter::Config("DVXplorer_sample");
6
7      // Define and contain a stream name in a variable
8      const std::string streamName = "keypoints";
9
10     // Add an output stream with a packet type and a defined stream name
11     config.addStream<dv::TimedKeyPointPacket>(streamName);
12
13     // Initialize the writer
14     dv::io::MonoCameraWriter writer("mono_writer_sample.aedat4", config);
15
16     // Let's create 10 packets of key-points
17     for (int i = 0; i < 10; i++) {
18         // Create a packet for writing
19         dv::TimedKeyPointPacket packet;
20
21         // Generate some monotonically increasing timestamp
22         const int64_t timestamp = i * 1000;
23
24         // 10 Let's generate 10 key-points for the packet
25         for (int y = 0; y < 10; y++) {
26             // Using emplace_back to directly allocate and insert the keypoint at the
27             ↪end of the vector

```

(continues on next page)

(continued from previous page)

```

27         packet.elements.emplace_back(dv::Point2f(5.f, 5.f), 10.f, 3.f, 1.f, 0, -1,
↪ timestamp);
28     }
29
30     // Write the packet using, requires a stream name that is the same with the
31     writer.writePacket(packet, streamName);
32 }
33
34 return 0;
35 }

```

Sample application - recording data from live camera

An application writing multiple types of data from a live camera can be found among the code samples in the source code repository of the dv-processing library:

- Camera writer in C++¹⁸
- Camera capture in Python¹⁹

3.8 Network streaming

The dv-processing library an easy to integrate solution to stream data over network TCP connections with optional encryption for security or through local sockets. This achieved by using client-server paradigm implemented using two classes: `dv::io::NetworkWriter` and `dv::io::NetworkReader`. The `dv::io::NetworkWriter` acts as a server, supports multiple concurrent clients, and streams a single data type stream to the clients. The `dv::io::NetworkReader` connects to a server and is able to receive the streamed data.

This tutorial will introduce basic usage of these classes to stream event data using a network writer and receive the same data using network reader.

3.8.1 Network streaming server

The following sample implements a network streaming server using `dv::io::NetworkWriter`, that streams periodic event data, that is synthesized in software:

C++

Python

```

1  #include <dv-processing/data/generate.hpp>
2  #include <dv-processing/io/network_writer.hpp>
3
4  int main() {
5      using namespace std::chrono_literals;
6
7      // Define image space resolution dimensions
8      const cv::Size resolution(200, 200);
9
10     // Define output event stream with valid resolution
11     const dv::io::Stream stream = dv::io::Stream::EventStream(0, "events", "TEST_DATA

```

(continues on next page)

¹⁸ <https://gitlab.com/inivation/dv/dv-processing/-/blob/master/samples/io/mono-live-writer>

¹⁹ https://gitlab.com/inivation/dv/dv-processing/-/blob/master/python/samples/writer_mono_dvs.py

(continued from previous page)

```

12  ↪", resolution);
13
14  // Initiate the server, needs stream definition to initiate
15  dv::io::NetworkWriter server("0.0.0.0", 10101, stream);
16
17  // Print the ready state of the server
18  std::cout << "Waiting for connections..." << std::endl;
19
20  // Stream interval defines the packet frequency for this sample
21  const dv::Duration streamInterval = 10ms;
22
23  // Starting coordinates of the rectangle data that is going to be sent out in_
24  ↪this sample
25  cv::Point2i offset(0, 0);
26
27  // Rectangle size in pixels
28  const cv::Point2i rectSize(20, 20);
29
30  // A boolean variable used to define movement direction of the rectangle
31  bool direction = true;
32
33  // Run indefinitely
34  while (true) {
35      // Do not produce output if there are no connected clients
36      if (server.getClientCount() > 0) {
37          // Generate the rectangle at given offset position
38          const dv::EventStore events =_
39          ↪dv::data::generate::eventRectangle(dv::now(), offset, offset + rectSize);
40
41          // Increase or decrease the position coordinates depending on the
42          ↪"direction" boolean
43          offset = direction ? offset + cv::Point2i(1, 1) : offset - cv::Point2i(1,_
44          ↪1);
45
46          // Check if the rectangle coordinates reaches borders of the image
47          if (offset.x == 0 || offset.y == 0 || offset.x + rectSize.x == resolution._
48          ↪width
49              || offset.y + rectSize.y == resolution.height) {
50              // Reverse the motion direction
51              direction = !direction;
52          }
53
54          // Send it out to clients
55          server.writeEvents(events);
56      }
57
58      // Sleep the application for the streaming interval duration
59      std::this_thread::sleep_for(streamInterval);
60  }
61
62  return EXIT_SUCCESS;
63 }

```

```

1 import dv_processing as dv
2 from datetime import timedelta
3 import time
4
5 # Define image space resolution dimensions
6 resolution = (200, 200)
7
8 # Define output event stream with valid resolution
9 stream = dv.io.Stream.EventStream(0, "events", "TEST_DATA", resolution)
10
11 # Initiate the server, needs stream definition to initiate
12 server = dv.io.NetworkWriter("0.0.0.0", 10101, stream)
13
14 # Print the ready state of the server
15 print("Waiting for connections...")
16
17 # Stream interval defines the packet frequency for this sample
18 streamInterval = timedelta(milliseconds=10)
19
20 # Starting coordinates of the rectangle data that is going to be sent out in this_
  ↳sample
21 offset = (0, 0)
22
23 # Rectangle size in pixels
24 rectSize = (20, 20)
25
26 # A boolean variable used to define movement direction of the rectangle
27 direction = True
28
29 # Run indefinitely
30 while True:
31     # Do not produce output if there are no connected clients
32     if server.getClientCount() > 0:
33         # Generate the rectangle at given offset position
34         events = dv.data.generate.eventRectangle(dv.now(), offset, (offset[0] +_
  ↳rectSize[0], offset[1] + rectSize[1]))
35
36         # Increase or decrease the position coordinates depending on the "direction"_
  ↳boolean
37         offset = (offset[0] + 1, offset[1] + 1) if direction else (offset[0] - 1,_
  ↳offset[1] - 1)
38
39         # Check if the rectangle coordinates reaches borders of the image
40         if offset[0] == 0 or offset[1] == 0 or offset[0] + rectSize[0] ==_
  ↳resolution[0] or offset[1] + rectSize[1] == \
41             resolution[1]:
42             # Reverse the motion direction
43             direction = not direction
44
45         # Send it out to clients
46         server.writeEvents(events)
47
48     # Sleep the application for the streaming interval duration

```

(continues on next page)

(continued from previous page)

```
49 time.sleep(streamInterval.total_seconds())
```

Compiling and running the sample above will run an event streaming server, which is accessible on port 10101 on all available network devices. The instance will wait for a client to connect, next chapter will introduce a minimal client application to receive and visualize the streamed data.

3.8.2 Network streaming client

The following code sample provide a basic usage of `dv::io::NetworkReader` to receive network streamed data.

C++

Python

```
1  #include <dv-processing/io/network_reader.hpp>
2  #include <dv-processing/visualization/event_visualizer.hpp>
3
4  #include <opencv2/highgui.hpp>
5
6  int main() {
7      using namespace std::chrono_literals;
8
9      // Initiate the client connection to the same port and localhost loopback address
10     dv::io::NetworkReader client("127.0.0.1", 10101);
11
12     // Validate that this client is connected to an event data stream
13     if (!client.isEventStreamAvailable()) {
14         throw dv::exceptions::RuntimeError("Server does not provide event data!");
15     }
16
17     // Initialize the event visualizer with server reported sensor resolution
18     dv::visualization::EventVisualizer visualizer(client.getEventResolution().
19     ↪value());
20
21     // Create a preview window to show the visualized events
22     cv::namedWindow("Preview", cv::WINDOW_NORMAL);
23
24     // Declare an event stream slicer to synchronized event data packets
25     dv::EventStreamSlicer slicer;
26
27     // Perform visualization every 10 milliseconds, which should match the server_
28     ↪publishing frequency
29     slicer.doEveryTimeInterval(10ms, [&visualizer](const dv::EventStore &events) {
30         // Display preview image
31         cv::imshow("Preview", visualizer.generateImage(events));
32
33         // Short sleep, if user clicks escape key (code 27), exit the application
34         if (cv::waitKey(2) == 27) {
35             exit(0);
36         }
37     });
38
39     // While client is connected
```

(continues on next page)

(continued from previous page)

```

38     while (client.isRunning()) {
39         // Read the event data, validate, and feed into the slicer
40         if (const auto events = client.getNextEventBatch(); events.has_value()) {
41             slicer.accept(*events);
42         }
43     }
44
45     return EXIT_SUCCESS;
46 }

```

```

1  import dv_processing as dv
2  import cv2 as cv
3  from datetime import timedelta
4
5  # Initiate the client connection to the same port and localhost loopback address
6  client = dv.io.NetworkReader("127.0.0.1", 10101)
7
8  # Validate that this client is connected to an event data stream
9  if not client.isEventStreamAvailable():
10     raise RuntimeError("Server does not provide event data!")
11
12 # Initialize the event visualizer with server reported sensor resolution
13 visualizer = dv.visualization.EventVisualizer(client.getEventResolution())
14
15 # Create a preview window to show the visualized events
16 cv.namedWindow("Preview", cv.WINDOW_NORMAL)
17
18 # Declare an event stream slicer to synchronized event data packets
19 slicer = dv.EventStreamSlicer()
20
21
22 # Callback method to show the generated event visualization
23 def show_preview(events: dv.EventStore):
24     # Display preview image
25     cv.imshow("Preview", visualizer.generateImage(events))
26
27     # Short sleep, if user clicks escape key (code 27), exit the application
28     if cv.waitKey(2) == 27:
29         exit(0)
30
31
32 # Perform visualization every 10 milliseconds, which should match the server_
33 ↪publishing frequency
34 slicer.doEveryTimeInterval(timedelta(milliseconds=10), show_preview)
35
36 # While client is connected
37 while True:
38     # Read the event data
39     events = client.getNextEventBatch()
40
41     # Validate the data and feed into the slicer
42     if events is not None:

```

(continues on next page)

(continued from previous page)

42

```
slicer.accept(events)
```

This code sample connects to the port 10101 on local loopback network, receives event data stream and uses `dv::visualization::EventVisualizer` to generate a preview image of the event data.

3.8.3 Running the samples

We can now combine both samples to stream the synthetic events and visualize the output. First, run the server sample application, when it prints “Waiting for connections...”, launch the second client application, it should show the preview window with a small rectangle moving from corner to corner.

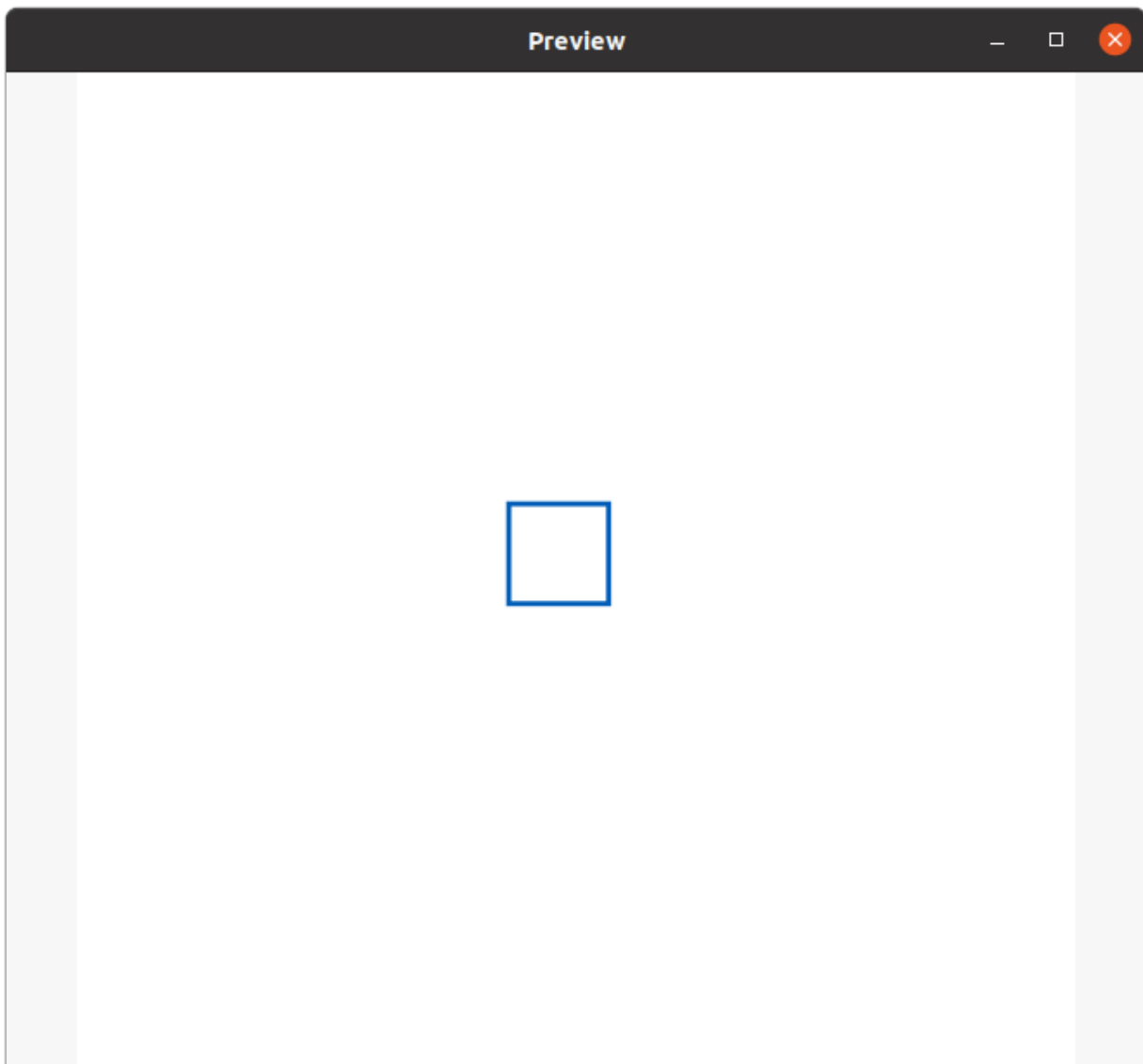


Fig. 17: Preview of the rectangle that is received as event packets, the triangle is expected to move diagonally through the window pixel space.

VISION ALGORITHMS

This chapter describes the available event processing algorithms that can be building blocks for computer vision with event cameras. Most notable algorithms and features:

- Camera geometry - sensor calibration, pixel projections, lens undistortion operations;
- Feature detection and tracking
- Minimal kinematics routines
- Mean-shift clustering

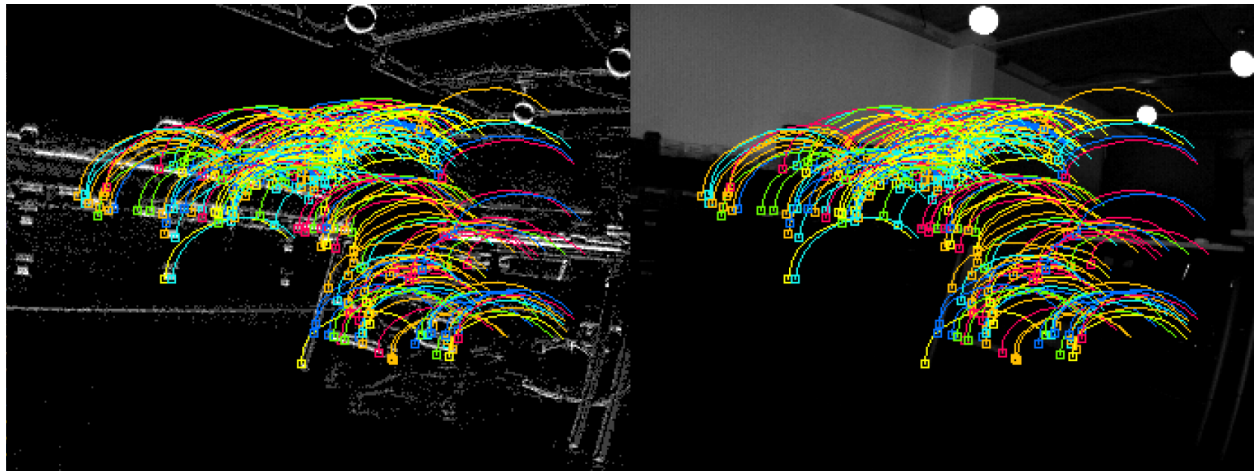


Fig. 1: Tracked features on frame and event streams from a camera.

4.1 Camera Geometry

This section introduces camera calibration related classes and primitives available in the dv-processing library.

4.1.1 Calibration files

Multiple calibration file formats were introduced in different DV software components. Single / stereo camera calibration files were introduced in DV camera calibration module, which produces these files after a successful calibration. A new yaml calibration file format is introduced in dv-processing, which can store calibration information not only for a single / stereo camera setups, but multi-camera setups with any number of cameras as well as calibration for IMU, and its noise parameters.

All of these calibration files can be opened and created by a provided `dv::camera::CalibrationSet`. The `CalibrationSet` class contains three main types of calibration parameters:

- `dv::camera::calibrations::CameraCalibration` - Single camera intrinsic calibration: pinhole camera and distortion model parameters. It also contains extrinsic transformation matrix which describes physical displacement of a camera w.r.t. one of the cameras. Usually first camera denoted “C0” is selected as a reference which will have an identity matrix for its transformation and all the other cameras will contain transformation matrices which describes the relationship to the first camera.
- `dv::camera::calibrations::StereoCalibration` - Stereo camera extrinsic parameters, this class contains essential and fundamental matrices between a select pair of cameras.
- `dv::camera::calibrations::IMUCalibration` - IMU extrinsic calibration and IMU noise parameters. This class contains extrinsic parameters - transformation between camera plane and timestamp offset (time calibration). It contains IMU measurement noise parameters: gyroscope and accelerometer measurement biases, noise densities, and random walk (white noise) parameters.

The `dv::camera::CalibrationSet` contains sets of these three types of calibration parameters, which is sufficient for storing parameters of any generic visual-inertial multi-camera rig calibration. To see the full list of exact available parameters, please see the list of public members of each of the calibration classes.

Writing a calibration file

The following code sample shows how to generate a calibration file given hardcoded calibration parameters. The generated file is going to be used as input for the next section *Reading calibration file*.

C++

Python

```

1  #include <dv-processing/camera/calibration_set.hpp>
2
3  int main() {
4      // Initialize a calibration set
5      dv::camera::CalibrationSet calibration;
6
7      // Add a camera calibration with hardcoded calibration parameters, the exact
8      // values are just for illustration.
9      calibration.addCameraCalibration(dv::camera::calibrations::CameraCalibration(
10         "DVXplorer_DXA000312", "left", true,
11         cv::Size(640, 480), cv::Point2f(320, 240), cv::Point2f(640, 640), {},
12         dv::camera::DistortionModel::NONE,
13         dv::kinematics::Transformationf(),
14         dv::camera::calibrations::CameraCalibration::Metadata()));
15
16     // Add an IMU calibration as well, the exact values are just for illustration of
17     // use here.
18     calibration.addImuCalibration(dv::camera::calibrations::IMUCalibration("DVXplorer_
19     DXA000312", 100.f, 98.1f,
20     cv::Point3f(0.f, 0.f, 0.f), cv::Point3f(0.f, 0.f, 0.f), 1.f, 1.f, 1.f, 1.f, 1.
21     f, 1.f, 3500,
22     dv::kinematics::Transformationf(),
23     dv::camera::calibrations::IMUCalibration::Metadata()));
24
25     // Just write the generated calibration set into a file.
26     calibration.writeToFile("calibration.json");
27
28     return 0;
29 }

```

```

1 import dv_processing as dv
2
3 calibration = dv.camera.CalibrationSet()
4
5 calibration.addCameraCalibration(
6     dv.camera.calibrations.CameraCalibration("DVXplorer_DXA000312", "left", True,
7     ↳ (640, 480), (320, 240), (640, 640),
8     ↳ kinematics.Transformationf(),
9     ↳ dv.camera.calibrations.CameraCalibration.
10    ↳ Metadata()))
11
12 calibration.addImuCalibration(
13     dv.camera.calibrations.IMUCalibration("DVXplorer_DXA000312", 100.0, 98.1, (0.0, 0.
14     ↳ 0, 0.0), (0.0, 0.0, 0.0), 1.0,
15     ↳ 1.0, 1.0, 1.0, 1.0, 1.0, 3500, dv.
16     ↳ kinematics.Transformationf(),
17     ↳ dv.camera.calibrations.IMUCalibration.
18     ↳ Metadata()))
19
20 calibration.writeToFile("calibration.json")

```

Reading calibration file

The following code sample shows the use of the `dv::camera::CalibrationSet` to load an existing calibration file.

Note

This sample requires an existing “calibration.json” file existing in the directory the binary is executed in. The file can be obtained by running previous sample “Writing a calibration file”.

C++

Python

```

1 #include <dv-processing/camera/calibration_set.hpp>
2
3 int main() {
4     // Initialize a calibration set
5     const auto calibrationSet = dv::camera::CalibrationSet::LoadFromFile("calibration.
6     ↳ json");
7
8     // Iterate through available camera calibrations. The designation here is an
9     ↳ internal camera abbreviation
10    // used to refer to a specific sensor in the camera rig.
11    for (const auto &[designation, calibration] : calibrationSet.
12    ↳ getCameraCalibrations()) {
13        // Print the designation and the camera name of current calibration
14        std::cout << "[" << designation << "] Found calibration for camera with name [
15        ↳ " << calibration.name << "]"
16        << std::endl;
17    }
18 }

```

(continues on next page)

(continued from previous page)

```

14      // Print the intrinsic calibration parameters for this camera: focal length,
    ↪ principal point, distortion model
15      // and parameters of the distortion model
16      std::cout << "\t Focal length: " << calibration.focalLength << std::endl;
17      std::cout << "\t Principal point: " << calibration.principalPoint <<
    ↪ std::endl;
18      std::cout << "\t Distortion model: " << calibration.
    ↪ getDistortionModelString() << std::endl;
19      std::cout << "\t Distortion parameters: "
20          << fmt::format("[{}]", fmt::join(calibration.distortion.begin(),
    ↪ calibration.distortion.end(), ", "))
21          << std::endl;
22  }
23
24  // Iterate through available IMU calibrations in the file
25  for (const auto &[designation, calibration] : calibrationSet.
    ↪ getImuCalibrations()) {
26      // Print the designation and the camera name of current calibration
27      std::cout << "[" << designation << "] Found IMU calibration for camera with
    ↪ name [" << calibration.name << "]"
28          << std::endl;
29
30      // Print some available information: accelerometer and gyroscope measurement
    ↪ limits, calibrated time offset and
31      // biases
32      std::cout << "\t Maximum acceleration: " << calibration.accMax << " [m/s^2]" <
    ↪ std::endl;
33      std::cout << "\t Maximum angular velocity: " << calibration.omegaMax << "
    ↪ [rad/s]" << std::endl;
34      std::cout << "\t Time offset: " << calibration.timeOffsetMicros << " [μs]" <<
    ↪ std::endl;
35      std::cout << "\t Accelerometer bias: " << calibration.accOffsetAvg << " [m/s^
    ↪ 2]" << std::endl;
36      std::cout << "\t Gyroscope bias: " << calibration.omegaOffsetAvg << " [rad/s]
    ↪ " << std::endl;
37
38      // Print noise density values for the IMU sensor
39      std::cout << "\t Accelerometer noise density: " << calibration.
    ↪ accNoiseDensity << " [m/s^2/sqrt(Hz)]"
40          << std::endl;
41      std::cout << "\t Gyroscope noise density: " << calibration.omegaNoiseDensity <
    ↪ " [rad/s/sqrt(Hz)]"
42          << std::endl;
43  }
44
45  return 0;
46  }

```

```

1  import dv_processing as dv
2
3  # Initialize a calibration set
4  calibration_set = dv.camera.CalibrationSet.LoadFromFile("calibration.json")

```

(continues on next page)

(continued from previous page)

```

5
6 # Iterate through available camera calibrations. The designation here is an internal
  ↳ camera abbreviation
7 # used to refer to a specific sensor in the camera rig.
8 for designation, calibration in calibration_set.getCameraCalibrations().items():
9     # Print the designation and the camera name of current calibration
10    print(f"[{designation}] Found calibration for camera with name [{calibration.name}
  ↳ ]")
11
12    # Print the intrinsic calibration parameters for this camera: focal length,
  ↳ principal point, distortion model
13    # and parameters of the distortion model
14    print(f"\t Focal length: {calibration.focalLength}")
15    print(f"\t Principal point: {calibration.principalPoint}")
16    print(f"\t Distortion model: {calibration.distortionModel}")
17    print(f"\t Distortion parameters: {calibration.distortion}")
18
19 # Iterate through available IMU calibrations in the file
20 for designation, calibration in calibration_set.getImuCalibrations().items():
21     # Print the designation and the camera name of current calibration
22     print(f"[{designation}] Found IMU calibration for camera with name [{calibration.
  ↳ name}]")
23
24     # Print some available information: accelerometer and gyroscope measurement
  ↳ limits, calibrated time offset and
25     # biases
26     print(f"\t Maximum acceleration: {calibration.accMax} [m/s^2]")
27     print(f"\t Maximum angular velocity: {calibration.omegaMax} [rad/s]")
28     print(f"\t Time offset: {calibration.timeOffsetMicros} [µs]")
29     print(f"\t Accelerometer bias: {calibration.accOffsetAvg} [m/s^2]")
30     print(f"\t Gyroscope bias: {calibration.omegaOffsetAvg} [rad/s]")
31
32     # Print noise density values for the IMU sensor
33     print(f"\t Accelerometer noise density: {calibration.accNoiseDensity} [m/s^2/
  ↳ sqrt(Hz)]")
34     print(f"\t Gyroscope noise density: {calibration.omegaNoiseDensity} [rad/s/
  ↳ sqrt(Hz)]")

```

4.1.2 Monocular camera geometry

While `dv::camera::CalibrationSet` class is designed for reading / writing the calibration parameters of multi-camera rig and `dv::camera::calibrations::CameraCalibration` stores intrinsic calibration parameters of a single pinhole camera model, these classes do not provide any operations that perform projections and (un)distortion of point coordinates. `dv::camera::CameraGeometry` class provides an efficient implementation of the mathematical operations for:

- Forward / backward projection,
- Distortion model distortion and undistortion.

Note

The operations are intended for sparse point operations, since event-camera produces sparse pixel data. For operations on full image frame prefer the use of highly optimized operations available in OpenCV.

The following code sample shows the use of `dv::camera::CameraGeometry` to back-project pixel coordinates onto 3D space, apply a 3D transformation on the points using the kinematics library and project the points back to pixel coordinates.

C++

Python

```

1  #include <dv-processing/camera/camera_geometry.hpp>
2  #include <dv-processing/core/event.hpp>
3  #include <dv-processing/data/generate.hpp>
4  #include <dv-processing/kinematics/transformation.hpp>
5  #include <dv-processing/visualization/colors.hpp>
6
7  #include <opencv2/highgui.hpp>
8
9  int main() {
10     // Use VGA resolution for this sample
11     const cv::Size resolution(640, 480);
12
13     // Initialize an ideal pinhole camera model parameters with no distortion
14     dv::camera::CameraGeometry geometry(640.f, 640.f, 320.f, 240.f, resolution);
15
16     dv::EventStore positiveEvents;
17
18     // Generate a sample set of events and filter out only positive events for this_
↪sample
19     dv::polarityFilter(dv::data::generate::dvLogoAsEvents(0, resolution),_
↪positiveEvents, true);
20
21     // Back project the events into a set of 3D points
22     const auto points = geometry.backProjectSequence<std::vector<dv::Point3f>>
↪(positiveEvents);
23
24     // Apply some 3D transformation
25     dv::kinematics::Transformationf shift(
26         0, Eigen::Vector3f(0.5f, 0.3f, 0.1f), Eigen::Quaternionf(0.24f, -0.31f, -0.
↪89f, 0.18f));
27
28     // Apply the transformation above to each of the back-projected points
29     std::vector<dv::Point3f> shiftedPoints;
30     for (const auto &point : points) {
31         shiftedPoints.push_back(shift.transformPoint<dv::Point3f>(point));
32     }
33
34     // Forward project the points with transformation
35     const auto rotatedPixels = geometry.projectSequence<std::vector<cv::Point2f>>
↪(shiftedPoints);
36
37     // Choose a color for visualization and store it in a variable that can be_

```

(continues on next page)

(continued from previous page)

```

↪efficiently assigned to a pixel intensity
38     const auto blue = dv::visualization::colors::iniBlue;
39     const cv::Vec3b color(static_cast<uint8_t>(blue[0]), static_cast<uint8_t>
↪(blue[1]), static_cast<uint8_t>(blue[2]));
40
41     // Draw input events on an image for input preview
42     cv::Mat input(resolution, CV_8UC3, dv::visualization::colors::white);
43     for (const auto &event : positiveEvents) {
44         input.at<cv::Vec3b>(event.y(), event.x()) = color;
45     }
46
47     // Draw output pixels on another image
48     cv::Mat output(resolution, CV_8UC3, dv::visualization::colors::white);
49     for (const auto &pixel : rotatedPixels) {
50         output.at<cv::Vec3b>(pixel) = color;
51     }
52
53     // Concatenate both image for a single image preview
54     cv::Mat preview;
55     cv::hconcat(input, output, preview);
56
57     // Create preview window and show the image
58     cv::namedWindow("Preview", cv::WINDOW_NORMAL);
59     cv::imshow("Preview", preview);
60     cv::waitKey();
61
62     return 0;
63 }

```

```

1  import dv_processing as dv
2  import cv2 as cv
3  import numpy as np
4
5  # Use VGA resolution for this sample
6  resolution = (640, 480)
7
8  # Initialize an ideal pinhole camera model parameters with no distortion
9  geometry = dv.camera.CameraGeometry(640, 640, 320, 240, resolution)
10
11 # Generate a sample set of events and filter out only positive events for this sample
12 positiveEvents = dv.polarityFilter(dv.data.generate.dvLogoAsEvents(0, resolution), ↪
↪True)
13
14 # Back project the events into a set of 3D points
15 points = geometry.backProjectSequence(positiveEvents)
16
17 # Apply some 3D transformation
18 shift = dv.kinematics.Transformationf(0, (0.5, 0.3, 0.1), (0.24, -0.31, -0.89, 0.18))
19
20 # Apply the transformation above to each of the back-projected points
21 shiftedPoints = []
22 for point in points:

```

(continues on next page)

(continued from previous page)

```

23     shiftedPoints.append(shift.transformPoint(point))
24
25     # Forward project the points with transformation
26     rotatedPixels = geometry.projectSequence(shiftedPoints)
27
28     # Draw input events on an image for input preview
29     input = np.ndarray((480, 640, 3), dtype=np.uint8)
30     input.fill(255)
31     for event in positiveEvents:
32         input[event.y(), event.x(), :] = dv.visualization.colors.iniBlue()
33
34     # Draw output pixels on another image
35     output = np.ndarray((480, 640, 3), dtype=np.uint8)
36     output.fill(255)
37     for pixel in rotatedPixels:
38         output[int(pixel[1]), int(pixel[0]), :] = dv.visualization.colors.iniBlue()
39
40     # Create a window and show a concatenated preview image of input events and output_
41     ↪coordinates
42     cv.namedWindow("Preview", cv.WINDOW_NORMAL)
43     cv.imshow("Preview", cv.hconcat([input, output]))
44     cv.waitKey()

```



Fig. 2: Expected result of running the sample code for camera geometry transformation.

4.1.3 Stereo camera geometry

The `dv::camera::StereoGeometry` class is intended for efficient rectification of sparse point coordinates and 3D depth estimation. The `dv::camera::StereoGeometry` is built on top of monocular `dv::io::CameraGeometry` projections and adds sparse stereo rectification that aligns coordinates for stereo disparity matching. The class also provides convenience methods to estimate 3D depth with known disparity.

For sample use of the stereo geometry class, please refer to full disparity estimation samples available [here](https://gitlab.com/inivation/dv/dv-processing/-/tree/master/samples/depth)²⁰.

²⁰ <https://gitlab.com/inivation/dv/dv-processing/-/tree/master/samples/depth>

4.2 Feature detection in events

The dv-processing library provides extensible interfaces and implementations for feature detection on event streams. Features are certain locations of interest in the data that can be detected with repeatability, such as corners. The detected features can be tracked over time in the event stream, so the library provides a reusable interface for feature detection algorithm implementations that can be used with a tracking algorithm.

4.2.1 Feature detectors

This section describes available feature detectors and their basic usage.

Feature detector using OpenCV

The most basic use of a feature detector is to use OpenCV feature detector implementation with the dv-processing feature detector wrapper on a `dv::Frame`. The feature detection wrapper base class `dv::features::FeatureDetector` was designed to wrap a feature detection algorithm and provide a few additional processing steps to handle image margins (ignore feature that are very close to edge of the image) and subsampling as a post-detection step (sampling only an optimal set of features).

Margins are set by a floating-point coefficient value, which expresses a relative margin size from the resolution of an input image, e.g. 0.05 would set the margins of 5% of the total width / height of the image.

Only a select number of features are useful for most applications. The `dv::features::FeatureDetector` simplifies feature subsampling by providing a post-processing step that sub-samples the output features. The sub-sampling can be performed by enabling one of the following post-processing options:

- **None:** Do not perform any post-processing, all features from detection will be returned.
- **TopN:** Retrieve a given number of the highest scoring features.
- **AdaptiveNMS:** Apply the AdaptiveNMS algorithm to retrieve equally spaced features in pixel space dimensions. More information on the AdaptiveNMS here: [original code](#)²¹ and [paper](#)²².

The following code sample shows how to detect the “good features to track” from OpenCV on an accumulated image from a live camera:

C++

Python

```

1  #include <dv-processing/core/frame.hpp>
2  #include <dv-processing/features/feature_detector.hpp>
3  #include <dv-processing/io/camera/discovery.hpp>
4
5  #include <opencv2/features2d.hpp>
6  #include <opencv2/highgui.hpp>
7
8  int main() {
9      using namespace std::chrono_literals;
10
11      // Open any camera
12      auto capture = dv::io::camera::open();
13
14      // Make sure it supports event stream output, throw an error otherwise

```

(continues on next page)

²¹ <https://github.com/BAILLOOL/ANMS-Codes>

²² https://www.researchgate.net/publication/323388062_Efficient_adaptive_non-maximal_suppression_algorithms_for_homogeneous_spatial_keypoint_distribution

(continued from previous page)

```

15     if (!capture->isEventStreamAvailable()) {
16         throw dv::exceptions::RuntimeError("Input camera does not provide an event_
↳ stream.");
17     }
18
19     // Initialize an accumulator with camera resolution
20     dv::Accumulator accumulator(*capture->getEventResolution());
21
22     // Initialize a preview window
23     cv::namedWindow("Preview", cv::WINDOW_NORMAL);
24
25     // Let's detect 100 features
26     const size_t numberOfFeatures = 100;
27
28     // Create an image feature detector with given resolution. By default, it uses_
↳ FAST feature detector
29     // with AdaptiveNMS post-processing.
30     auto detector = dv::features::ImageFeatureDetector(*capture->getEventResolution(),
↳ cv::GFTTDetector::create());
31
32     // Initialize a slicer
33     dv::EventStreamSlicer slicer;
34
35     // Register a callback every 33 milliseconds
36     slicer.doEveryTimeInterval(33ms, [&accumulator, &detector](const dv::EventStore &
↳ events) {
37         // Pass events into the accumulator and generate a preview frame
38         accumulator.accept(events);
39         dv::Frame frame = accumulator.generateFrame();
40
41         // Run the feature detection on the accumulated frame
42         const auto features = detector.runDetection(frame, numberOfFeatures);
43
44         // Create a colored preview image by converting from grayscale to BGR
45         cv::Mat preview;
46         cv::cvtColor(frame.image, preview, cv::COLOR_GRAY2BGR);
47
48         // Draw detected features
49         cv::drawKeypoints(preview, dv::data::fromTimedKeyPoints(features), preview);
50
51         // Show the accumulated image
52         cv::imshow("Preview", preview);
53         cv::waitKey(2);
54     });
55
56     // Run the event processing while the camera is connected
57     while (capture->isRunning()) {
58         // Receive events, check if anything was received
59         if (const auto events = capture->getNextEventBatch()) {
60             // If so, pass the events into the slicer to handle them
61             slicer.accept(*events);
62         }

```

(continues on next page)

(continued from previous page)

```

63     }
64
65     return 0;
66 }

```

```

1  from datetime import timedelta
2
3  import cv2 as cv
4  import dv_processing as dv
5
6  # Open any camera
7  capture = dv.io.camera.open()
8
9  # Make sure it supports event stream output, throw an error otherwise
10 if not capture.isEventStreamAvailable():
11     raise RuntimeError("Input camera does not provide an event stream.")
12
13 # Initialize an accumulator with some resolution
14 accumulator = dv.Accumulator(capture.getEventResolution())
15
16 # Initialize preview window
17 cv.namedWindow("Preview", cv.WINDOW_NORMAL)
18
19 # Let's detect 100 features
20 number_of_features = 100
21
22 # Create an image feature detector with given resolution. By default, it uses FAST_
23 ↪ feature detector
24 # with AdaptiveNMS post-processing.
25 detector = dv.features.ImageFeatureDetector(capture.getEventResolution())
26
27 # Initialize a slicer
28 slicer = dv.EventStreamSlicer()
29
30 # Declare the callback method for slicer
31 def slicing_callback(events: dv.EventStore):
32     # Pass events into the accumulator and generate a preview frame
33     accumulator.accept(events)
34     frame = accumulator.generateFrame()
35
36     # Run the feature detection on the accumulated frame
37     features = detector.runDetection(frame, number_of_features)
38
39     # Create a colored preview image by converting from grayscale to BGR
40     preview = cv.cvtColor(frame.image, cv.COLOR_GRAY2BGR)
41     for feature in features:
42         # Draw a rectangle marker on each feature location
43         cv.drawMarker(preview, (int(feature.pt[0]), int(feature.pt[1])),
44                       dv.visualization.colors.someNeonColor(feature.class_id), cv.
45 ↪ MARKER_SQUARE, 10, 2)

```

(continues on next page)

(continued from previous page)

```
46     # Show the accumulated image
47     cv.imshow("Preview", preview)
48     cv.waitKey(2)
49
50
51     # Register a callback every 33 milliseconds
52     slicer.doEveryTimeInterval(timedelta(milliseconds=33), slicing_callback)
53
54     # Run the event processing while the camera is connected
55     while capture.isRunning():
56         # Receive events
57         events = capture.getNextEventBatch()
58
59         # Check if anything was received
60         if events is not None:
61             # If so, pass the events into the slicer to handle them
62             slicer.accept(events)
```

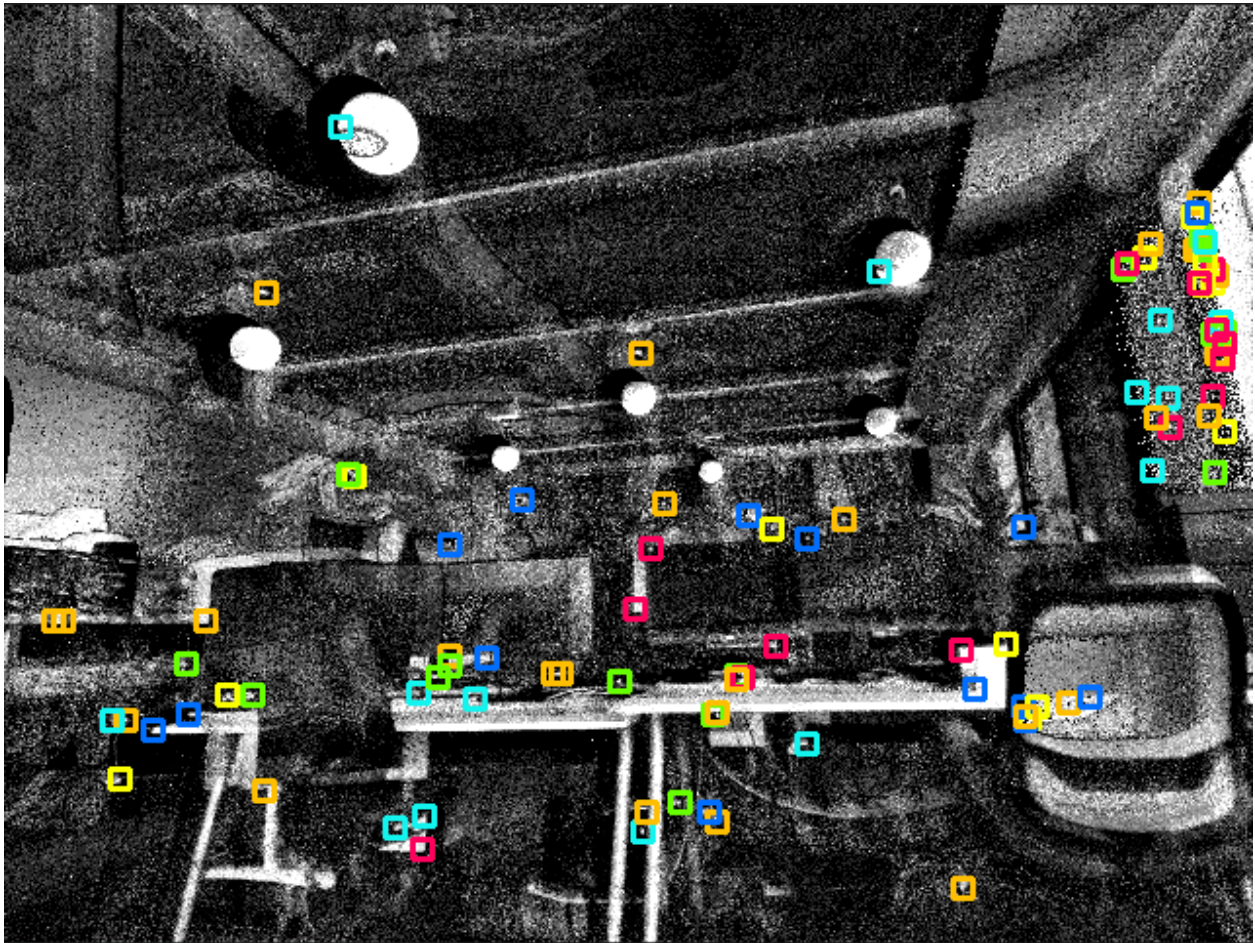


Fig. 3: Detected “good features to track” on an accumulated image.

Arc* Feature detector

The library provides an implementation of Arc* feature detector. Arc* performs corner detection on a per-event basis, so it uses `dv::EventStore` as an input. More details on this feature detection algorithm can be found in the [original paper publication](#)²³.

C++

Python

```

1  #include <dv-processing/core/frame.hpp>
2  #include <dv-processing/features/arc_corner_detector.hpp>
3  #include <dv-processing/features/feature_detector.hpp>
4  #include <dv-processing/io/camera/discovery.hpp>
5  #include <dv-processing/visualization/colors.hpp>
6
7  #include <opencv2/features2d.hpp>
8  #include <opencv2/highgui.hpp>
9
10 int main() {
11     using namespace std::chrono_literals;
12
13     // Open any camera
14     auto capture = dv::io::camera::open();
15
16     // Make sure it supports event stream output, throw an error otherwise
17     if (!capture->isEventStreamAvailable()) {
18         throw dv::exceptions::RuntimeError("Input camera does not provide an event_
19         ↪ stream.");
20     }
21
22     const auto resolution = *capture->getEventResolution();
23
24     // Initialize an accumulator with camera resolution
25     dv::Accumulator accumulator(resolution);
26
27     // Initialize a preview window
28     cv::namedWindow("Preview", cv::WINDOW_NORMAL);
29
30     // Let's detect 100 features
31     const size_t numberOfFeatures = 100;
32
33     // Create an Arc* feature detector with given resolution. Corner range is set to_
34     ↪ 5 millisecond.
35     auto detector = dv::features::FeatureDetector<dv::EventStore, _
36     ↪ dv::features::ArcCornerDetector<>> (
37         resolution, std::make_shared<dv::features::ArcCornerDetector<>>(resolution, _
38         ↪ 5000, false));
39
40     // Initialize a slicer
41     dv::EventStreamSlicer slicer;
42
43     // Register a callback every 33 milliseconds

```

(continues on next page)

²³ <https://www.research-collection.ethz.ch/bitstream/handle/20.500.11850/277131/RAL2018-camera-ready.pdf>

(continued from previous page)

```

40     slicer.doEveryTimeInterval(33ms, [&accumulator, &detector](const dv::EventStore &
↳events) {
41         // Run the feature detection on the incoming events. Features are extracted
↳on a per-event basis
42         const auto features = detector.runDetection(events, numberOfFeatures);
43
44         // Run accumulator for an image, it is going to be used only for preview
45         accumulator.accept(events);
46         const dv::Frame frame = accumulator.generateFrame();
47
48         // Create a colored preview image by converting from grayscale to BGR
49         cv::Mat preview;
50         cv::cvtColor(frame.image, preview, cv::COLOR_GRAY2BGR);
51
52         // Draw detected features
53         cv::drawKeypoints(preview, dv::data::fromTimedKeyPoints(features), preview);
54
55         // Show the accumulated image
56         cv::imshow("Preview", preview);
57         cv::waitKey(2);
58     });
59
60     // Run the event processing while the camera is connected
61     while (capture->isRunning()) {
62         // Receive events, check if anything was received
63         if (const auto events = capture->getNextEventBatch()) {
64             // If so, pass the events into the slicer to handle them
65             slicer.accept(*events);
66         }
67     }
68
69     return 0;
70 }

```

```

1  from datetime import timedelta
2
3  import cv2 as cv
4  import dv_processing as dv
5
6  # Open any camera
7  capture = dv.io.camera.open()
8
9  # Make sure it supports event stream output, throw an error otherwise
10 if not capture.isEventStreamAvailable():
11     raise RuntimeError("Input camera does not provide an event stream.")
12
13 # Initialize an accumulator with some resolution
14 accumulator = dv.Accumulator(capture.getEventResolution())
15
16 # Initialize preview window
17 cv.namedWindow("Preview", cv.WINDOW_NORMAL)
18

```

(continues on next page)

(continued from previous page)

```

19 # Let's detect 100 features
20 number_of_features = 100
21
22 # Create an Arc* feature detector with given resolution. Corner range is set to 5_
  ↳millisecond.
23 detector = dv.features.ArcEventFeatureDetector(capture.getEventResolution(), 5000, _
  ↳False)
24
25 # Initialize a slicer
26 slicer = dv.EventStreamSlicer()
27
28
29 # Declare the callback method for slicer
30 def slicing_callback(events: dv.EventStore):
31     # Run the feature detection on the incoming events
32     features = detector.runDetection(events, number_of_features)
33
34     # Pass events into the accumulator and generate a preview frame
35     accumulator.accept(events)
36     frame = accumulator.generateFrame()
37
38     # Create a colored preview image by converting from grayscale to BGR
39     preview = cv.cvtColor(frame.image, cv.COLOR_GRAY2BGR)
40     for feature in features:
41         # Draw a rectangle marker on each feature location
42         cv.drawMarker(preview, (int(feature.pt[0]), int(feature.pt[1])),
43                       dv.visualization.colors.someNeonColor(feature.class_id), cv.
  ↳MARKER_SQUARE, 10, 2)
44
45     # Show the accumulated image
46     cv.imshow("Preview", preview)
47     cv.waitKey(2)
48
49
50 # Register a callback every 33 milliseconds
51 slicer.doEveryTimeInterval(timedelta(milliseconds=33), slicing_callback)
52
53 # Run the event processing while the camera is connected
54 while capture.isRunning():
55     # Receive events
56     events = capture.getNextEventBatch()
57
58     # Check if anything was received
59     if events is not None:
60         # If so, pass the events into the slicer to handle them
61         slicer.accept(events)

```

Note

The provided implementation is not suitable for running real-time with high event rates.

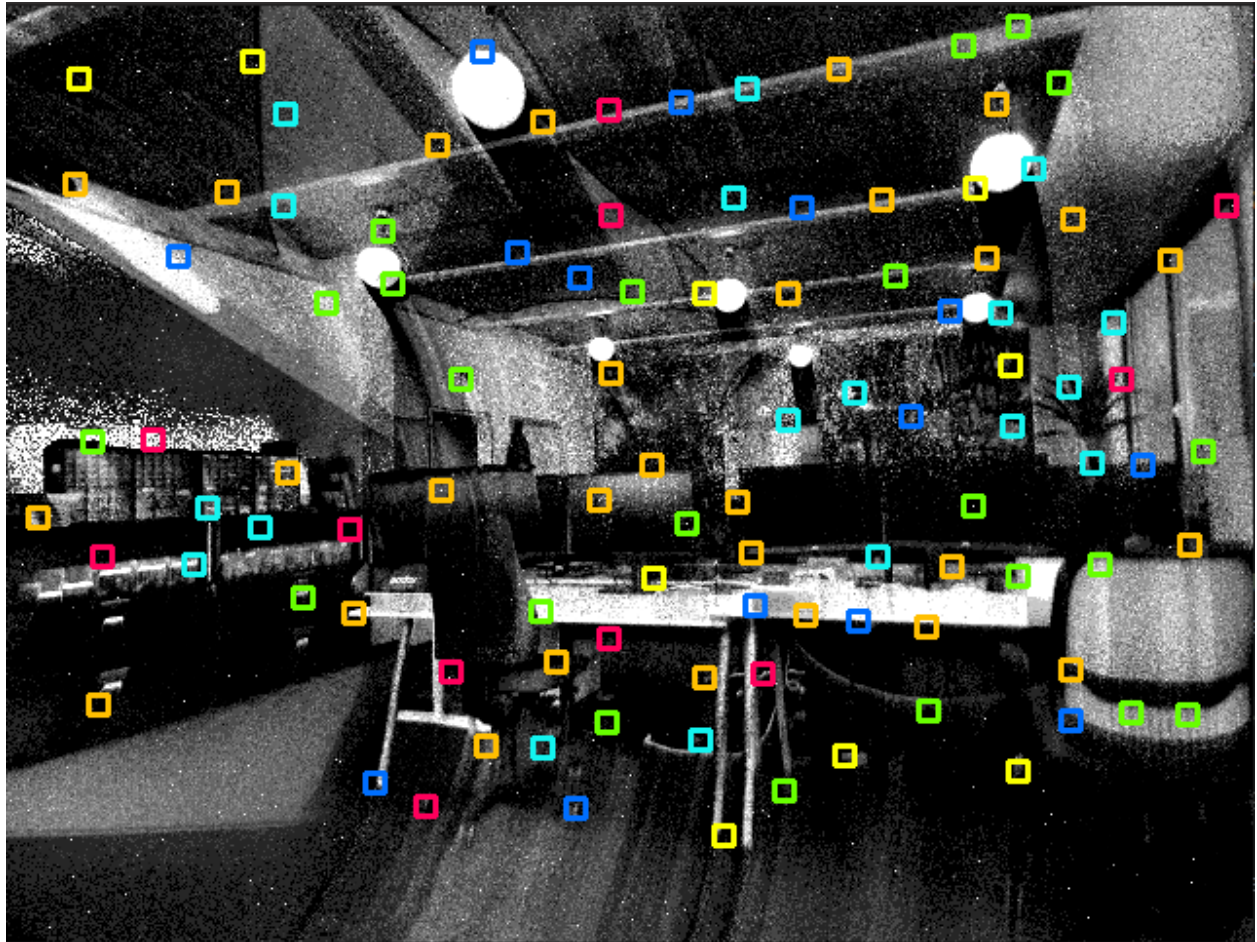


Fig. 4: Detected Arc* features displayed on an accumulated image.

Event-based blob detector

The library provides an implementation of a simple event-based blob detector. The class makes use of `cv::SimpleBlobDetector` to detect blobs on an image. The image is generated using `dv::EdgeMapAccumulator`. A default `cv::SimpleBlobDetector` is provided with reasonable values to safely detect blobs and not noise in the accumulated image. It is also possible to specify the detector that should be used to find the blobs and some specific region of interests in the image plane where the blobs should be searched. In addition to this, eventually, it is also possible to specify the down sampling factor to be applied to the image before performing the detection and also specify any additional pre-processing step that should be applied before running the actual detection step. In summary, the detection steps are as following:

1. Compute accumulated image from events
2. Apply ROI and mask to the accumulated event image
3. Down sample image (optional)
4. Apply pre-process function (optional)
5. Detect blobs
6. Rescale blobs to original resolution (if down sampling was performed)

C++

Python

```

1  #include <dv-processing/features/event_blob_detector.hpp>
2  #include <dv-processing/features/feature_detector.hpp>
3  #include <dv-processing/io/camera/discovery.hpp>
4  #include <dv-processing/visualization/colors.hpp>
5
6  #include <opencv2/features2d.hpp>
7  #include <opencv2/highgui.hpp>
8
9  int main() {
10     using namespace std::chrono_literals;
11
12     // Open any camera
13     auto capture = dv::io::camera::open();
14
15     // Make sure it supports event stream output, throw an error otherwise
16     if (!capture->isEventStreamAvailable()) {
17         throw dv::exceptions::RuntimeError("Input camera does not provide an event_
18         stream.");
19     }
20
21     const auto resolution = *capture->getEventResolution();
22
23     // Initialize an accumulator with camera resolution (for visualization only)
24     dv::Accumulator accumulator(resolution);
25
26     // Initialize a preview window
27     cv::namedWindow("Preview", cv::WINDOW_NORMAL);
28
29     // Let's detect up to 100 features
30     const size_t numberOfFeatures = 100;
31
32     // define preprocessing step (here we do nothing but show how it can be

```

(continues on next page)

(continued from previous page)

```

↪implemented)
32     std::function<void(cv::Mat &)> preprocess = [] (cv::Mat &image) {
33     };
34
35     // Define number of pyr-down to be applied to the image before detecting blobs.↪
↪If pyramidLevel is set to zero, the
36     // blobs will be detected on the original image resolution, if pyramid level is↪
↪one, the image will be down sampled
37     // by a factor of 2 (factor of 4 if pyramid level is 2 and so on...) before↪
↪performing the detection. The blobs will
38     // then be scaled back to the original resolution size.
39     const int32_t pyramidLevel = 0;
40     // Create an event-based blob detector
41     const auto blobDetector = std::make_shared<dv::features::EventBlobDetector>
↪(resolution, pyramidLevel, preprocess);
42     auto detector
↪= dv::features::EventFeatureBlobDetector(resolution,↪
↪blobDetector);
43
44     // Initialize a slicer
45     dv::EventStreamSlicer slicer;
46
47     // Register a callback every 33 milliseconds
48     slicer.doEveryTimeInterval(33ms, [&accumulator, &detector] (const dv::EventStore &
↪events) {
49         // Run the feature detection on the incoming events.
50         const auto features = detector.runDetection(events, numberOfFeatures);
51
52         // Run accumulator for an image, it is going to be used only for preview
53         accumulator.accept(events);
54         const dv::Frame frame = accumulator.generateFrame();
55
56         // Create a colored preview image by converting from grayscale to BGR
57         cv::Mat preview;
58         cv::cvtColor(frame.image, preview, cv::COLOR_GRAY2BGR);
59
60         // Draw detected features
61         cv::drawKeypoints(preview, dv::data::fromTimedKeyPoints(features), preview);
62
63         // Show the accumulated image
64         cv::imshow("Preview", preview);
65         cv::waitKey(2);
66     });
67
68     // Run the event processing while the camera is connected
69     while (capture->isRunning()) {
70         // Receive events, check if anything was received
71         if (const auto events = capture->getNextEventBatch()) {
72             // If so, pass the events into the slicer to handle them
73             slicer.accept(*events);
74         }
75     }
76

```

(continues on next page)

(continued from previous page)

```

77     return 0;
78 }

```

```

1  from datetime import timedelta
2
3  import cv2 as cv
4  import dv_processing as dv
5
6
7  def preprocess(image):
8      # add your pre processing here..
9      return image
10
11
12  # Open any camera
13  capture = dv.io.camera.open()
14
15  # Make sure it supports event stream output, throw an error otherwise
16  if not capture.isEventStreamAvailable():
17      raise RuntimeError("Input camera does not provide an event stream.")
18
19  # Initialize an accumulator with some resolution (for visualization only)
20  accumulator = dv.Accumulator(capture.getEventResolution())
21
22  # Initialize preview window
23  cv.namedWindow("Preview", cv.WINDOW_NORMAL)
24
25  # Let's detect 100 features
26  number_of_features = 100
27
28  # Define number of pyr-down to be applied to the image before detecting blobs. If
29  # ↳ pyramidLevel is set to zero, the
30  # ↳ blobs will be detected on the original image resolution, if pyramid level is one,
31  # ↳ the image will be down sampled by a
32  # ↳ factor of 2 (factor of 4 if pyramid level is 2 and so on..) before performing the
33  # ↳ detection. The blobs will then be
34  # ↳ scaled back to the original resolution size.
35  pyramidLevel = 0
36
37  # Create an event-based blob detector
38  detector = dv.features.EventFeatureBlobDetector(capture.getEventResolution(),
39  # ↳ pyramidLevel, preprocess)
40
41  # Initialize a slicer
42  slicer = dv.EventStreamSlicer()
43
44  # Declare the callback method for slicer
45  def slicing_callback(events: dv.EventStore):
46      # Run the feature detection on the incoming events
47      features = detector.runDetection(events, number_of_features)

```

(continues on next page)

(continued from previous page)

```

46     # Pass events into the accumulator and generate a preview frame
47     accumulator.accept(events)
48     frame = accumulator.generateFrame()
49
50     # Create a colored preview image by converting from grayscale to BGR
51     preview = cv.cvtColor(frame.image, cv.COLOR_GRAY2BGR)
52     for feature in features:
53         # Draw a rectangle marker on each feature location
54         cv.drawMarker(preview, (int(feature.pt[0]), int(feature.pt[1])),
55                        dv.visualization.colors.someNeonColor(feature.class_id), cv.
56 ↪MARKER_SQUARE, 10, 2)
57
58     # Show the accumulated image
59     cv.imshow("Preview", preview)
60     cv.waitKey(2)
61
62     # Register a callback every 33 milliseconds
63     slicer.doEveryTimeInterval(timedelta(milliseconds=33), slicing_callback)
64
65     # Run the event processing while the camera is connected
66     while capture.isRunning():
67         # Receive events
68         events = capture.getNextEventBatch()
69
70         # Check if anything was received
71         if events is not None:
72             # If so, pass the events into the slicer to handle them
73             slicer.accept(events)

```

4.3 Feature tracking

The dv-processing library provides a few algorithm implementations to perform visual tracking of detected features. Feature tracking was intended for use in the frontends of visual odometry pipelines. While tracking on event input is feasible, the library also provides frame-based and hybrid (which uses both events and frames) trackers that allow to build visual odometry pipelines that leverage both input modalities.

4.3.1 Frame-based tracking

Frame based feature tracking is performed by using Lucas-Kanade tracking algorithm. The following sample shows how to use the available frame based tracker with a stream of incoming frames.

The following code sample shows how to run a feature tracker on frames coming from a live camera.

Note

This sample requires a camera that is capable of producing frames, e.g. a DAVIS series camera.

C++

Python

```

1  #include <dv-processing/features/feature_tracks.hpp>
2  #include <dv-processing/features/image_feature_lk_tracker.hpp>
3  #include <dv-processing/io/camera/discovery.hpp>
4
5  #include <opencv2/highgui.hpp>
6
7  int main() {
8      // Open any camera
9      auto capture = dv::io::camera::open();
10
11      // Make sure it supports event stream output, throw an error otherwise
12      if (!capture->isFrameStreamAvailable()) {
13          throw dv::exceptions::RuntimeError("Input camera does not provide a frame_
↳stream.");
14      }
15
16      const cv::Size resolution = capture->getFrameResolution().value();
17
18      // Initialize a preview window
19      cv::namedWindow("Preview", cv::WINDOW_NORMAL);
20
21      // Instantiate a visual tracker with known resolution, all parameters kept default
22      auto tracker = dv::features::ImageFeatureLKTracker::RegularTracker(resolution);
23
24      // Create a track container instance that is used to visualize tracks on an image
25      dv::features::FeatureTracks tracks;
26
27      // Run the frame processing while the camera is connected
28      while (capture->isRunning()) {
29          // Try to receive a frame, check if anything was received
30          if (const auto frame = capture->getNextFrame()) {
31              // Pass the frame to the tracker
32              tracker->accept(*frame);
33
34              // Run tracking
35              const auto result = tracker->runTracking();
36
37              // Pass tracking result into the track container which aggregates track_
↳history
38              tracks.accept(result);
39
40              // Generate and show a preview of recent tracking history
41              cv::imshow("Preview", tracks.visualize(frame->image));
42          }
43          cv::waitKey(2);
44      }
45
46      return 0;
47  }

```

```

1  import cv2 as cv
2  import dv_processing as dv

```

(continues on next page)

(continued from previous page)

```

3
4 # Open any camera
5 capture = dv.io.camera.open()
6
7 # Make sure it supports event stream output, throw an error otherwise
8 if not capture.isFrameStreamAvailable():
9     raise RuntimeError("Input camera does not provide a frame stream.")
10
11 # Initialize preview window
12 cv.namedWindow("Preview", cv.WINDOW_NORMAL)
13
14 # Instantiate a visual tracker with known resolution, all parameters kept default
15 tracker = dv.features.ImageFeatureLKTracker.RegularTracker(capture.
16     ↪getEventResolution())
17
18 # Create a track container instance that is used to visualize tracks on an image
19 tracks = dv.features.FeatureTracks()
20
21 # Run the frame processing while the camera is connected
22 while capture.isRunning():
23     # Try to receive a frame
24     frame = capture.getNextFrame()
25
26     # Check if anything was received
27     if frame is not None:
28         # Pass the frame to the tracker
29         tracker.accept(frame)
30
31         # Run tracking
32         result = tracker.runTracking()
33
34         # Pass tracking result into the track container which aggregates track history
35         tracks.accept(result)
36
37         # Generate and show a preview of recent tracking history
38         cv.imshow("Preview", tracks.visualize(frame.image))
39
40     cv.waitKey(2)

```

4.3.2 Event-based tracking

Event-based Lucas Kanade tracker

Features can be detected and tracked on a stream of events. The `dv::features::EventFeatureLKTracker` can perform this, it accumulates a frame from events internally, runs feature detection and performs Lucas-Kanade tracking on the accumulated frames.

The following sample code shows how to use the event-only Lucas-Kanade tracker on event stream coming from a live camera.

C++

Python



Fig. 5: Tracked features on a live frame from a camera.

```

1  #include <dv-processing/features/event_feature_lk_tracker.hpp>
2  #include <dv-processing/features/feature_tracks.hpp>
3  #include <dv-processing/io/camera/discovery.hpp>
4
5  #include <opencv2/highgui.hpp>
6
7  int main() {
8      // Open any camera
9      auto capture = dv::io::camera::open();
10
11     // Make sure it supports event stream output, throw an error otherwise
12     if (!capture->isEventStreamAvailable()) {
13         throw dv::exceptions::RuntimeError("Input camera does not provide an event_
↳stream.");
14     }
15
16     const cv::Size resolution = capture->getEventResolution().value();
17
18     // Initialize a preview window
19     cv::namedWindow("Preview", cv::WINDOW_NORMAL);
20
21     // Instantiate a visual tracker with known resolution, all parameters kept default
22     auto tracker = dv::features::EventFeatureLKTracker<>::RegularTracker(resolution);
23
24     // Run tracking by accumulating frames with 100 FPS
25     tracker->setFramerate(100);
26
27     // Create a track container instance that is used to visualize tracks on an image
28     dv::features::FeatureTracks tracks;
29
30     // Run the frame processing while the camera is connected
31     while (capture->isRunning()) {
32         // Try to receive a batch of events, check if anything was received
33         if (const auto events = capture->getNextEventBatch()) {
34             // Pass the frame to the tracker
35             tracker->accept(*events);
36
37             // Run tracking
38             const auto result = tracker->runTracking();
39
40             // Since we are passing events in fine-grained batches, tracking will not_
↳execute
41             // until enough events is received, returning invalid pointer if tracking_
↳did not execute
42             if (!result) {
43                 continue;
44             }
45
46             // Pass tracking result into the track container which aggregates track_
↳history
47             tracks.accept(result);
48
49             // Generate and show a preview of recent tracking history

```

(continues on next page)

(continued from previous page)

```

50         cv::imshow("Preview", tracks.visualize(tracker->getAccumulatedFrame()));
51     }
52     cv::waitKey(2);
53 }
54
55 return 0;
56 }

```

```

1  import cv2 as cv
2  import dv_processing as dv
3
4  # Open any camera
5  capture = dv.io.camera.open()
6
7  # Make sure it supports event stream output, throw an error otherwise
8  if not capture.isEventStreamAvailable():
9      raise RuntimeError("Input camera does not provide an event stream.")
10
11 # Initialize preview window
12 cv.namedWindow("Preview", cv.WINDOW_NORMAL)
13
14 # Instantiate a visual tracker with known resolution, all parameters kept default
15 tracker = dv.features.EventFeatureLKTracker.RegularTracker(capture.
    ↳getEventResolution())
16
17 # Run tracking by accumulating frames with 100 FPS
18 tracker.setFramerate(100)
19
20 # Create a track container instance that is used to visualize tracks on an image
21 tracks = dv.features.FeatureTracks()
22
23 # Run the frame processing while the camera is connected
24 while capture.isRunning():
25     # Try to receive a batch of events
26     events = capture.getNextEventBatch()
27
28     # Check if anything was received
29     if events is not None:
30         # Pass the events to the tracker
31         tracker.accept(events)
32
33         # Run tracking
34         result = tracker.runTracking()
35
36         # Since we are passing events in fine-grained batches, tracking will not
    ↳execute
37         # until enough events is received, returning a `None` if tracking did not
    ↳execute
38         if result is None:
39             continue
40
41     # Pass tracking result into the track container which aggregates track history

```

(continues on next page)

(continued from previous page)

```
42     tracks.accept(result)
43
44     # Generate and show a preview of recent tracking history
45     cv.imshow("Preview", tracks.visualize(tracker.getAccumulatedFrame()))
46
47     cv.waitKey(2)
```

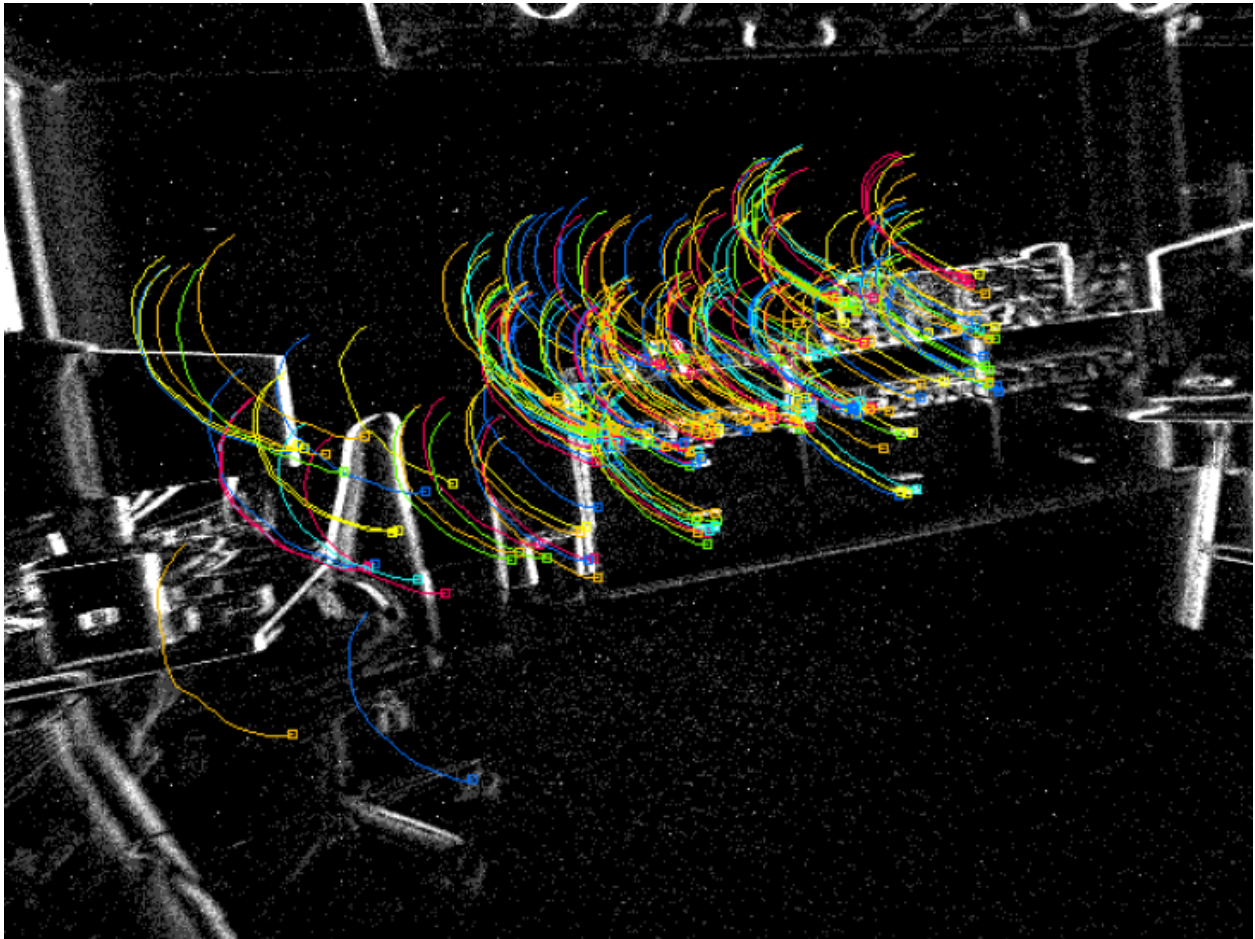


Fig. 6: Tracked features on a stream of events from a camera.

Event-based mean shift tracker

Detect and track features on a stream of events using mean shift algorithm. Although commonly used for clustering, the `dv::features::MeanShiftTracker` class provides a tracking implementation on event data based on mean shift update. The class internally detects interesting features to track from events (by default it uses `dv::features::EventBlobDetector`) and tracks them by running a mean shift update on a normalized time surface of events. The tracking is performed by following the interesting points detected on the time surface. The algorithm will shift the tracks towards the latest events, since it takes into account the intensity of the time surface when performing the track location update.

The algorithm can be summarized as follows:

1. Given a set of events, detect interesting blobs using `dv::features::EventBlobDetector`. (Note, this step happens if no track has been initialized or if redetection is enabled)

2. Compute the time surface representation of a given interval duration.
3. Given a set of input track locations, for each non-converged track retrieve the time surface of events within a configured window.
4. Calculate the mean of coordinates for the retrieved neighborhood, weighting each coordinate by the time surface intensity value.
5. Shift the initial track location by a mode, which is a vector going from the initial point to the mean coordinate multiplied by a learning rate factor.
6. If the mode of a vector is lower than a configured threshold, the track is considered to have converged into the new position, otherwise repeat from step one.

This algorithm is useful to track event blobs that could be used as point of interest in event processing algorithms.

The following code sample shows the use of our mean-shift tracker implementation to find and track events on sample data generated synthetically.

C++

Python

```

1  #include <dv-processing/core/event.hpp>
2  #include <dv-processing/data/generate.hpp>
3  #include <dv-processing/features/mean_shift_tracker.hpp>
4  #include <dv-processing/visualization/event_visualizer.hpp>
5
6  #include <opencv2/highgui.hpp>
7  #include <opencv2/imgproc.hpp>
8
9  [[nodiscard]] dv::EventStore generateEventClustersAtTime(const int64_t time, const_
↳std::vector<dv::Point2f> &clusters,
10      const uint64_t numIter, const cv::Size &resolution, const int shift = -5);
11
12 int main() {
13     using namespace std::chrono_literals;
14
15     // Use VGA resolution
16     const cv::Size resolution(640, 480);
17
18     // Initialize a slicer
19     dv::EventStreamSlicer slicer;
20
21     // Initialize a preview window
22     cv::namedWindow("Preview", cv::WINDOW_NORMAL);
23
24     // Initialize a list of clusters for synthetic data generation
25     const std::vector<dv::Point2f> clusters(
26         {dv::Point2f(550.f, 400.f), dv::Point2f(70.f, 300.f), dv::Point2f(305.f, 100.
↳f)});
27
28     // Generate some random events for a background
29     dv::EventStore events = dv::data::generate::uniformlyDistributedEvents(0,
↳resolution, 10'000);
30
31     std::vector<int64_t> timestamps = {0, 40000, 80000, 120000, 160000, 200000,

```

(continues on next page)

(continued from previous page)

```

32 ↪240000, 280000, 320000, 360000};
33     uint64_t numIter = 0;
34     for (const auto time : timestamps) {
35         auto eventCluster = generateEventClustersAtTime(time, clusters, numIter, ↪
36 ↪resolution);
37         events += eventCluster;
38         events += dv::data::generate::uniformlyDistributedEvents(time, ↪
39 ↪resolution, 10000, numIter);
40         numIter++;
41     }
42     // Bandwidth value defining the size of the search window in which updated track ↪
43 ↪location will be searched
44     const int bandwidth = 10;
45     // Time window used for the normalized time surface computation. In this case we ↪
46 ↪take the last 50ms of events and
47     // compute a normalized time surface over them
48     const dv::Duration timeWindow = 50ms;
49     // Initialize a mean shift tracker.
50     dv::features::MeanShiftTracker meanShift = ↪
51 ↪dv::features::MeanShiftTracker(resolution, bandwidth, timeWindow);
52
53     dv::visualization::EventVisualizer visualizer(resolution);
54
55     // Register a callback every 40 milliseconds
56     slicer.doEveryTimeInterval(40ms, [&](const dv::EventStore &events) {
57         meanShift.accept(events);
58         auto meanShiftTracks = meanShift.runTracking();
59
60         if (!meanShiftTracks) {
61             return;
62         }
63
64         // visualize mean shift tracks
65         auto preview = visualizer.generateImage(events);
66         auto points = dv::data::fromTimedKeyPoints(meanShiftTracks->keypoints);
67         cv::drawKeypoints(preview, points, preview, dv::visualization::colors::red);
68
69         cv::imshow("Preview", preview);
70         cv::waitKey(300);
71     });
72
73     slicer.accept(events);
74
75     return EXIT_SUCCESS;
76 }
77
78 dv::EventStore generateEventClustersAtTime(const int64_t time, const std::vector
79 ↪<dv::Point2f> &clusters,

```

(continues on next page)

(continued from previous page)

```

77     const uint64_t numIter, const cv::Size &resolution, const int shift) {
78         // Declare a region filter which we will use to filter out-of-bounds events in_
↳the next step
79         dv::EventRegionFilter filter(cv::Rect(0, 0, resolution.width, resolution.height));
80         const float offset = static_cast<float>(shift * static_cast<int>(numIter));
81         dv::EventStore eventFiltered;
82         for (const auto &cluster : clusters) {
83             const auto xShift      = cluster.x() + offset;
84             const auto yShift      = cluster.y() + offset;
85             const dv::Point2f point = dv::Point2f(xShift, yShift);
86             // Generate a batch of normally distributed events around each of the cluster_
↳centers
87             filter.accept(dv::data::generate::normallyDistributedEvents(time, point,
↳dv::Point2f(3.f, 3.f), 1'000));
88
89             // Apply region filter to the events to filter out events outside valid_
↳dimensions
90             eventFiltered += filter.generateEvents();
91         }
92
93         return eventFiltered;
94     }

```

```

1  import datetime
2
3  import dv_processing as dv
4  import cv2 as cv
5
6
7  def generate_event_clusters_at_time(time, clusters, num_iter, shift=-5):
8      # Declare a region filter which we will use to filter out-of-bounds events in the_
↳next step
9      event_filter = dv.EventRegionFilter((0, 0, resolution[0], resolution[1]))
10     event_filtered = dv.EventStore()
11     track_id = 0
12     offset = shift * num_iter
13     for cluster in clusters:
14         x_coord = cluster[0] + offset
15         y_coord = cluster[1] + offset
16
17         # Generate a batch of normally distributed events around each of the cluster_
↳centers
18         event_filter.accept(dv.data.generate.normallyDistributedEvents(time, (x_coord,
↳y_coord), (3, 3), 1000))
19
20         # Apply region filter to the events to filter out events outside valid_
↳dimensions
21         event_filtered.add(event_filter.generateEvents())
22
23         track_id += 1
24
25     return event_filtered

```

(continues on next page)

(continued from previous page)

```

26
27
28 def run_mean_shift(events):
29     mean_shift.accept(events)
30     mean_shift_tracks = mean_shift.runTracking()
31
32     preview = visualizer.generateImage(events)
33
34     # Draw markers on each of the track coordinates
35     if len(mean_shift_tracks.keypoints) > 0:
36         for index in range(len(mean_shift_tracks.keypoints)):
37             track = mean_shift_tracks.keypoints[index]
38             cv.drawMarker(preview, (int(track.pt[0]), int(track.pt[1])), dv.
↳ visualization.colors.red(), cv.MARKER_CROSS,
39                             20, 2)
40
41     # Show the preview image with detected tracks
42     cv.imshow("Preview", preview)
43     cv.waitKey(10)
44
45
46 # Use VGA resolution
47 resolution = (640, 480)
48
49 # Initialize a slicer
50 slicer = dv.EventStreamSlicer()
51
52 # Initialize a preview window
53 cv.namedWindow("Preview", cv.WINDOW_NORMAL)
54
55 # Initialize a list of clusters for synthetic data generation
56 clusters = [(550, 400), (70, 300), (305, 100)]
57
58 # Generate some random events for a background
59 events = dv.data.generate.uniformlyDistributedEvents(0, resolution, 10000)
60
61 timestamps = [0, 40000, 80000, 120000, 160000, 200000, 240000, 280000, 320000, 360000]
62
63 num_iter = 0
64 for time in timestamps:
65     event_cluster = generate_event_clusters_at_time(time, clusters, num_iter)
66     events.add(event_cluster)
67     events.add(dv.data.generate.uniformlyDistributedEvents(time, resolution, 10000,
↳ num_iter))
68     num_iter += 1
69
70 # parameter defining the spatial window [pixels] in which the new track position will
↳ be searched
71 bandwidth = 10
72
73 # window of time used to compute the time surface used for the tracking update
74 time_window = datetime.timedelta(milliseconds=50)

```

(continues on next page)

(continued from previous page)

```

75
76 # Initialize a mean shift tracker
77 mean_shift = dv.features.MeanShiftTracker(resolution, bandwidth, time_window)
78
79 visualizer = dv.visualization.EventVisualizer(resolution)
80
81 slicer.doEveryTimeInterval(datetime.timedelta(milliseconds=40), run_mean_shift)
82
83 slicer.accept(events)

```

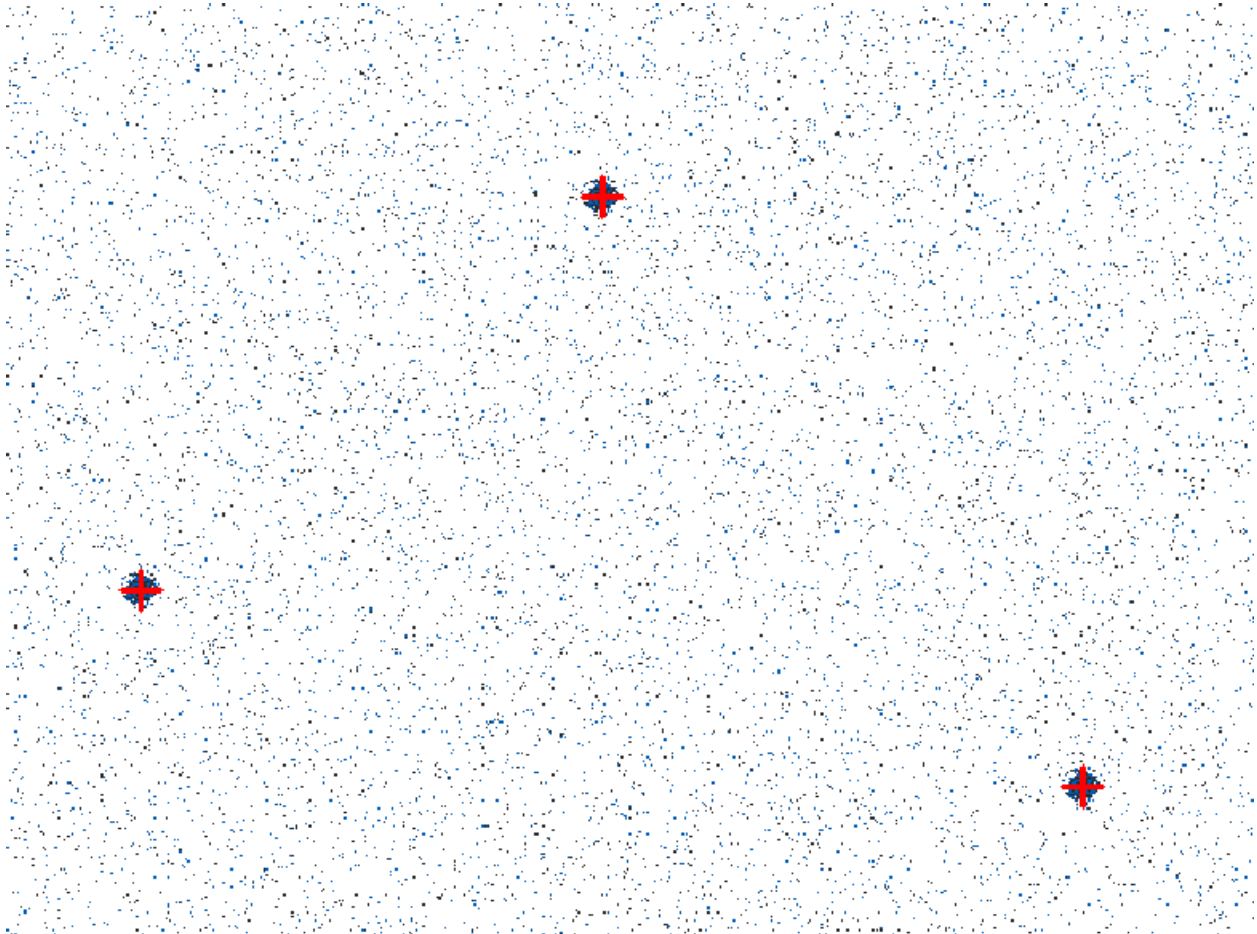


Fig. 7: Expected output of the mean-shift-tracker sample usage. Tracking eight blobs marked with red crosses.

4.3.3 Hybrid tracking

The high-framerate tracking on event stream suggests that the feature tracking on frames can be improved by tracking features between frames on intermediate accumulated frames from events. The intermediate tracking results can be used as a prior to the frame tracking algorithm. Such an approach is implemented in `dv::features::EventCombinedLKTracker`, it performs regular Lucas-Kanade tracking on frames, but also constructs intermediate accumulated frames to predict the locations of tracks in the next frame and uses this information as a prior to the Lucas-Kanade tracking algorithm.

The following sample code shows how to use the hybrid event-frame Lucas-Kanade tracker on both streams coming from

a live camera.

Note

This sample requires a camera that is capable of producing frames and events, e.g. a DAVIS series camera.

C++

Python

```

1  #include <dv-processing/features/event_combined_lk_tracker.hpp>
2  #include <dv-processing/features/feature_tracks.hpp>
3  #include <dv-processing/io/camera/discovery.hpp>
4
5  #include <opencv2/highgui.hpp>
6
7  int main() {
8      // Open any camera
9      auto capture = dv::io::camera::open();
10
11     // Make sure it supports correct stream outputs, throw an error otherwise
12     if (!capture->isEventStreamAvailable()) {
13         throw dv::exceptions::RuntimeError("Input camera does not provide an event_
14         ↪stream.");
15     }
16     if (!capture->isFrameStreamAvailable()) {
17         throw dv::exceptions::RuntimeError("Input camera does not provide a frame_
18         ↪stream.");
19     }
20
21     const cv::Size resolution = capture->getEventResolution().value();
22
23     // Initialize a preview window
24     cv::namedWindow("Preview", cv::WINDOW_NORMAL);
25
26     // Instantiate a visual tracker with known resolution, all parameters kept default
27     auto tracker = dv::features::EventCombinedLKTracker<>::RegularTracker(resolution);
28
29     // Accumulate and track on 5 intermediate accumulated frames between each actual_
30     ↪frame pair
31     tracker->setNumIntermediateFrames(5);
32
33     // Create a track container instance that is used to visualize tracks on an image
34     dv::features::FeatureTracks tracks;
35
36     // Use a queue to store incoming frames to make sure the all data has arrived_
37     ↪prior to running the tracking
38     std::queue<dv::Frame> frameQueue;
39
40     // Run the frame processing while the camera is connected
41     while (capture->isRunning()) {
42         // Try to receive a frame, check if anything was received
43         if (const auto frame = capture->getNextFrame()) {

```

(continues on next page)

(continued from previous page)

```

40     // Push the received frame into the frame queue
41     frameQueue.push(*frame);
42 }
43
44 // Try to receive a batch of events, check if anything was received
45 if (const auto events = capture->getNextEventBatch()) {
46     // Pass the frame to the tracker
47     tracker->accept(*events);
48
49     // Check if we have ready frames and if enough events have arrived already
50     if (frameQueue.empty() || frameQueue.front().timestamp > events->
↳getHighestTime()) {
51         continue;
52     }
53
54     // Take the last frame from the queue
55     const auto frame = frameQueue.front();
56
57     // Pass it to the tracker as well
58     tracker->accept(frame);
59
60     // Remove the last used frame from the queue
61     frameQueue.pop();
62
63     // Run tracking
64     const auto result = tracker->runTracking();
65
66     // Validate that the tracking was successful
67     if (!result) {
68         continue;
69     }
70
71     // Pass tracking result into the track container which aggregates track_
↳history
72     tracks.accept(result);
73
74     // Generate and show a preview of recent tracking history on both_
↳accumulated frames and the frame image
75     // Take the set of intermediate accumulated frames from the tracker
76     const auto accumulatedFrames = tracker->getAccumulatedFrames();
77     if (!accumulatedFrames.empty()) {
78         cv::Mat preview;
79         // Draw visualization on both image and concatenate them horizontally
80         cv::hconcat(
81             tracks.visualize(accumulatedFrames.back().pyramid.front()),
↳tracks.visualize(frame.image), preview);
82         // Show the final preview image
83         cv::imshow("Preview", preview);
84     }
85 }
86
87 cv::waitKey(2);

```

(continues on next page)

(continued from previous page)

```

88     }
89
90     return 0;
91 }

```

```

1  import cv2 as cv
2  import dv_processing as dv
3
4  # Open any camera
5  capture = dv.io.camera.open()
6
7  # Make sure it supports correct stream outputs, throw an error otherwise
8  if not capture.isEventStreamAvailable():
9      raise RuntimeError("Input camera does not provide an event stream.")
10 if not capture.isEventStreamAvailable():
11     raise RuntimeError("Input camera does not provide a frame stream.")
12
13 # Initialize preview window
14 cv.namedWindow("Preview", cv.WINDOW_NORMAL)
15
16 # Instantiate a visual tracker with known resolution, all parameters kept default
17 tracker = dv.features.EventCombinedLKTracker.RegularTracker(capture.
↳getEventResolution())
18
19 # Accumulate and track on 5 intermediate accumulated frames between each actual frame.
↳pair
20 tracker.setNumIntermediateFrames(5)
21
22 # Create a track container instance that is used to visualize tracks on an image
23 tracks = dv.features.FeatureTracks()
24
25 # Use a list to store incoming frames to make sure the all data has arrived prior to.
↳running the tracking
26 frame_queue = []
27
28 # Run the frame processing while the camera is connected
29 while capture.isRunning():
30     # Try to receive a frame
31     frame = capture.getNextFrame()
32
33     # Check if anything was received
34     if frame is not None:
35         frame_queue.append(frame)
36
37     # Try to receive a batch of events
38     events = capture.getNextEventBatch()
39
40     # Check if anything was received
41     if events is not None:
42         # Pass the events to the tracker
43         tracker.accept(events)
44

```

(continues on next page)

(continued from previous page)

```

45     # Check if we have ready frames and if enough events have arrived already
46     if len(frame_queue) == 0 or frame_queue[0].timestamp > events.
↳getHighestTime():
47         continue
48
49     # Take the last frame from the queue and remove it
50     frame = frame_queue.pop(0)
51
52     # Pass it to the tracker as well
53     tracker.accept(frame)
54
55     # Run tracking
56     result = tracker.runTracking()
57
58     # Validate that the tracking was successful
59     if result is None:
60         continue
61
62     # Pass tracking result into the track container which aggregates track history
63     tracks.accept(result)
64
65     # Generate and show a preview of recent tracking history on both accumulated_
↳frames and the frame image
66     # Take the set of intermediate accumulated frames from the tracker
67     accumulated_frames = tracker.getAccumulatedFrames()
68     if len(accumulated_frames) > 0:
69         # Draw visualization on both image and concatenate them horizontally
70         preview = cv.hconcat([tracks.visualize(accumulated_frames[-1].pyramid[0]),
↳ tracks.visualize(frame.image)])
71
72     # Show the final preview image
73     cv.imshow("Preview", preview)
74
75     cv.waitKey(2)

```

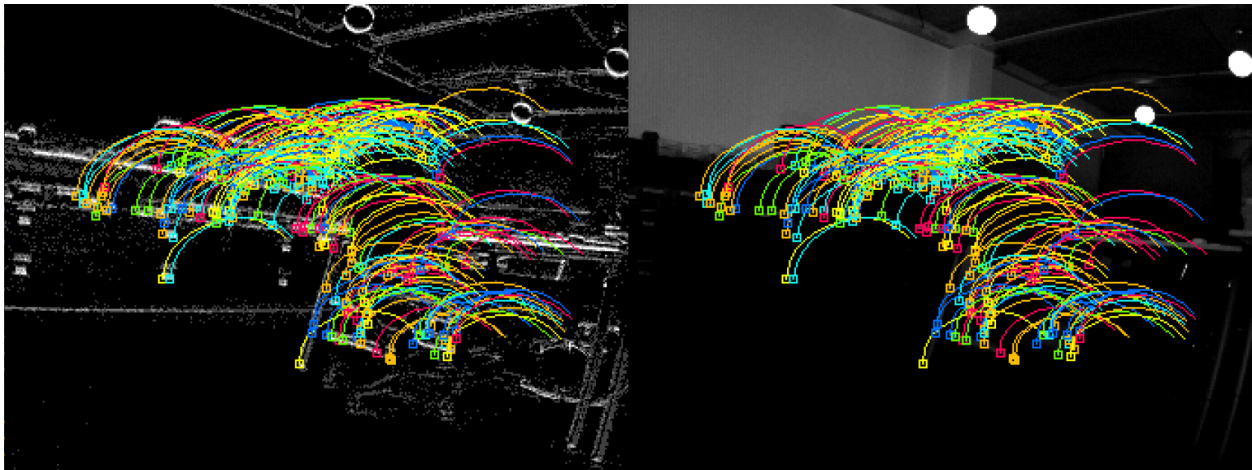


Fig. 8: Tracked features on frame and event streams from a camera.

4.4 Kinematics primitives

The dv-processing library provides a minimal implementation of kinematic transformation primitives that are useful for geometric computer vision algorithms. This includes transformations of 3D points using transformation matrices and handling rigid-body motion trajectories over time. These basic primitives can be used to implement motion compensation algorithm for event data, which can be used to reduce or eliminate motion blur in the event stream. The underlying implementation uses mathematical operations from Eigen library, so the mathematical operations are expected to be highly efficient.

4.4.1 Transformation

A transformation in dv-processing describes an object's orientation and position in 3D space at a certain point in time. Transformation contains a timestamp, rotational and translational transformation expressed as a 4x4 homogenous transformation matrix T , like this:

$$T = \begin{bmatrix} r_0 & r_1 & r_2 & t_0 \\ r_3 & r_4 & r_5 & t_1 \\ r_6 & r_7 & r_8 & t_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} r & t \\ 0 & 1 \end{bmatrix}$$

Here:

- $r, r_{0..9}$ - is a rotation matrix (and it's coefficients) that describes an object's rotation.
- $t, t_{0..2}$ - is a vector describing translational vector, which is an object's displacement.

A transformation matrix together with a timestamp describes complete attitude with 6 degrees of freedom. This transformation can be applied to other transformations as well as 3D points to obtain a new relative position with the applied transformation. The transformation is implemented in the `dv::kinematics::Transformation` class, which is a templated class. The template parameter sets the underlying matrix scalar data type for the 4x4 matrix, which is either `float` or `double`. To simplify the use case, two predefined aliases are defined: `dv::kinematics::Transformationf` and `dv::kinematics::Transformationd` - they differ in the underlying scalar data type:

- `Transformationf` uses 32-bit single precision floating point values,
- `Transformationd` uses 64-bit double precision floating point values.

The library usually prefers the use of single precision floating point scalar type, since the representation is accurate enough for sub-millimeter accuracy with lower memory footprint.

The following sample code shows how to initialize a transformation and apply it to a 3D point.

C++

Python

```

1  #include <dv-processing/kinematics/transformation.hpp>
2
3  #include <iostream>
4
5  int main() {
6      Eigen::Matrix4f matrix;
7
8      // Mirror rotation matrix with 0.5 translational offsets on all axes. The
↪ rotation matrix should flip
9      // x and z axes of the input.
10     matrix << -1.f, 0.f, 0.f, 0.5f, 0.f, 1.f, 0.f, 0.5f, 0.f, 0.f, -1.f, 0.5f, 0.f, 0.
↪ f, 0.f, 1.f;

```

(continues on next page)

(continued from previous page)

```

11
12 // Initialize the transformation with the above matrix. The timestamp can be_
↳ ignored for this sample, so its set
13 // to zero.
14 const dv::kinematics::Transformationf transformation(0, matrix);
15
16 // Let's take a sample point with offsets of 1 on all axes.
17 const Eigen::Vector3f point(1.f, 1.f, 1.f);
18
19 // Apply this transformation to the above point. This should invert x and z axes_
↳ and add 0.5 to all values.
20 const Eigen::Vector3f transformed = transformation.transformPoint(point);
21
22 // Print the resulting output.
23 std::cout << "Transformed from [" << point.transpose() << "]" to [" << transformed.
↳ transpose() << "]" << std::endl;
24
25 return 0;
26 }

```

```

1 import dv_processing as dv
2 import numpy as np
3
4 # Mirror rotation matrix with 0.5 translational offsets on all axes. The rotation_
↳ matrix should flip
5 # x and z axes of the input.
6 matrix = np.array([[ -1.0, 0.0, 0.0, 0.5], [0.0, 1.0, 0.0, 0.5], [0.0, 0.0, -1.0, 0.5],
↳ [0.0, 0.0, 0.0, 1.]])
7
8 # Initialize the transformation with the above matrix. The timestamp can be ignored_
↳ for this sample, so its set
9 # to zero.
10 transformation = dv.kinematics.Transformationf(0, matrix)
11
12 # Let's take a sample point with offsets of 1 on all axes.
13 point = np.array([1.0, 1.0, 1.0])
14
15 # Apply this transformation to the above point. This should invert x and z axes and_
↳ add 0.5 to all values.
16 transformed = transformation.transformPoint(point)
17
18 # Print the resulting output.
19 print(f"Transformed from {point} to {transformed}")

```

4.4.2 Linear transformer

A set of transformations that are monotonically increasing in time can be formed into a motion trajectory. Linear transformer can be used to store a set of transformation representing a single objects trajectory and extract transformations at specified points in time, which are calculated using linear interpolation between the nearest available transformations.

The following sample code shows how to use the `dv::kinematics::LinearTransformerf` class to interpolate intermediate transformations in time:

C++

Python

```
1 #include <dv-processing/kinematics/linear_transformer.hpp>
2 #include <dv-processing/kinematics/transformation.hpp>
3
4 #include <iostream>
5
6 int main() {
7     // Declare linear transformer with capacity of 100 transformations. Internally it
8     ↪ uses a bounded FIFO queue
9     // to manage the transformations.
10    dv::kinematics::LinearTransformerf transformer(100);
11
12    // Push first transformation which is an identity matrix, so it starts with no
13    ↪ rotation at zero coordinates
14    transformer.pushTransformation(
15        dv::kinematics::Transformationf(1000000, Eigen::Vector3f(0.f, 0.f, 0.f),
16        ↪ Eigen::Quaternionf::Identity()));
17
18    // Add a second transformation with no rotation as well, but with different
19    ↪ translational coordinates
20    transformer.pushTransformation(
21        dv::kinematics::Transformationf(2000000, Eigen::Vector3f(1.f, 2.f, 3.f),
22        ↪ Eigen::Quaternionf::Identity()));
23
24    // Interpolate transformation at a midpoint (time-wise), this should device the
25    ↪ translational coordinates
26    // by a factor of 2.0
27    const auto midpoint = transformer.getTransformAt(1500000);
28
29    // Print the resulting output.
30    std::cout << "Interpolated position at [" << midpoint->getTimestamp() << "]: ["
31        << midpoint->getTranslation().transpose() << "]" << std::endl;
32
33    return 0;
34 }
```

```
1 import dv_processing as dv
2 import numpy as np
3
4 # Declare linear transformer with capacity of 100 transformations. Internally it uses
5 ↪ a bounded FIFO queue
6 # to manage the transformations.
7 transformer = dv.kinematics.LinearTransformerf(100)
8
9 # Push first transformation which is an identity matrix, so it starts with no
10 ↪ rotation at zero coordinates
11 transformer.pushTransformation(dv.kinematics.Transformationf(1000000, np.array([0.0,
12 ↪ 0.0, 0.0]), (1.0, 0.0, 0.0, 0.0)))
13
14 # Add a second transformation with no rotation as well, but with different
15 ↪ translational coordinates
```

(continues on next page)

(continued from previous page)

```

12 transformer.pushTransformation(dv.kinematics.Transformationf(2000000, np.array([1.0, ↵
    ↵2.0, 3.0]), (1.0, 0.0, 0.0, 0.0)))
13
14 # Interpolate transformation at a midpoint (time-wise), this should device the ↵
    ↵translational coordinates
15 # by a factor of 2.0
16 midpoint = transformer.getTransformAt(1500000)
17
18 # Print the resulting output.
19 print(f"Interpolated position at [{midpoint.getTimestamp()}]: {midpoint.
    ↵getTranslation()}")

```

4.5 Mean-shift clustering

Mean-shift clustering algorithm is used to find high density clusters of events (or point) within a set of events. The dv-processing library provides an implementation of such an algorithm which is efficiently implemented for incoming streams of events from event cameras.

The algorithm can be summarized as follows:

1. Given a set of coordinates, for each coordinate retrieve a neighborhood of events within a configured window.
2. Calculate the mean of coordinates for the retrieved neighborhood.
3. Shift the initial coordinates by a mode, which is a vector going from the initial point to the mean coordinate multiplied by a learning rate factor.
4. If the mode of a vector is lower than a configured threshold, the coordinate is considered to have converged into a cluster center, otherwise repeat from step one.

This algorithm is useful to find event clusters that could be used as point of interest in event processing algorithms.

4.5.1 Sample usage

The following code sample shows the use of mean-shift clustering implementation to find and track clusters of events on simulated events.

C++

Python

```

1  #include <dv-processing/cluster/mean_shift/event_store_adaptor.hpp>
2  #include <dv-processing/core/event.hpp>
3  #include <dv-processing/data/generate.hpp>
4  #include <dv-processing/visualization/event_visualizer.hpp>
5
6  #include <opencv2/highgui.hpp>
7
8  int main() {
9      // Use VGA resolution
10     const cv::Size resolution(640, 480);
11
12     // Initialize a list of clusters for synthetic data generation
13     const std::vector<dv::Point2f> clusters(
14         {dv::Point2f(100.f, 100.f), dv::Point2f(462.f, 25.f), dv::Point2f(105.f, 340.

```

(continues on next page)

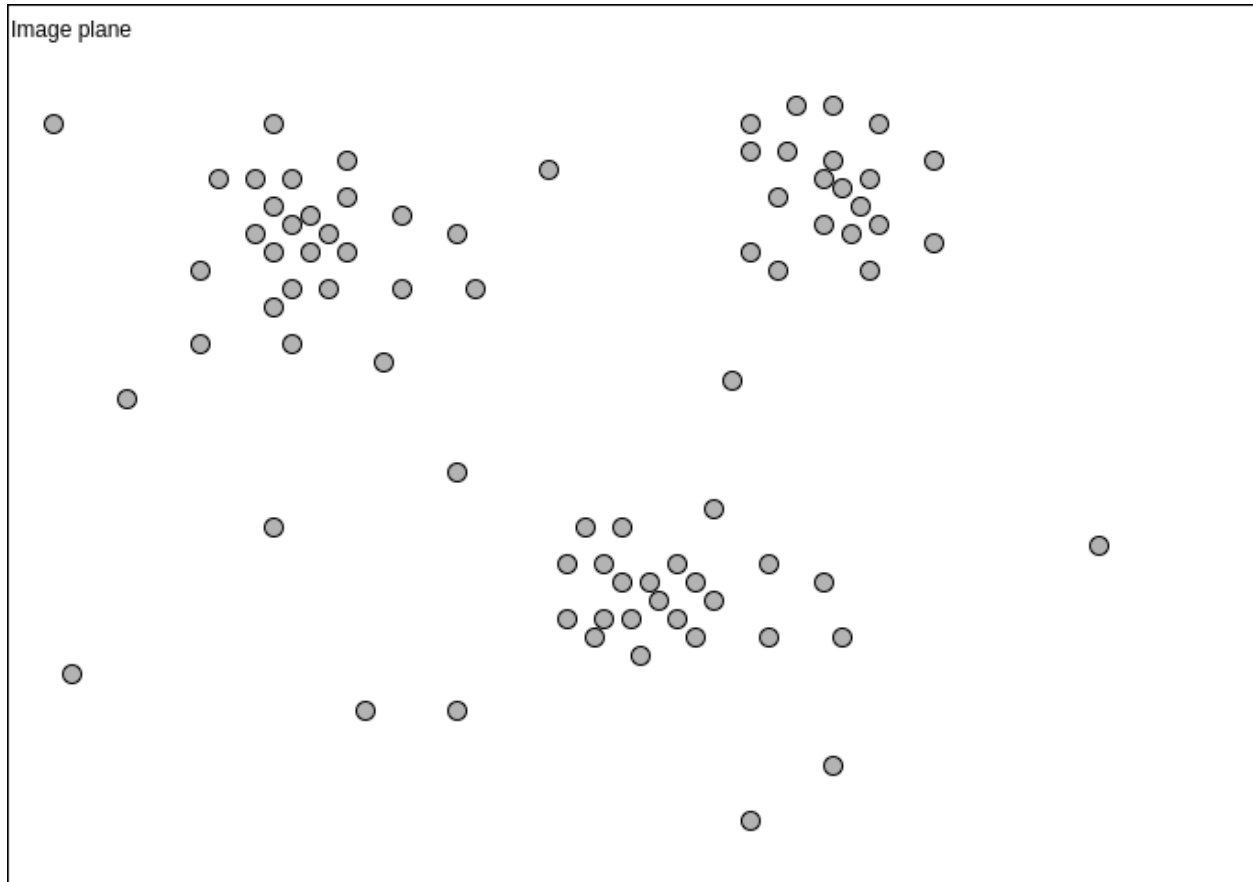


Fig. 9: An approximate visualization of a mean-shift clustering algorithm to find centers of coordinate clusters in a 2D plane.

(continued from previous page)

```

15  ↪f), dv::Point2f(540.f, 420.f) });
16      // Generate some random events for a background
17      dv::EventStore events = dv::data::generate::uniformlyDistributedEvents(0, ↪
18  ↪resolution, 10'000);
19      // Declare a region filter which we will use to filter out-of-bounds events in ↪
20  ↪the next step
21      dv::EventRegionFilter filter(cv::Rect(0, 0, resolution.width, resolution.height));
22      for (const auto &cluster : clusters) {
23          // Generate a batch of normally distributed events around each of the cluster ↪
24  ↪centers
25          filter.accept(dv::data::generate::normallyDistributedEvents(0, cluster, ↪
26  ↪dv::Point2f(15.f, 15.f), 5'000));
27          // Apply region filter to the events to filter out events outside valid ↪
28  ↪dimensions
29          events += filter.generateEvents();
30      }
31      // Initialize mean shift clustering algorithm, with initial parameters of:
32      // bandwidth = 100, this is pixel search radius around a point
33      // conv = 0.01, the search converges when the magnitude of mean-shift vector is ↪
34  ↪below this value
35      // maxIter = 10000, maximum number of mean-shift update iterations
36      // numStartingPoints = 100, number of randomly selected starting points
37      dv::cluster::mean_shift::MeanShiftEventStoreAdaptor meanShift(events, 100.0f, 0. ↪
38  ↪01f, 10000, 100);
39      // Perform the mean-shift, the algorithm returns a tuple of center coordinates, ↪
40  ↪labels, count of event in
41      // the cluster, and variances of the cluster
42      auto [centers, labels, counts, variances] = meanShift.fit();
43      // Let's assign the cluster size to the response value of the center keypoint
44      for (int i = 0; i < centers.size(); i++) {
45          centers[i].response = static_cast<float>(counts[i]);
46      }
47      // Sort the estimated centers by the number of events in cluster, the values are ↪
48  ↪sorted in descending order
49      std::sort(centers.begin(), centers.end(), [](const auto &a, const auto &b) {
50          return a.response > b.response;
51      });
52      // Choose top four center with most events; these centers should be close to ↪
53  ↪initial hardcoded cluster centers
54      if (centers.size() > 4) {
55          centers.resize(4);
56      }

```

(continues on next page)

(continued from previous page)

```

56 // Use event visualizer to generate a preview image
57 dv::visualization::EventVisualizer visualizer(resolution);
58 auto preview = visualizer.generateImage(events);
59
60 // Draw markers on each of the center coordinates
61 for (const auto &center : centers) {
62     cv::drawMarker(preview, cv::Point2f(center.pt.x(), center.pt.y()), ↵
↵ dv::visualization::colors::red);
63 }
64
65 // Show the preview image with detected cluster centers
66 cv::namedWindow("Preview", cv::WINDOW_NORMAL);
67 cv::imshow("Preview", preview);
68 cv::waitKey();
69
70 return 0;
71 }

```

```

1  import dv_processing as dv
2  import cv2 as cv
3
4  # Use VGA resolution
5  resolution = (640, 480)
6
7  # Initialize a list of clusters for synthetic data generation
8  clusters = [(100, 100), (462, 25), (105, 340), (540, 420)]
9
10 # Generate some random events for a background
11 events = dv.data.generate.uniformlyDistributedEvents(0, resolution, 10000)
12
13 # Declare a region filter which we will use to filter out-of-bounds events in the ↵
↵ next step
14 filter = dv.EventRegionFilter((0, 0, resolution[0], resolution[1]))
15
16 for cluster in clusters:
17     # Generate a batch of normally distributed events around each of the cluster ↵
↵ centers
18     filter.accept(dv.data.generate.normallyDistributedEvents(0, cluster, (15, 15), ↵
↵ 5000))
19
20     # Apply region filter to the events to filter out events outside valid dimensions
21     events.add(filter.generateEvents())
22
23 # Initialize mean shift clustering algorithm, with initial parameters of:
24 # bandwidth = 100, this is pixel search radius around a point
25 # conv = 0.01, the search converges when the magnitude of mean-shift vector is below ↵
↵ this value
26 # maxIter = 10000, maximum number of mean-shift update iterations
27 # numStartingPoints = 100, number of randomly selected starting points
28 mean_shift = dv.cluster.mean_shift.MeanShiftEventStoreAdaptor(events, 100, 0.01, ↵
↵ 10000, 100)
29

```

(continues on next page)

(continued from previous page)

```
30 # Perform the mean-shift, the algorithm returns a tuple of center coordinates, labels,  
    ↳ count of event in the  
31 # cluster, and variances of the cluster  
32 centers, labels, counts, variances = mean_shift.fit()  
33  
34 # Let's assign the cluster size to the response value of the center keypoint  
35 for i in range(len(centers)):  
36     centers[i].response = counts[i]  
37  
38 # Sort the estimated centers by the number of events in cluster, the values are_  
    ↳ sorted in descending order  
39 centers.sort(key=lambda a: a.response, reverse=True)  
40  
41 # Choose top four center with most events; these centers should be close to initial_  
    ↳ hardcoded cluster centers  
42 if len(centers) > 4:  
43     centers = centers[:4]  
44  
45 # Use event visualizer to generate a preview image  
46 visualizer = dv.visualization.EventVisualizer(resolution)  
47 preview = visualizer.generateImage(events)  
48  
49 # Draw markers on each of the center coordinates  
50 for center in centers:  
51     cv.drawMarker(preview, (int(center.pt[0]), int(center.pt[1])), dv.visualization.  
    ↳ colors.red())  
52  
53 # Show the preview image with detected cluster centers  
54 cv.namedWindow("Preview", cv.WINDOW_NORMAL)  
55 cv.imshow("Preview", preview)  
56 cv.waitKey()
```

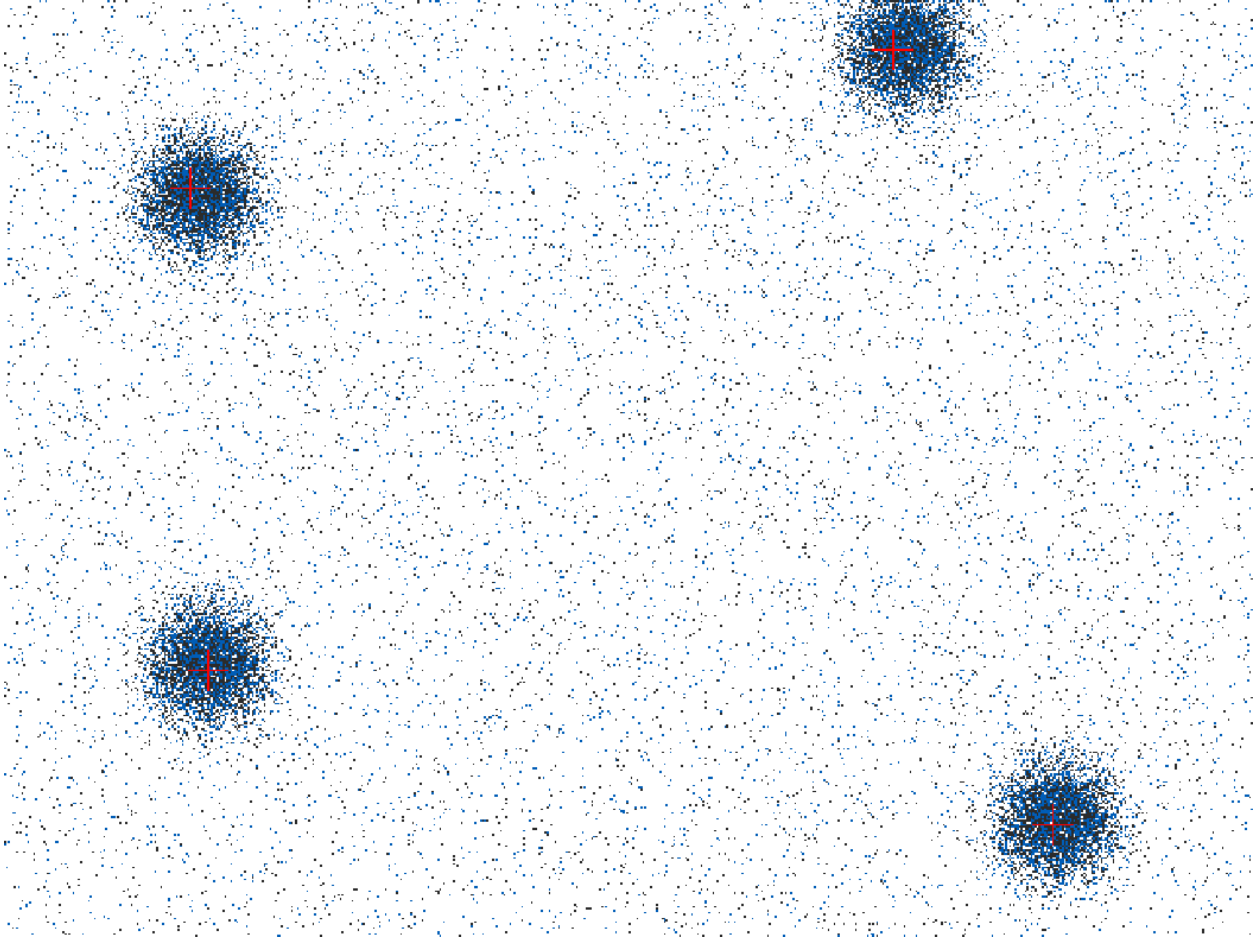


Fig. 10: Expected output of the mean-shift sample usage, four cluster centers marked with red crosses.

ADVANCED APPLICATIONS

This section describes the use of available library features and algorithms to build more complex applications and algorithms for: depth estimation, motion compensation and contrast maximization.

5.1 Depth estimation

Depth estimation with event cameras is possible by applying the same approach of disparity calculation on a calibrated stereo camera rig. The straightforward approach is to accumulate frames from events on both cameras and use the same disparity estimation algorithm. This approach might have some limitations, since accumulating events might result in suboptimal results due to low texture available in an accumulated frame.

The dv-processing library provides the `dv::camera::StereoGeometry` and a few disparity estimation algorithms that, in combination, can be used to build a depth estimation pipeline.

5.1.1 Semi-dense stereo block matching

Dense block matching here refers to the most straightforward approach: accumulating full frames and running a conventional disparity estimation on top to estimate depth. Since the accumulated frames only contain limited texture due to pixels reacting to brightness changes - this approach is referred to as semi-dense. The `SemiDenseStereoMatcher` class wraps the disparity estimation part, where estimated disparity can be used to calculate depth with `dv::camera::StereoGeometry`.

Following sample code show the use of `SemiDenseStereoMatcher` with `dv::camera::StereoGeometry` to run a real-time depth estimation pipeline on a calibration stereo camera.

C++

Python

```

1  #include <dv-processing/camera/calibration_set.hpp>
2  #include <dv-processing/core/stereo_event_stream_slicer.hpp>
3  #include <dv-processing/depth/semi_dense_stereo_matcher.hpp>
4  #include <dv-processing/io/camera/discovery.hpp>
5  #include <dv-processing/noise/background_activity_noise_filter.hpp>
6
7  #include <opencv2/highgui.hpp>
8
9  int main() {
10     using namespace std::chrono_literals;
11
12     // Path to a stereo calibration file, replace with a file path on your local file_
↪system
13     const std::string calibrationFilePath = "path/to/calibration.json";

```

(continues on next page)

(continued from previous page)

```

14
15 // Load the calibration file
16 auto calibration = dv::camera::CalibrationSet::LoadFromFile(calibrationFilePath);
17
18 // It is expected that calibration file will have "C0" as the leftEventBuffer_
↪ camera
19 auto leftCameraCalib = calibration.getCameraCalibration("C0").value();
20
21 // The second camera is assumed to be rightEventBuffer-side camera
22 auto rightCameraCalib = calibration.getCameraCalibration("C1").value();
23
24 // Open the stereo camera with camera names from calibration
25 auto leftCamera = dv::io::camera::openSync(leftCameraCalib.name);
26 auto rightCamera = dv::io::camera::openSync(rightCameraCalib.name);
27
28 dv::io::camera::synchronizeAnyTwo(leftCamera, rightCamera);
29
30 // Make sure both cameras support event stream output, throw an error otherwise
31 if (!leftCamera->isEventStreamAvailable() || !rightCamera->
↪ isEventStreamAvailable()) {
32     throw dv::exceptions::RuntimeError("Input camera does not provide an event_
↪ stream.");
33 }
34
35 // Initialize a stereo block matcher with a stereo geometry from calibration and_
↪ the preconfigured SGBM instance
36 dv::SemiDenseStereoMatcher blockMatcher(
37     std::make_unique<dv::camera::StereoGeometry>(leftCameraCalib,
↪ rightCameraCalib));
38
39 // Initialization of a stereo event slicer
40 dv::StereoEventStreamSlicer slicer;
41
42 // Initialize a window to show previews of the output
43 cv::namedWindow("Preview", cv::WINDOW_NORMAL);
44
45 // Local event buffers to implement overlapping window of events for accumulation
46 dv::EventStore leftEventBuffer, rightEventBuffer;
47
48 // Use one third of the resolution as count of events per accumulated frame
49 const size_t eventCount = static_cast<size_t>(leftCameraCalib.resolution.area()) /
↪ 3;
50
51 // Register a callback to be done at 30Hz
52 slicer.doEveryTimeInterval(33ms, [&blockMatcher, &leftEventBuffer, &
↪ rightEventBuffer, eventCount](
53     const auto &leftEvents, const auto &
↪ rightEvents) {
54     // Push input events into the local buffers
55     leftEventBuffer.add(leftEvents);
56     rightEventBuffer.add(rightEvents);
57

```

(continues on next page)

(continued from previous page)

```

58     // If the number of events is above the count, just keep the latest events
59     if (leftEventBuffer.size() > eventCount) {
60         leftEventBuffer = leftEventBuffer.sliceBack(eventCount);
61     }
62     if (rightEventBuffer.size() > eventCount) {
63         rightEventBuffer = rightEventBuffer.sliceBack(eventCount);
64     }
65
66     // Pass these events into block matcher and estimate disparity, the matcher
↳will accumulate frames
67     // internally. The disparity output is 16-bit integer, that has sub-pixel
↳precision.
68     const auto disparity = blockMatcher.computeDisparity(leftEventBuffer,
↳rightEventBuffer);
69
70     // Convert disparity into 8-bit integers with scaling and normalize the
↳output for a nice preview.
71     // This loses the actual numeric value of the disparity, but it's a nice way
↳to visualize the disparity.
72     cv::Mat disparityU8, disparityColored;
73     disparity.convertTo(disparityU8, CV_8UC1, 1.0 / 16.0);
74     cv::normalize(disparityU8, disparityU8, 0, 255, cv::NORM_MINMAX);
75
76     // Convert the accumulated frames into colored images for preview.
77     std::vector<cv::Mat> images(3);
78     cv::cvtColor(blockMatcher.getLeftFrame().image, images[0], cv::COLOR_
↳GRAY2BGR);
79     cv::cvtColor(blockMatcher.getRightFrame().image, images[1], cv::COLOR_
↳GRAY2BGR);
80
81     // Apply color-mapping to the disparity image, this will encode depth with
↳color: red - close; blue - far.
82     cv::applyColorMap(disparityU8, images[2], cv::COLORMAP_JET);
83
84     // Concatenate images and show them in a window
85     cv::Mat preview;
86     cv::hconcat(images, preview);
87     cv::imshow("Preview", preview);
88     });
89
90     // Buffer input events in these variables to synchronize inputs
91     std::optional<dv::EventStore> leftEvents = std::nullopt;
92     std::optional<dv::EventStore> rightEvents = std::nullopt;
93
94     // Run the processing loop while both cameras are connected
95     while (leftCamera->isRunning() && rightCamera->isRunning()) {
96         // Read events from respective left / right cameras
97         if (!leftEvents.has_value()) {
98             leftEvents = leftCamera->getNextEventBatch();
99         }
100         if (!rightEvents.has_value()) {
101             rightEvents = rightCamera->getNextEventBatch();

```

(continues on next page)

(continued from previous page)

```

102     }
103
104     // Feed the data into the slicer and reset the buffer
105     if (leftEvents && rightEvents) {
106         slicer.accept(*leftEvents, *rightEvents);
107         leftEvents = std::nullopt;
108         rightEvents = std::nullopt;
109     }
110
111     // Wait for a small amount of time to avoid CPU overhaul
112     cv::waitKey(1);
113 }
114
115 return 0;
116 }

```

```

1  from datetime import timedelta
2
3  import cv2 as cv
4  import dv_processing as dv
5
6  # Path to a stereo calibration file, replace with a file path on your local file_
   ↪ system
7  calibration_file_path = "path/to/calibration.json"
8
9  # Load the calibration file
10 calibration = dv.camera.CalibrationSet.LoadFromFile(calibration_file_path)
11
12 # It is expected that calibration file will have "C0" as the leftEventBuffer camera
13 left_camera = calibration.getCameraCalibration("C0")
14
15 # The second camera is assumed to be rightEventBuffer-side camera
16 right_camera = calibration.getCameraCalibration("C1")
17
18 # Open the stereo camera with camera names from calibration
19 left_camera_capture = dv.io.camera.openSync(left_camera.name)
20 right_camera_capture = dv.io.camera.openSync(right_camera.name)
21
22 dv.io.camera.synchronizeAnyTwo(left_camera_capture, right_camera_capture)
23
24 # Make sure both cameras support event stream output, throw an error otherwise
25 if not left_camera_capture.isEventStreamAvailable() or not right_camera_capture.
   ↪ isEventStreamAvailable():
26     raise RuntimeError("Input camera does not provide an event stream.")
27
28 # Initialize a stereo block matcher with a stereo geometry from calibration and the_
   ↪ preconfigured SGBM instance
29 block_matcher = dv.SemiDenseStereoMatcher(dv.camera.StereoGeometry(left_camera, right_
   ↪ camera))
30
31 # Initialization of a stereo event sliver
32 slicer = dv.StereoEventStreamSlicer()

```

(continues on next page)

(continued from previous page)

```

33
34 # Initialize a window to show previews of the output
35 cv.namedWindow("Preview", cv.WINDOW_NORMAL)
36
37 # Local event buffers to implement overlapping window of events for accumulation
38 global left_event_buffer, right_event_buffer
39 left_event_buffer = dv.EventStore()
40 right_event_buffer = dv.EventStore()
41
42 # Use one third of the resolution as count of events per accumulated frame
43 event_count = int((left_camera.resolution[0] * left_camera.resolution[1]) / 3)
44
45
46 # Stereo slicer callback method
47 def callback(left_events: dv.EventStore, right_events: dv.EventStore):
48     # Push input events into the local buffers
49     global left_event_buffer, right_event_buffer
50     left_event_buffer.add(left_events)
51     right_event_buffer.add(right_events)
52
53     # If the number of events is above the count, just keep the latest events
54     if len(left_event_buffer) > event_count:
55         left_event_buffer = left_event_buffer.sliceBack(event_count)
56     if len(right_event_buffer) > event_count:
57         right_event_buffer = right_event_buffer.sliceBack(event_count)
58
59     # Pass these events into block matcher and estimate disparity, the matcher will
60     ↳ accumulate frames
61     # internally. The disparity output is 16-bit integer, that has sub-pixel
62     ↳ precision.
63     disparity = block_matcher.computeDisparity(left_event_buffer, right_event_buffer)
64
65     # Convert the accumulated frames into colored images for preview.
66     images = []
67     images.append(cv.cvtColor(block_matcher.getLeftFrame().image, cv.COLOR_GRAY2BGR))
68     images.append(cv.cvtColor(block_matcher.getRightFrame().image, cv.COLOR_GRAY2BGR))
69
70     # Convert disparity into 8-bit integers with scaling and normalize the output for
71     ↳ a nice preview.
72     # This loses the actual numeric value of the disparity, but it's a nice way to
73     ↳ visualize the disparity.
74     # Apply color-mapping to the disparity image, this will encode depth with color:
75     ↳ red - close; blue - far.
76     images.append(cv.applyColorMap(cv.normalize(disparity, None, 0, 255, cv.NORM_
77     ↳ MINMAX, cv.CV_8UC1), cv.COLORMAP_JET))
78
79     # Concatenate images and show them in a window
80     cv.imshow("Preview", cv.hconcat(images))
81
82 # Register a callback to be done at 30Hz
83 slicer.doEveryTimeInterval(timedelta(milliseconds=33), callback)

```

(continues on next page)

(continued from previous page)

```

79
80 # Buffer input events in these variables to synchronize inputs
81 left_events = None
82 right_events = None
83
84 # Run the processing loop while both cameras are connected
85 while left_camera_capture.isRunning() and right_camera_capture.isRunning():
86     # Read events from respective left / right cameras
87     if left_events is None:
88         left_events = left_camera_capture.getNextEventBatch()
89     if right_events is None:
90         right_events = right_camera_capture.getNextEventBatch()
91
92     # Feed the data into the slicer and reset the buffer
93     if left_events is not None and right_events is not None:
94         slicer.accept(left_events, right_events)
95         left_events = None
96         right_events = None
97
98     # Wait for a small amount of time to avoid CPU overhaul
99     cv.waitKey(1)

```

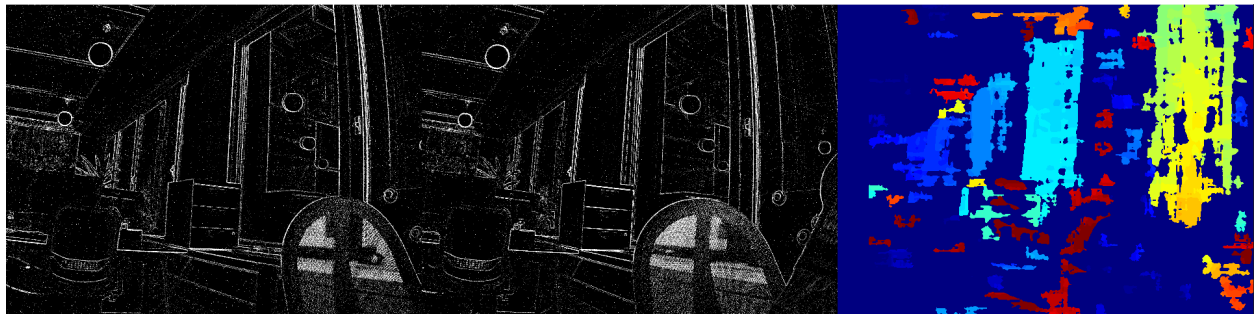


Fig. 1: Expected result of semi-dense disparity estimation. The output provides two accumulated frames and color-coded disparity map.

Note

Disparity map yields results only in areas with visible texture, areas without texture contain speckle noise.

5.1.2 Sparse disparity estimation

The semi-dense approach is a straightforward to stereo disparity estimation. Another approach is to perform disparity estimation on sparse selected regions within accumulated image. Sparse estimation approach allows the implementation to select regions with enough texture to be selected for the disparity, reducing computational complexity and improving quality. The sparse approach takes point coordinates of where the disparity needs to be estimated, performs sparse accumulation only in the regions where disparity matching actually needs to happen and runs correlation based template matching of left image patches on the right camera image. Each template is matched against the other image on a horizontal line using normalized correlation coefficient (Pearson correlation) and the best scoring match is considered to be the correct match and according disparity is assigned to that point.

The following sample code shows the use of sparse disparity block matcher with a live calibrated stereo camera:

C++

Python

```

1  #include <dv-processing/camera/calibration_set.hpp>
2  #include <dv-processing/cluster/mean_shift/event_store_adaptor.hpp>
3  #include <dv-processing/core/stereo_event_stream_slicer.hpp>
4  #include <dv-processing/data/utilities.hpp>
5  #include <dv-processing/depth/sparse_event_block_matcher.hpp>
6  #include <dv-processing/io/camera/discovery.hpp>
7  #include <dv-processing/visualization/colors.hpp>
8
9  #include <opencv2/highgui.hpp>
10
11 int main() {
12     using namespace std::chrono_literals;
13
14     // Path to a stereo calibration file, replace with a file path on your local file.
15     ↪system
16     const std::string calibrationFilePath = "path/to/calibration.json";
17
18     // Load the calibration file
19     auto calibration = dv::camera::CalibrationSet::LoadFromFile(calibrationFilePath);
20
21     // It is expected that calibration file will have "C0" as the leftEventBuffer.
22     ↪camera
23     auto leftCameraCalib = calibration.getCameraCalibration("C0").value();
24
25     // The second camera is assumed to be rightEventBuffer-side camera
26     auto rightCameraCalib = calibration.getCameraCalibration("C1").value();
27
28     // Open the stereo camera with camera names from calibration
29     auto leftCamera = dv::io::camera::openSync(leftCameraCalib.name);
30     auto rightCamera = dv::io::camera::openSync(rightCameraCalib.name);
31
32     dv::io::camera::synchronizeAnyTwo(leftCamera, rightCamera);
33
34     // Make sure both cameras support event stream output, throw an error otherwise
35     if (!leftCamera->isEventStreamAvailable() || !rightCamera->
36     ↪isEventStreamAvailable()) {
37         throw dv::exceptions::RuntimeError("Input camera does not provide an event.
38     ↪stream.");
39     }
40
41     // Matching window size for the block matcher
42     const cv::Size window(24, 24);
43     // Minimum disparity value to measure
44     const int minDisparity = 0;
45     // Maximum disparity value
46     const int maxDisparity = 40;
47     // Minimum z-score value that a valid match can have
48     const float minScore = 0.0f;
49
50     // Initialize the block matcher with rectification

```

(continues on next page)

(continued from previous page)

```

47     auto matcher
48         = dv::SparseEventBlockMatcher(std::make_unique<dv::camera::StereoGeometry>
↳ (leftCameraCalib, rightCameraCalib),
49             window, maxDisparity, minDisparity, minScore);
50
51     // Initialization of a stereo event sliver
52     dv::StereoEventStreamSlicer slicer;
53
54     // Initialize a window to show previews of the output
55     cv::namedWindow("Preview", cv::WINDOW_NORMAL);
56
57     // Local event buffers to implement overlapping window of events for accumulation
58     dv::EventStore leftEventBuffer, rightEventBuffer;
59
60     // Use one third of the resolution as count of events per accumulated frame
61     const size_t eventCount = static_cast<size_t>(leftCameraCalib.resolution.area()) /
↳ 3;
62
63     // Register a callback to be done at 50Hz
64     slicer.doEveryTimeInterval(20ms, [&matcher, &leftEventBuffer, &rightEventBuffer,
↳ eventCount, &window] {
65         const auto &leftEvents, const auto &
↳ rightEvents) {
66             // Push input events into the local buffers
67             leftEventBuffer.add(leftEvents);
68             rightEventBuffer.add(rightEvents);
69
70             // If the number of events is above the count, just keep the latest events
71             if (leftEventBuffer.size() > eventCount) {
72                 leftEventBuffer = leftEventBuffer.sliceBack(eventCount);
73             }
74             if (rightEventBuffer.size() > eventCount) {
75                 rightEventBuffer = rightEventBuffer.sliceBack(eventCount);
76             }
77
78             // Number of clusters to extract
79             constexpr int numClusters = 100;
80
81             // Initialize the mean-shift clustering algorithm
82             dv::cluster::mean_shift::MeanShiftEventStoreAdaptor meanShift(leftEventBuffer,
↳ 10.f, 1.0f, 20, numClusters);
83
84             // Find cluster centers which are going to be used for disparity estimation
85             auto centers = meanShift.findClusterCentres<dv::cluster::mean_
↳ shift::kernel::Epanechnikov>();
86
87             // Run disparity estimation, the output will contain a disparity estimate for
↳ each of the given points.
88             const std::vector<dv::SparseEventBlockMatcher::PixelDisparity> estimates
89                 = matcher.computeDisparitySparse(leftEventBuffer, rightEventBuffer,
↳ dv::data::convertToCvPoints(centers));
90

```

(continues on next page)

(continued from previous page)

```

91     // Convert the accumulated frames into colored images for preview.
92     std::vector<cv::Mat> images(2);
93     cv::cvtColor(matcher.getLeftFrame().image, images[0], cv::COLOR_GRAY2BGR);
94     cv::cvtColor(matcher.getRightFrame().image, images[1], cv::COLOR_GRAY2BGR);
95
96     // Visualize the matched blocks
97     int32_t index = 0;
98     for (const auto &point : estimates) {
99         // If point estimation is invalid, do not show a preview of it
100         if (!point.valid) {
101             continue;
102         }
103
104         // The rest of the code performs drawing of the match according to the
105         ↪disparity value on the
106         // preview images.
107         const cv::Scalar color = ↪
108         ↪dv::visualization::colors::someNeonColor(index++);
109         // Draw some nice colored markers and rectangles.
110         cv::drawMarker(images[1], *point.matchedPosition, color, cv::MARKER_CROSS,
111         ↪ 7);
112         cv::rectangle(images[1],
113             cv::Rect(point.matchedPosition->x - (window.width / 2), point.
114         ↪matchedPosition->y - (window.height / 2),
115             window.width, window.height),
116             color);
117         cv::rectangle(images[0],
118             cv::Rect(point.templatePosition->x - (window.width / 2),
119             point.templatePosition->y - (window.height / 2), window.width, ↪
120         ↪window.height),
121             color);
122     }
123
124     // Concatenate images and show them in a window
125     cv::Mat preview;
126     cv::hconcat(images, preview);
127     cv::imshow("Preview", preview);
128
129     });
130
131     // Buffer input events in these variables to synchronize inputs
132     std::optional<dv::EventStore> leftEvents = std::nullopt;
133     std::optional<dv::EventStore> rightEvents = std::nullopt;
134
135     // Run the processing loop while both cameras are connected
136     while (leftCamera->isRunning() && rightCamera->isRunning()) {
137         // Read events from respective left / right cameras
138         if (!leftEvents.has_value()) {
139             leftEvents = leftCamera->getNextEventBatch();
140         }
141         if (!rightEvents.has_value()) {
142             rightEvents = rightCamera->getNextEventBatch();
143         }
144     }

```

(continues on next page)

(continued from previous page)

```

138
139     // Feed the data into the slicer and reset the buffer
140     if (leftEvents && rightEvents) {
141         slicer.accept(*leftEvents, *rightEvents);
142         leftEvents = std::nullopt;
143         rightEvents = std::nullopt;
144     }
145
146     // Wait for a small amount of time to avoid CPU overhaul
147     cv::waitKey(1);
148 }
149
150 return 0;
151 }

```

```

1  from datetime import timedelta
2
3  import cv2 as cv
4  import dv_processing as dv
5
6  # Path to a stereo calibration file, replace with a file path on your local file_
   ↪ system
7  calibration_file_path = "path/to/calibration.json"
8
9  # Load the calibration file
10 calibration = dv.camera.CalibrationSet.LoadFromFile(calibration_file_path)
11
12 # It is expected that calibration file will have "C0" as the leftEventBuffer camera
13 left_camera = calibration.getCameraCalibration("C0")
14
15 # The second camera is assumed to be rightEventBuffer-side camera
16 right_camera = calibration.getCameraCalibration("C1")
17
18 # Open the stereo camera with camera names from calibration
19 left_camera_capture = dv.io.camera.openSync(left_camera.name)
20 right_camera_capture = dv.io.camera.openSync(right_camera.name)
21
22 dv.io.camera.synchronizeAnyTwo(left_camera_capture, right_camera_capture)
23
24 # Make sure both cameras support event stream output, throw an error otherwise
25 if not left_camera_capture.isEventStreamAvailable() or not right_camera_capture.
   ↪ isEventStreamAvailable():
26     raise RuntimeError("Input camera does not provide an event stream.")
27
28 # Matching window size for the block matcher
29 window = (24, 24)
30
31 # Minimum disparity value to measure
32 min_disparity = 0
33
34 # Maximum disparity value
35 max_disparity = 40

```

(continues on next page)

(continued from previous page)

```

36
37 # Minimum z-score value that a valid match can have
38 min_score = 0.0
39
40 # Initialize the block matcher with rectification
41 matcher = dv.SparseEventBlockMatcher(dv.camera.StereoGeometry(left_camera, right_
    ↪camera), window, max_disparity,
42                                     min_disparity, min_score)
43
44 # Initialization of a stereo event slicer
45 slicer = dv.StereoEventStreamSlicer()
46
47 # Initialize a window to show previews of the output
48 cv.namedWindow("Preview", cv.WINDOW_NORMAL)
49
50 # Local event buffers to implement overlapping window of events for accumulation
51 global left_event_buffer, right_event_buffer
52 left_event_buffer = dv.EventStore()
53 right_event_buffer = dv.EventStore()
54
55 # Use one third of the resolution as count of events per accumulated frame
56 event_count = int((left_camera.resolution[0] * left_camera.resolution[1]) / 3)
57
58
59 # Stereo slicer callback method
60 def callback(left_events: dv.EventStore, right_events: dv.EventStore):
61     # Push input events into the local buffers
62     global left_event_buffer, right_event_buffer
63     left_event_buffer.add(left_events)
64     right_event_buffer.add(right_events)
65
66     # If the number of events is above the count, just keep the latest events
67     if len(left_event_buffer) > event_count:
68         left_event_buffer = left_event_buffer.sliceBack(event_count)
69     if len(right_event_buffer) > event_count:
70         right_event_buffer = right_event_buffer.sliceBack(event_count)
71
72     # Number of clusters to extract
73     num_clusters = 100
74
75     # Initialize the mean-shift clustering algorithm
76     mean_shift = dv.cluster.mean_shift.MeanShiftEventStoreAdaptor(left_event_buffer,
    ↪10, 1, 20, num_clusters)
77
78     # Find cluster centers which are going to be used for disparity estimation
79     centers = mean_shift.findClusterCentresEpanechnikov()
80
81     # Run disparity estimation, the output will contain a disparity estimate for each
    ↪of the given points.
82     estimates = matcher.computeDisparitySparse(left_event_buffer, right_event_buffer,
    ↪list(map(lambda x: x.pt,
83

```

(continues on next page)

(continued from previous page)

```

84         centers)))
85
86     # Convert the accumulated frames into colored images for preview.
87     images = []
88     images.append(cv.cvtColor(matcher.getLeftFrame().image, cv.COLOR_GRAY2BGR))
89     images.append(cv.cvtColor(matcher.getRightFrame().image, cv.COLOR_GRAY2BGR))
90
91     # Visualize the matched blocks
92     index = 0
93     for point in estimates:
94         # If point estimation is invalid, do not show a preview of it
95         if not point.valid:
96             continue
97
98         # The rest of the code performs drawing of the match according to the
99         ↪disparity value on the
100         # preview images.
101         color = dv.visualization.colors.someNeonColor(index)
102         index += 1
103
104         # Draw some nice colored markers and rectangles.
105         cv.drawMarker(images[1], point.matchedPosition, color, cv.MARKER_CROSS, 7)
106         cv.rectangle(images[1],
107                     (int(point.matchedPosition[0] - (window[0] / 2)), int(point.
108         ↪matchedPosition[1] - (window[1] / 2))),
109                     (int(point.matchedPosition[0] + (window[0] / 2)), int(point.
110         ↪matchedPosition[1] + (window[1] / 2))),
111                     color)
112         cv.rectangle(
113             images[0],
114             (int(point.templatePosition[0] - (window[0] / 2)), int(point.
115         ↪templatePosition[1] - (window[1] / 2))),
116             (int(point.templatePosition[0] + (window[0] / 2)), int(point.
117         ↪templatePosition[1] + (window[1] / 2))), color)
118
119     # Concatenate images and show them in a window
120     cv.imshow("Preview", cv.hconcat(images))
121
122     # Register a callback to be done at 30Hz
123     slicer.doEveryTimeInterval(timedelta(milliseconds=33), callback)
124
125     # Buffer input events in these variables to synchronize inputs
126     left_events = None
127     right_events = None
128
129     # Run the processing loop while both cameras are connected
130     while left_camera_capture.isRunning() and right_camera_capture.isRunning():
131         # Read events from respective left / right cameras
132         if left_events is None:
133             left_events = left_camera_capture.getNextEventBatch()
134         if right_events is None:

```

(continues on next page)

(continued from previous page)

```

130     right_events = right_camera_capture.getNextEventBatch()
131
132     # Feed the data into the slicer and reset the buffer
133     if left_events is not None and right_events is not None:
134         slicer.accept(left_events, right_events)
135         left_events = None
136         right_events = None
137
138     # Wait for a small amount of time to avoid CPU overhaul
139     cv.waitKey(1)

```

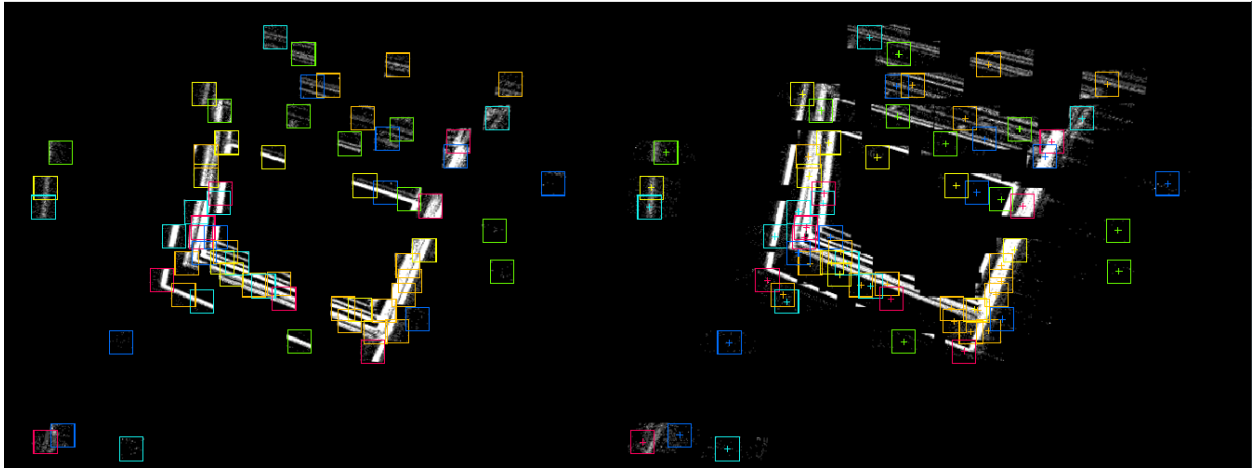


Fig. 2: Expected result of sparse disparity estimation. The colored rectangles represent sparse blocks that are matched on the right side image. Block colors are matched on both images. Note that frame are sparse as well - the accumulation happens only in relevant areas around points of interest. The points of interest are selected on high density event areas as per mean-shift cluster extraction.

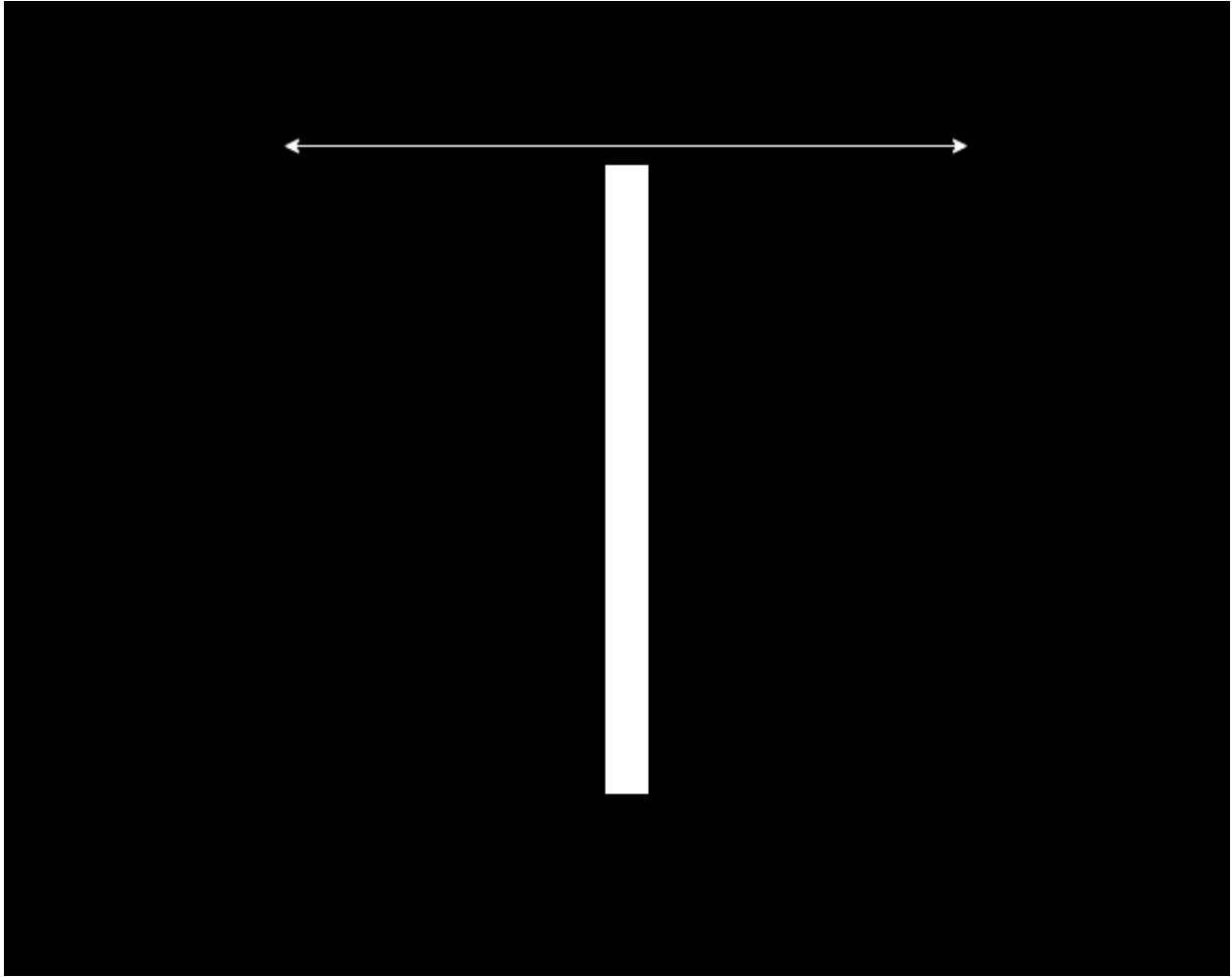
5.2 Motion Compensation

5.2.1 What is the motion compensation algorithm?

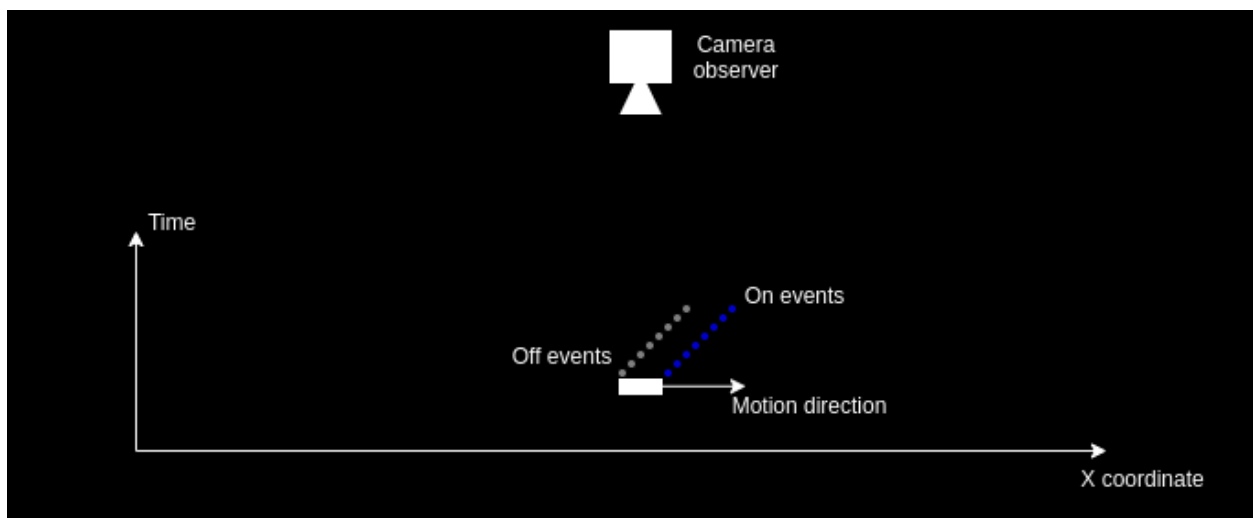
An algorithm to align events in a stream to produce sharp images of edges. Due to camera motion, the events are distributed within the pixel space of a moving camera sensor. If the camera geometry (calibration) and camera motion is known, the camera motion can be applied to the incoming events, so the event locations align to same pixel locations over time.

5.2.2 Visual explanation

Let's discuss a scenario where a white rectangle is moving at a constant speed on a white background. Let's say it's only moving back-and-forth within the X dimension (image below):



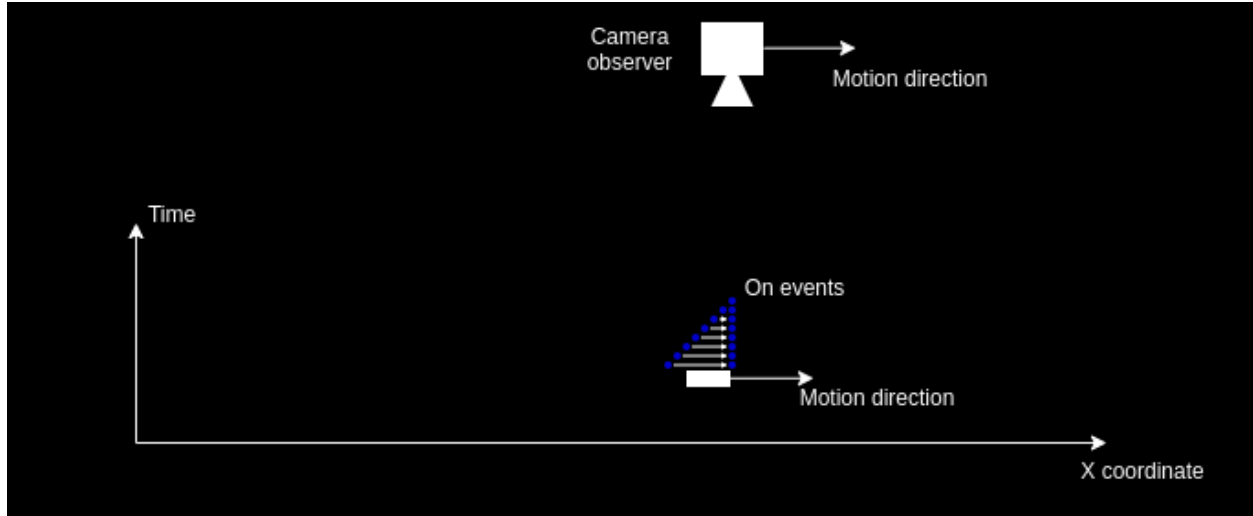
The motion of the rectangle will generate events over time. In the direction of motion the rectangle will generate ON events at the front of rectangle and OFF events at the opposite side. Let's ignore the Y axis and let's look at the event stream from Y axis perspective (given Z axis is time) at the image below.



So the events come in at a slope, so if we accumulate the slice of events they will be represented as a thicker band of ON and OFF events instead of seeing the corresponding edge. This happens due to the fact that the motion happened over

time and event the motion within time-slice is accumulated.

A way to correct this would be to move the camera “virtually” in same direction and speed. This would cause the events to align and be represented at the same pixel location on the sensor (see image below, for simplicity it shows only the ON events aligned).



This is just a scenario to imagine on the concept itself. Usual application case would be when the environment of the camera is static and only the camera moves. If we know the camera motion, we can apply the known motion to the received event coordinate to match the camera motion. In that case the algorithm reverses the problem, assumes the camera is stationary at a single point in time and aligns measured events to happen at the same pixel location in the sensor at a given perspective.

5.2.3 Details of the implementation

Motion compensation applies measured motion of the camera to the event stream. The approach doesn't cover how the motion is estimated: it can be either some external sensor (e.g. IMU) or visual odometry system estimating camera ego-motion, these measurements are assumed to be known by the algorithm.

Camera geometry

To apply known motion of the camera, firstly event pixel coordinates must be represented in a 3D space. This is performed using camera geometry and back-projection. Back-projection is performed to find a pixel coordinate ray that maps pixel coordinates into a flat 3D plane in front of the camera, that is one-meter away from the focal point.

Camera matrix (K in the formula below) contains focal length (f_x , f_y) and principal point (c_x , c_y) coordinates retrieved from calibration of the camera sensor.

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Back-projection can be achieved by multiplying pixel coordinate homogenous representation of $[px, py, 1.0]$ multiplied (px and py are the pixel coordinates on the sensor) by the inverse of the camera matrix.

$$R = K^{-1} \cdot \begin{bmatrix} p_x \\ p_y \\ 1 \end{bmatrix}$$

This estimated ray (R) can be multiplied by an estimated depth distance (d) to the scene provide a reasonable estimate of the actual 3D position that cause the pixel to generate an event (let's call it P). If the distance to the scene is not known, a guess of 3 meters is usually a good enough estimate.

$$P = R \cdot d$$

Kinematics (applying the motion)

Let's denote a transformation matrix that describes a motion in 3D, 4x4 matrix containing a 3x3 rotation matrix and 3x1 translation vector.

$$T = \begin{bmatrix} r & t \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} r_{00} & r_{01} & r_{02} & t_x \\ r_{10} & r_{11} & r_{12} & t_y \\ r_{20} & r_{21} & r_{22} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

By multiplying a 3D point P (that was captured at time t1) by the matrix that describes camera motion between time-points t1 and t2, we can estimate the position of the object at camera perspective t2.

$$\begin{bmatrix} P_{t_2} \\ 1 \end{bmatrix} = T_{t_1}^{t_2} \cdot \begin{bmatrix} P_{t_1} \\ 1 \end{bmatrix}$$

The resulting point at t2 can be normalized by the Z axis (X, Y, and Z components divided by Z) results in a projection at one meter plane.

$$R_{t_2} = \hat{P}_{t_2}$$

which can be projected into pixel frame by multiplying the point by camera matrix K, thus aligning the pixel coordinates in space.

$$\begin{bmatrix} p_x^{t_2} \\ p_y^{t_2} \\ 1 \end{bmatrix} = K \cdot R_{t_2}$$

5.2.4 Usage of available implementation

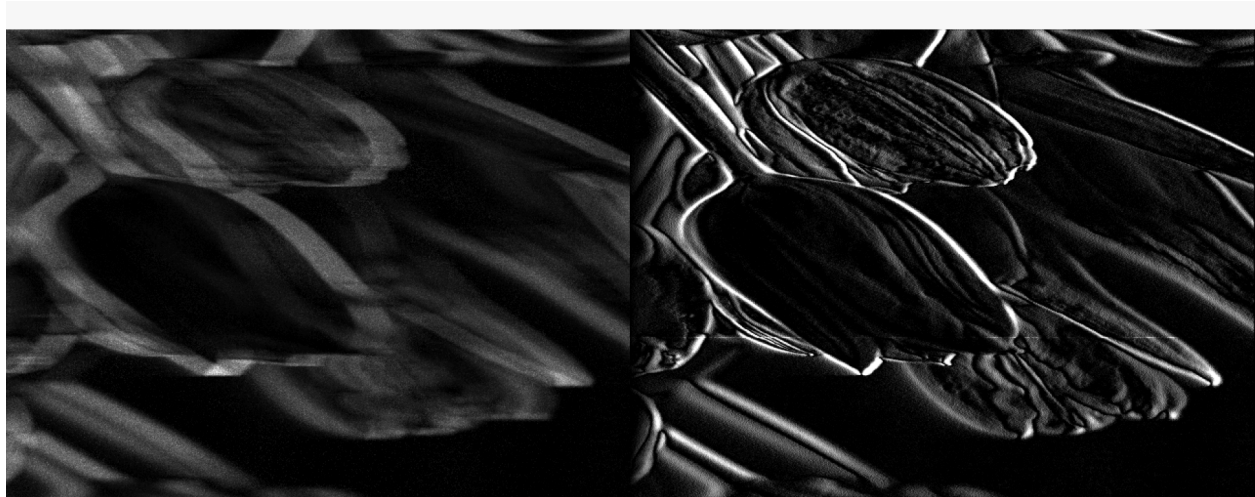
The motion compensation algorithm is available in C++ and Python.

A sample usage of the motion compensator can be found in `samples/motion-compensation-samples/imu_compensation.cpp`. In this sample, measurement from a gyroscope are concatenated to estimate camera rotational motion, this information is then used to reduce motion blur induced by the camera motion.

5.3 Contrast Maximization

5.3.1 What is contrast maximization?

Given a set of events obtained from the camera and/or scene motion, contrast maximization seeks to find the optimal parameters of the motion underlying the generated events. The movement is modeled a-priori (shared by all events) and defines the set of parameters to be optimized. Those parameters are optimized using non-linear optimization, which aims at maximizing the contrast of the event image generated by warped events.



The image on the top shows a sample of an accumulated event image without motion compensation (left) versus an event image after warping events using contrast maximization. Note that, assuming no brightness variation in the scene, each event carries camera or scene motion information.

Examples of possible motions to be optimized are:

- Camera translational motion
- Camera rotational motion
- Full camera ego-motion (translation + depth)
- Optical flow
- Scene depth

Let's now explain the idea of contrast maximization more in detail for pure camera motion (i.e. static scene). The same principles can be applied to a dynamic scene without loss of generality.

Algorithm in details

Events will be generated by a static scene and a camera moving in space due to the camera motion. All events corresponding to the same 3D point in the scene will fall at the same location in the pixels space. Taking a set of events in a given time interval, if the camera's motion in this interval is perfectly known, it is possible to warp events in space to a given previous point in time by applying the inverse of the camera motion. On the other hand, if events are warped with incorrect motion, they will fall at different locations. Based on this simple idea, it is possible to build an objective function that estimates the motion estimate's goodness by analyzing warped events. This is done by measuring the sharpness of the event image generated by warping events with the assumed camera motion.

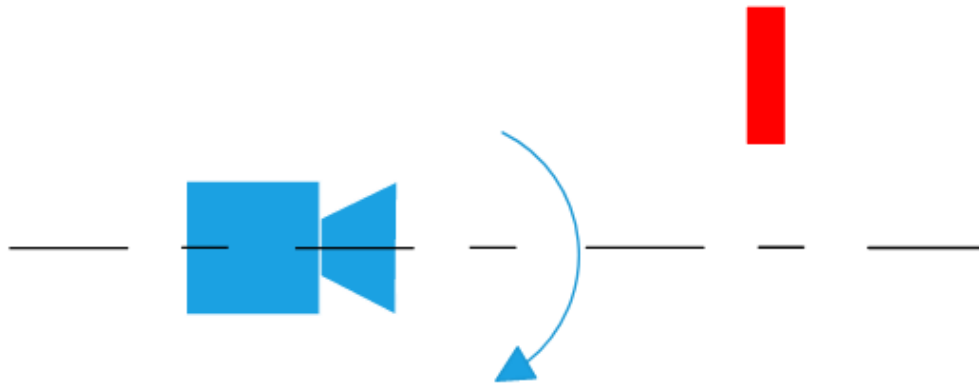
What is contrast?

Since the algorithm's name is contrast maximization, it is important to understand where the name comes from. Indeed, the contrast is the value of the cost function used in the optimization. The contrast of the event image corresponds to computing its variance. If the events are warped with the wrong motion assumption, they will fall at random locations on the image plane. Assuming enough events, there will be at least one event per pixel in the image plane in the extreme case. In this case, the contrast of the image will be 0. On the other hand, if an event image has strong edges, the contrast will be high. Because event-cameras detect edges, it is reasonable to find the camera motion by warping events such that those edges are "reconstructed" at a specific time.

To understand this principle, below you can find an example of how contrast maximization would work for a rotating camera in front of a static bar.

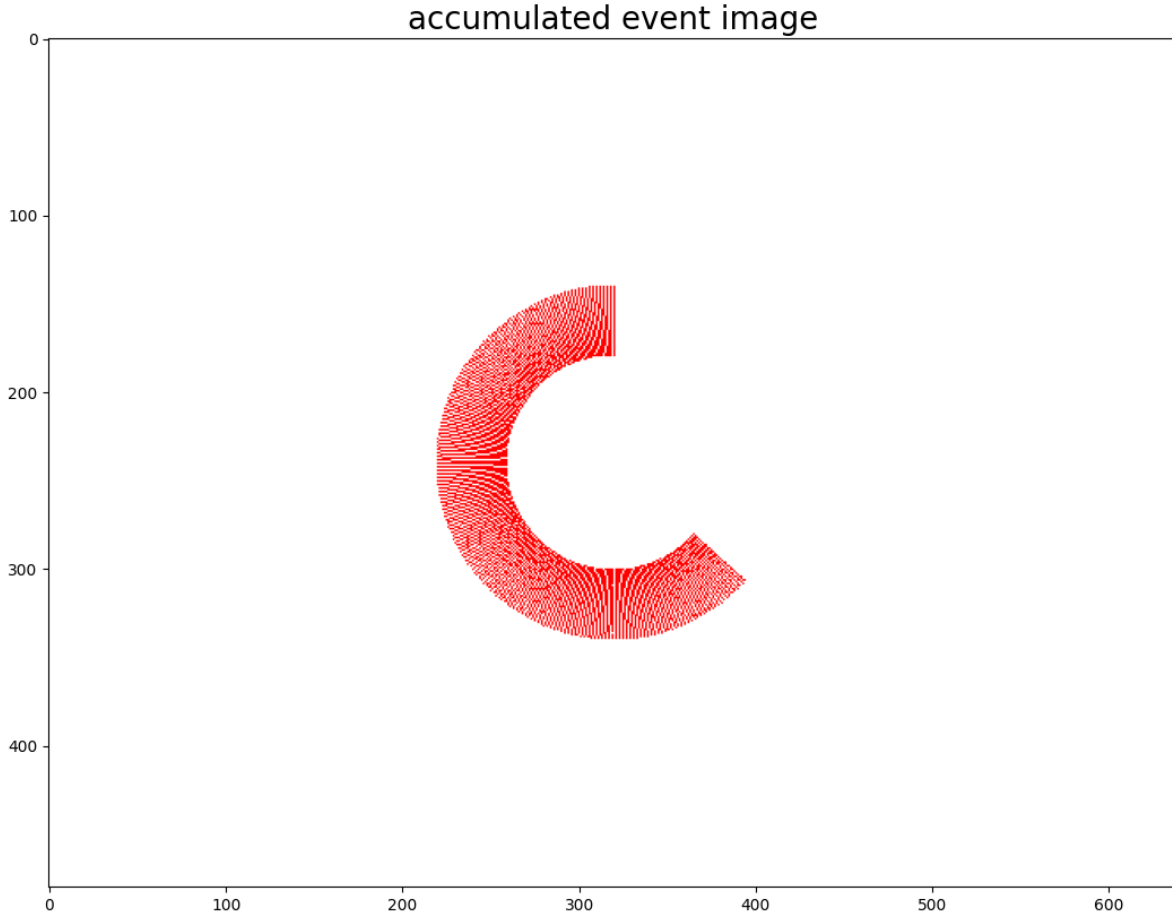
5.3.2 Example: static bar - rotating camera

To understand the basic principle of contrast maximization, we use a simple scene composed of a static bar and a camera pointing to it. We assume the camera motion is defined by a pure rotation around its optical axis. As long as the camera is not moving, no event is generated. The image below shows the example setup viewed from the side.



An x-axis point defines the camera coordinate system to the right, the y-axis to the bottom and the z-axis pointing in front. Once the camera starts rotating at constant speed ω_{camera} (this value is not known) around its optical axis, events

due to the relative motion between the static bar and the camera will be generated on the image plane. By reconstructing the event image corresponding to the accumulated events, we would get an image similar to the one on the bottom.



Now we explain how to estimate the camera speed ω_{camera} from the event set E_{t0-t1} generated in the time interval between $t0$ and $t1$. Select motion model used to explain generated events. We assume the camera to rotate only. The camera motion is defined by a pure rotation $\omega k = [\omega x, \omega y, \omega z]$. Since the camera rotate only around its optical axis we set $\omega x = 0, \omega y = 0$ and optimize only ωz value.

Here is a pseudocode of the non-linear optimization to be solved:

- Solve for ω_{opt} by maximizing variance.
- While (not_converged):
 - current camera speed guess: ωk
 - for each event $ei == [xi, yi, poli, ti]$:
 - * compute the total camera motion between $t0$ and ti as: $\theta z = \omega z(k) \cdot (ti - t0)$. Since the camera rotate only along z axis, it implies $\theta = [0, 0, \theta z]$
 - * get equivalent camera rotation matrix R defined by θ .

$$p_{t(0)} = K^{-1}R^{-1}p_{t(i)}$$

- * warp event ei to time $t0$ using

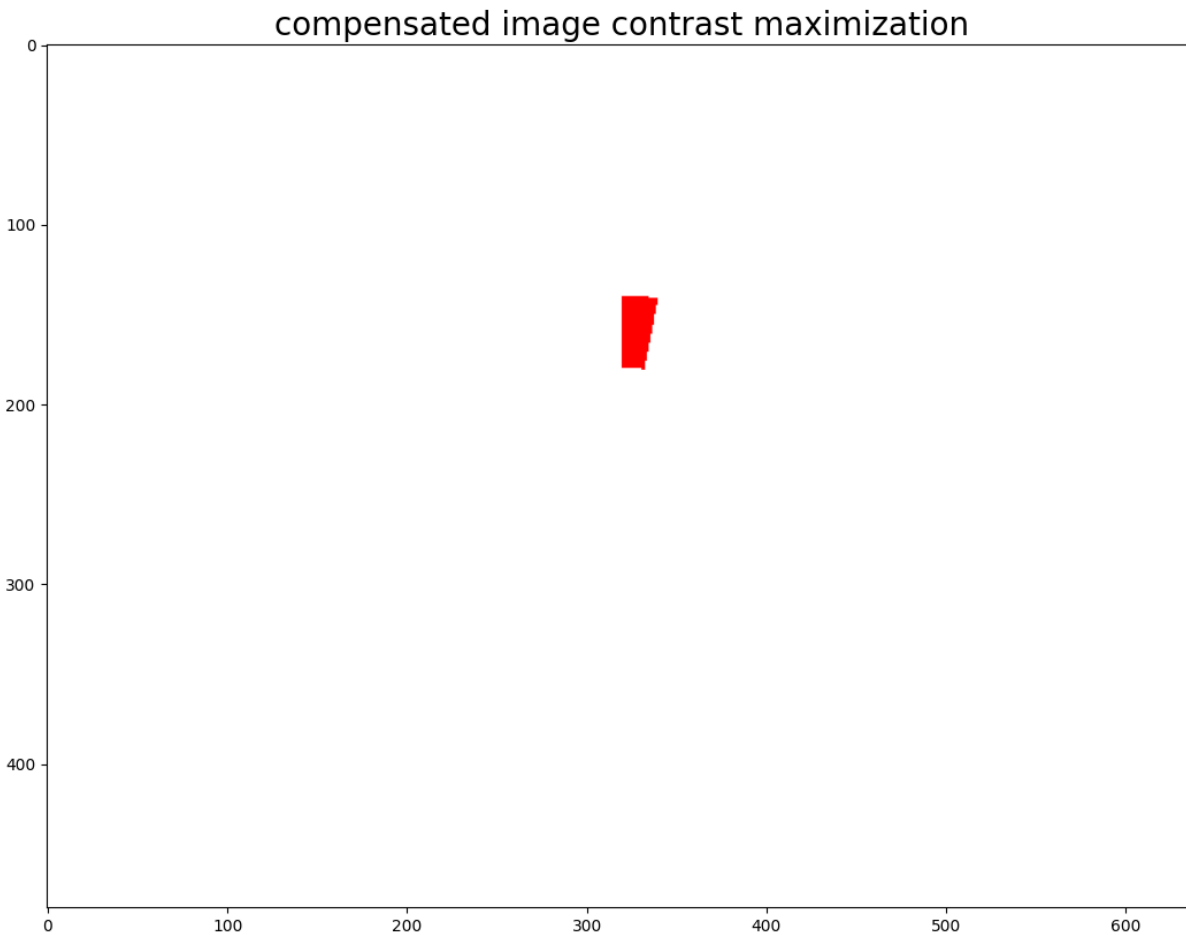
where K is the camera intrinsic matrix,

R is camera rotation matrix defined by θ and $pt(i) = [x_i, y_i, 1]$.

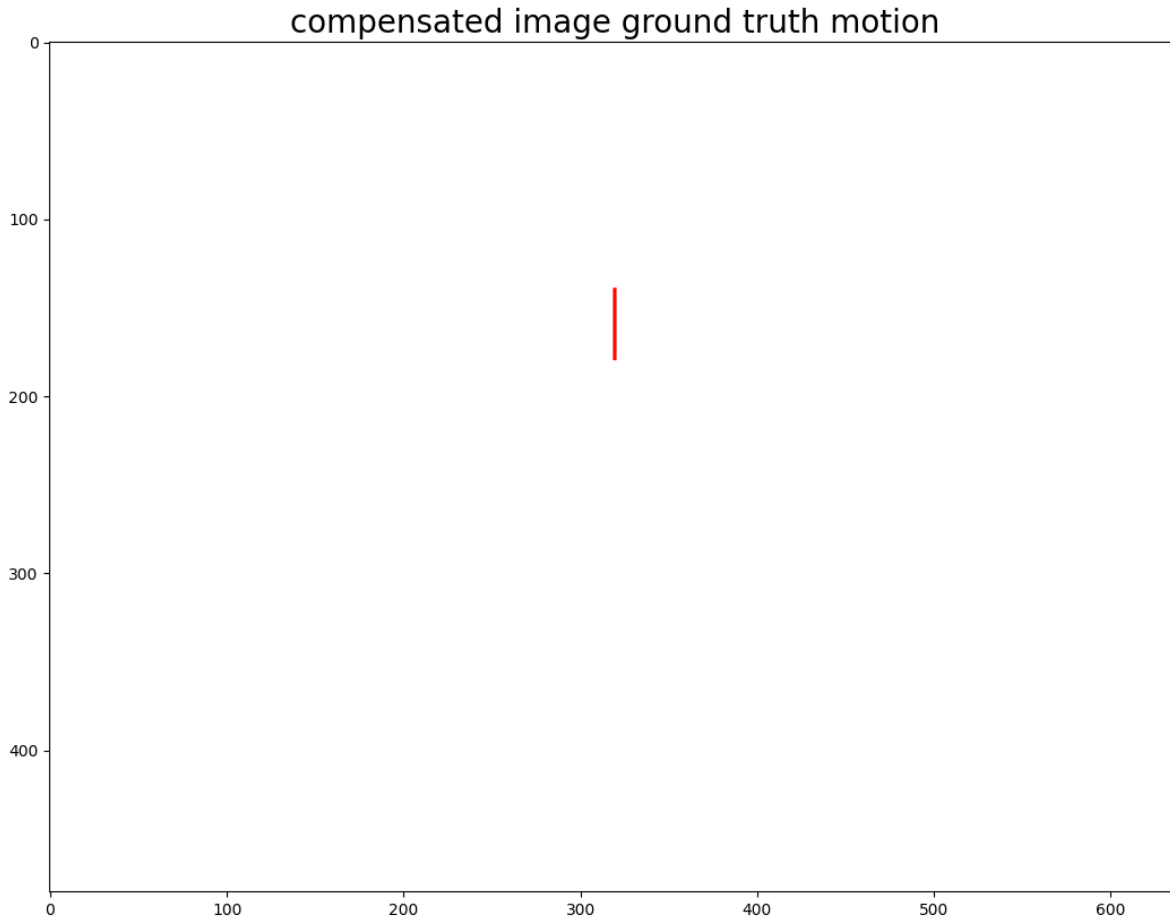
- Get set of warped events E_{warped} , representing all events warped at time t_0 .
- Compute event image from E_{warped} .
- Compute variance of event image

Usually, the camera motion is defined by a 4×4 transformation. In this case, since a pure rotational model is assumed, we neglect the translational part. Because of this, the whole camera motion can be defined by a 3×3 matrix representing the camera rotation.

At each iteration, if the value of ω_k is close to ω_{camera} events will be warped in such a way that the event image will be similar to the image below. Vice versa, if the value of ω_k is far from ω_{camera} , then the events will not fall along the vertical bar, and the generated event image will have lower contrast. The non-linear optimization computes the gradient by evaluating multiple event images at time k , and this gradient is then used to update the camera speed for the next iteration. If the algorithm converges correctly (i.e. $\omega_{opt} \sim \omega_{camera}$), the resulting image will be similar to the one shown on the bottom.



Note that the algorithm rarely converges to the exact motion underlying the generated events. In most cases, the results will be very close to the ground truth motion (if the initialization of the parameters to be optimized is good enough). For completeness, the image at the bottom shows the compensated image assuming perfect motion from the camera is known. As you can see, all events converge to the same vertical line when motion is compensated with ground truth motion.



Summary: algorithm steps

In summary, the steps of the contrast maximization algorithm for a given motion model are the following:

1. Warp events according to the trajectories defined by the model and its model parameters.
2. Generate event image from warped events.
3. Compute score based on the image of warped events.
4. Optimize score for model parameters.

Importance of initial guess

Since contrast maximization is based on a non-linear optimization, the initial guess for the parameters to be optimized has a substantial impact on the accuracy of the final optimization. If the initial guess is very different from the proper motion, the non-linear optimization will most likely end in a local minimum or converge to unexpected results.

5.3.3 Usage of available implementation

The contrast maximization algorithm is available in C++.

There are two sample usages. The first one can be found at `samples/motion-compensation-samples/contrast_maximization_rotation.cpp`: the gyroscope offsets along x, y, and z are optimized

in this sample. The second one can be found at `samples/motion-compensation-samples/contrast_maximization_translation_depth.cpp`: here, the camera translation alone on the three-axis is optimized, together with scene depth.

It is possible to run the code `samples/motion-compensation-samples/contrast_maximization_translation_depth.cpp` using a sample recording that can be downloaded from https://s3.eu-central-1.amazonaws.com/release.inivation.com/datasets/translation_tunnel_sideways.aedat4. The code can be run by passing as argument the path to the downloaded aedat4 file, the path of the corresponding calibration file that can be found in `/docs/source/assets/contrast_maximization/calibration.json` and as initial translation and depth guesses:

depth = 2, translation = [-0.12, 0.0, 0.0].

More information about the two samples can be found in the code.

5.3.4 Limitations

This algorithm is very powerful, but at the same time, it has some inherent limitations that need to be taken into account.

Wrong motion model

The results would not be very accurate if the motion model assumed differs from the underlying motion that generates the event set. This can be the case if multiple motions are present in the scene (e.g., camera motion + dynamic scene). It would be necessary to first cluster events into separate groups based on similar motion and then apply contrast maximization on each event set separately.

Too complex optimization

Another possible cause that might lead to incorrect results is the case in which the optimization is too complex. For example, optimizing both camera rotation and translation simultaneously will create a vast search space for the optimization. The optimization result might degenerate into an unexpected output that does not consider the scene's geometry.

Events not generated by motion

Contrast maximization assumes all events generated by motion, either from the camera or from the scene. However, suppose events are generated from brightness change. Contrast maximization will try to explain those events with some motion, even if events are caused by brightness change and not movement. Due to this, the optimization will converge to unexpected results.

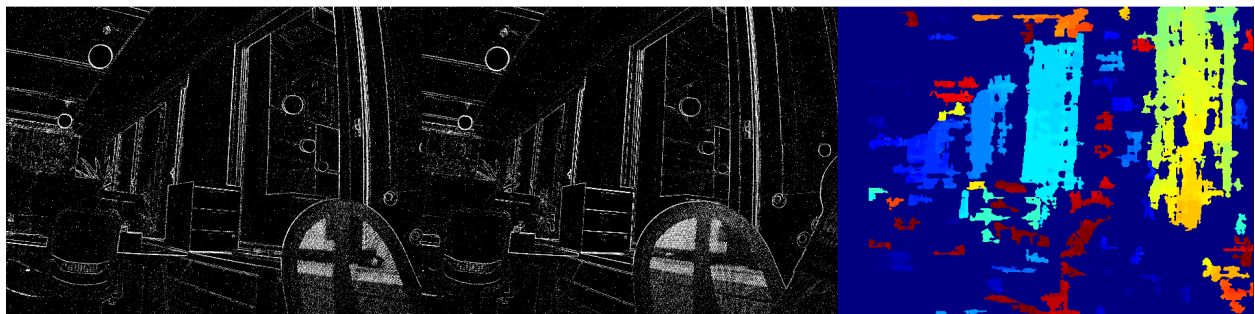


Fig. 3: Expected result of semi-dense disparity estimation. The output provides two accumulated frames and color-coded disparity map.

API documentation provides references for available classes and methods in the library. You can find detailed documentation on each method for their use case, function arguments, and produced outputs.

6.1 API

Full API documentation, automatically generated from doxygen comments.

class **Accumulator** : public *dv::AccumulatorBase*

#include </builds/finivation/dv/dv-processing/include/dv-processing/core/frame/accumulator.hpp> Common accumulator class that allows to accumulate events into a frame. The class is highly configurable to adapt to various use cases. This is the preferred functionality for projecting events onto a frame.

Accumulation of the events is performed on a floating point frame, with every event contributing a fixed amount to the potential. Timestamps of the last contributions are stored as well, to allow for a decay.

Due to performance, no check on the event coordinates inside image plane is performed, unless compiled specifically in DEBUG mode. Events out of the image plane bounds will result in undefined behaviour, or program termination in DEBUG mode.

Public Types

enum class **Decay**

Decay function to be used to decay the surface potential.

- **NONE**: Do not decay at all. The potential can be reset manually by calling the `clear` function
- **LINEAR**: Perform a linear decay with given slope. The linear decay goes from current potential until the potential reaches the neutral potential
- **EXPONENTIAL**: Exponential decay with time factor tau. The potential eventually converges to zero.
- **STEP**: Decay sharply to neutral potential after the given time. Constant potential before.

Values:

enumerator **NONE**

enumerator **LINEAR**

enumerator **EXPONENTIAL**

enumerator **STEP**

Public Functions

inline **Accumulator** ()

Silly default constructor. This generates an accumulator with zero size. An accumulator with zero size does not work. This constructor just exists to make it possible to default initialize an *Accumulator* to later redefine.

inline explicit **Accumulator** (const cv::Size &resolution, *Accumulator*::Decay decayFunction = *Decay*::**EXPONENTIAL**, double decayParam = 1.0e+6, bool synchronousDecay = false, float eventContribution = 0.15f, float maxPotential = 1.0f, float neutralPotential = 0.f, float minPotential = 0.f, bool ignorePolarity = false)

Accumulator constructor Creates a new *Accumulator* with the given params. By selecting the params the right way, the *Accumulator* can be used for a multitude of applications. The class also provides static factory functions that adjust the parameters for common use cases.

Parameters

- **resolution** – The size of the resulting frame. This must be at least the dimensions of the eventstream supposed to be added to the accumulator, otherwise this will result in memory errors.
- **decayFunction** – The decay function to be used in this accumulator. The decay function is one of **NONE**, **LINEAR**, **EXPONENTIAL**, **STEP**. The function behave like their mathematical definitions, with **LINEAR** AND **STEP** going back to the `neutralPotential` over time, **EXPONENTIAL** going back to 0.
- **decayParam** – The parameter to tune the decay function. The parameter has a different meaning depending on the decay function chosen: **NONE**: The parameter is ignored **LINEAR**: The parameter describes the (negative) slope of the linear function **EXPONENTIAL**: The parameter describes tau, by which the time difference is divided.
- **synchronousDecay** – if set to true, all pixel values get decayed to the same time as soon as the frame is generated. If set to false, pixel values remain at the state they had when the last contribution came in.
- **eventContribution** – The contribution a single event has onto the potential surface. This value gets interpreted positively or negatively depending on the event polarity
- **maxPotential** – The upper cut-off value at which the potential surface is clipped
- **neutralPotential** – The potential the decay function converges to over time.
- **minPotential** – The lower cut-off value at which the potential surface is clipped
- **ignorePolarity** – Describes if the polarity of the events should be kept or ignored. If set to true, all events behave like positive events.

inline virtual void **accumulate** (const *EventStore* &packet) override

Accumulates all the events in the supplied packet and puts them onto the accumulation surface.

Parameters

packet – The packet containing the events that should be accumulated.

inline virtual *dv::Frame* **generateFrame** () override

Generates the accumulation frame (potential surface) at the time of the last consumed event. The function writes the output image into the given `frame` argument. The output frame will contain data with type `CV_8U`.

Returns

accumulated frame

inline void **clear**()

Clears the potential surface by setting it to the neutral value. This function does not reset the time surface.

inline void **setIgnorePolarity**(const bool ignorePolarity)

If set to true, all events will incur a positive contribution.

Parameters

ignorePolarity – The new value to set

inline void **setEventContribution**(float eventContribution)

Contribution to the potential surface an event shall incur. This contribution is either counted positively (for positive events or when `rectifyPolatity` is set).

Parameters

eventContribution – The contribution a single event shall incur

inline void **setMaxPotential**(float maxPotential)

Parameters

maxPotential – the max potential at which the surface should be capped at

inline void **setNeutralPotential**(const float neutralPotential)

Set a new neutral potential value. This will also reset the cached potential surface to the given new value.

Parameters

neutralPotential – The neutral potential to which the decay function should go. Exponential decay always goes to 0. The parameter is ignored there.

inline void **setMinPotential**(float minPotential)

Parameters

minPotential – the min potential at which the surface should be capped at

inline void **setDecayFunction**(*Decay* decayFunction)

Parameters

decayFunction – The decay function the module should use to perform the decay

inline void **setDecayParam**(double decayParam)

The decay param. This is slope for linear decay, tau for exponential decay

Parameters

decayParam – The param to be used

inline void **setSynchronousDecay**(bool synchronousDecay)

If set to true, all valued get decayed to the frame generation time at frame generation. If set to false, the values only get decayed on activity.

Parameters

synchronousDecay – the new value for synchronoues decay

inline bool **isIgnorePolarity**() const

Check whether polarity of events is ignored.

Returns

True if polarity is ignored, false otherwise.

inline float **getEventContribution**() const

```
inline float getMaxPotential () const

inline float getNeutralPotential () const

inline float getMinPotential () const

inline Decay getDecayFunction () const

inline double getDecayParam () const

inline Accumulator &operator<< (const EventStore &store)
    Accumulates the event store into the accumulator.
```

Parameters

store – The event store to be accumulated.

Returns

A reference to this *Accumulator*.

```
inline cv::Mat getPotentialSurface () const
```

Retrieve a copy of the currently accumulated potential surface. Potential surface contains raw floating point values aggregated by the accumulator, the values are within the configured range of [minPotential; maxPotential]. This returns a deep copy of the potential surface.

Returns

Potential surface image containing CV_32FC1 data.

Private Functions

```
inline void decay (int16_t x, int16_t y, int64_t time)
```

INTERNAL_USE_ONLY Decays the potential at coordinates x, y to the given time, respecting the decay function. Updates the time surface to the last decay.

Parameters

- **x** – The x coordinate of the value to be decayed
- **y** – The y coordinate of the value to be decayed
- **time** – The time to which the value should be decayed to.

```
inline void contribute (int16_t x, int16_t y, bool polarity)
```

INTERNAL_USE_ONLY Contributes the effect of a single event onto the potential surface.

Parameters

- **x** – The x coordinate of where to contribute to
- **y** – The y coordinate of where to contribute to
- **polarity** – The polarity of the contribution

Private Members

```
bool rectifyPolarity_ = false
```

```
float eventContribution_ = .0
```

```
float maxPotential_ = .0
```

```

float neutralPotential_ = .0

float minPotential_ = .0

Decay decayFunction_ = Decay::NONE

double decayParam_ = .0

bool synchronousDecay_ = false

TimeSurface decayTimeSurface_

cv::Mat potentialSurface_

int64_t highestTime_ = 0

int64_t lowestTime_ = -1

bool resetTimestamp = true

```

Friends

```
inline friend std::ostream &operator<< (std::ostream &os, const Decay &var)
```

class AccumulatorBase

#include </builds/inivation/dv/dv-processing/include/dv-processing/core/frame/accumulator_base.hpp> An accumulator base that can be used to implement different types of accumulators. Two provided implementations are the *dv::Accumulator* which is highly configurable and provides numerous ways of generating a frame from events. Another implementation is the *dv::EdgeMapAccumulator* which accumulates event in a histogram representation with configurable contribution, but it is more efficient compared to generic accumulator since it uses 8-bit unsigned integers as internal memory type.

Subclassed by *dv::Accumulator*, *dv::EdgeMapAccumulator*

Public Types

```
typedef std::shared_ptr<AccumulatorBase> SharedPtr
```

```
typedef std::unique_ptr<AccumulatorBase> UniquePtr
```

Public Functions

```
inline explicit AccumulatorBase (const cv::Size &resolution)
```

Accumulator constructor from known event camera sensor dimensions.

Parameters

resolution – Sensor dimensions

virtual void **accumulate** (const *EventStore* &packet) = 0

Accumulate given event store packet into a frame.

Parameters

packet – Event packet to be accumulated.

inline cv::Size **getResolution** () const

Get the image dimensions expected by the accumulator.

Returns

Image dimensions

virtual *dv::Frame* **generateFrame** () = 0

Generates the accumulation frame (potential surface) at the time of the last consumed event. The function returns an OpenCV frame to work with.

Returns

An OpenCV frame containing the accumulated potential surface.

inline *dv::Frame* &**operator>>** (*dv::Frame* &mat)

Output stream operator support for frame generation.

Parameters

mat – Output image

Returns

Output image

inline void **accept** (const *EventStore* &packet)

Accumulate the given packet.

Parameters

packet – Input event packet.

virtual ~**AccumulatorBase** () = default

Protected Attributes

cv::Size **mResolution**

template<*concepts::AddressableEvent* **EventType**, class **EventPacketType**>

class **AddressableEventStorage**

#include </builds/inivation/dv/dv-processing/include/dv-processing/core/core.hpp> *EventStore* class. An *EventStore* is a collection of consecutive events, all monotonically increasing in time. *EventStore* is the basic data structure for handling event data. Event packets hold their data in shards of fixed size. Copying an *EventStore* results in a shallow copy with shared ownership of the shards that are common to both *EventStores*. *EventStores* can be sliced by number of events or by time. Slicing creates a shallow copy of the *EventPackage*.

Public Types

using **value_type** = *EventType*

using **const_value_type** = const *EventType*

```

using pointer = EventType*

using const_pointer = const EventType*

using reference = EventType&

using const_reference = const EventType&

using size_type = size_t

using difference_type = ptrdiff_t

using packet_type = EventPacketType

using const_packet_type = const EventPacketType

using iterator = AddressableEventStorageIterator<EventType, EventPacketType>

using const_iterator = iterator

```

Public Functions

AddressableEventStorage () = default

Default constructor. Creates an empty *EventStore*. This does not allocate any memory as long as there is no data.

inline void **add** (const *AddressableEventStorage* &store)

Merges the contents of the supplied Event Store into the current event store. This operation can cause event data copies if that results in more optimal memory layout, otherwise the operation only performs shallow copies of the data by sharing the ownership with previous event storage. The two event stores have to be in ascending order.

Parameters

store – the store to be added to this store

inline Eigen::Matrix<int64_t, Eigen::Dynamic, 1> **timestamps** () const

Retrieve timestamps of events into a one-dimensional eigen matrix. This performs a copy of the values. The values are guaranteed to be monotonically increasing.

Returns

A one-dimensional eigen matrix containing timestamps of events.

inline Eigen::Matrix<int16_t, Eigen::Dynamic, 2> **coordinates** () const

Retrieve coordinates of events in a 2xN eigen matrix. Method performs a copy of the values. Coordinates maintain the same order as within the event store. First column is the x coordinate, second column is the y coordinate.

Returns

A two-dimensional eigen matrix containing x and y coordinates of events.

```
inline Eigen::Matrix<uint8_t, Eigen::Dynamic, 1> polarities () const
```

Retrieve polarities of events in a one-dimensional eigen matrix. Method performs a copy of the values. Polarities maintain the same order as within the event store. Polarities are converted into unsigned 8-bit integer values, where 0 stands for negative polarity event and 1 stands for positive polarity event.

Returns

A one-dimensional eigen matrix containing polarities of events.

```
inline EigenEvents eigen () const
```

Convert the event store into eigen matrices. This function performs a deep copy of the memory.

Returns

Events in represented in eigen matrices.

```
inline explicit AddressableEventStorage (std::shared_ptr<const EventPacketType> packet)
```

Creates a new *EventStore* with the data from an *EventPacket*. This is a shallow operation. No data is copied. The *EventStore* gains shared ownership of the supplied data. This constructor also allows the implicit conversion from *dv::InputVectorDataWrapper*<*dv::EventPacket*, *dv::Event*> to *dv::AddressableEventStorage*<*dv::Event*, *dv::EventPacket*> Implicit conversion intended.

Parameters

packet – the packet to construct the *EventStore* from

```
inline AddressableEventStorage &operator= (std::shared_ptr<const EventPacketType> packet)
```

Assignment operator for packet const-pointer type. Will construct a new *EventStore* within the variable.

Parameters

packet – A pointer to the event data packet.

Returns

```
inline void push_back (const EventType &event)
```

Adds a single *Event* to the *EventStore*. This will potentially allocate more memory when the currently available shards are exhausted. Any new memory receives exclusive ownership by this packet.

Parameters

event – A reference to the event to be added.

```
inline void push_back (EventType &&event)
```

Moves a single *Event* into the *EventStore*. This will potentially allocate more memory when the currently available shards are exhausted. Any new memory receives exclusive ownership by this packet.

Parameters

event – A movable reference to the event to be added.

```
template<class ...Args>
```

```
inline EventType &emplace_back (Args&&... args)
```

Construct an event at the end of the storage.

Template Parameters

_constr_args – Argument template

Parameters

_args – Argument values

Returns

Reference to the last newly created element

inline *AddressableEventStorage* **operator+** (const *AddressableEventStorage* &other) const

Returns a new EventStore that is the sum of this event store as well as the supplied event store. This is a const operation that does not modify this event store. The returned event store holds all the data of this store and the other. This is a shallow operation, no event data has to be copied for this.

Parameters

other – The other store to be added

Returns

A new EventStore, containing the events from this and the other store

inline void **operator+=** (const *AddressableEventStorage* &other)

Adds all the events of the other event store to this event store.

Parameters

other – The event store to be added

inline *AddressableEventStorage* &**operator<<** (const *EventType* &event)

Adds the given event to the end of this EventStore.

Parameters

event – The event to be added

Returns

A reference to this EventStore.

inline size_t **size** () const noexcept

Returns the total size of the EventStore.

Returns

The total size (in events) of the packet.

inline *AddressableEventStorage* **slice** (const size_t start, const size_t length) const

Returns a new EventStore which is a shallow representation of a slice of this EventStore. The slice is from *start* (number of, events, minimum 0, maximum *getLength()*) and has a length of *length*.

As a slice is a shallow representation, no EventData gets copied by this operation. The resulting EventStore receives shared ownership over the relevant parts of the data. Should the original EventStore get out of scope, memory that is not relevant to the sliced EventStore will get freed.

Parameters

- **start** – The start index of the slice (in number of events)
- **length** – The desired length of the slice (in number of events)

Returns

A new EventStore object which references to the sliced, shared data. No Event data is copied.

inline *AddressableEventStorage*<*EventType*, *EventPacketType*> **slice** (const size_t start) const

Returns a new EventStore which is a shallow representation of a slice of this EventStore. The slice is from *start* (number of, events, minimum 0, maximum *getLength()*) and goes to the end of the EventStore. This method slices off the front of an EventStore.

As a slice is a shallow representation, no EventData gets copied by this operation. The resulting EventStore receives shared ownership over the relevant parts of the data. Should the original EventStore get out of scope, memory that is not relevant to the sliced EventStore will get freed.

Parameters

start – The start index of the slice (in number of events). The slice will be from this index to the end of the packet.

Returns

A new EventStore object which references to the sliced, shared data. No Event data is copied.

```
inline AddressableEventStorage sliceTime (const int64_t startTime, const int64_t endTime, size_t &retStart,  
                                         size_t &retEnd) const
```

Returns a new EventStore which is a shallow representation of a slice of this EventStore. The slice is from a specific startTime (in event timestamps, microseconds) to a specific endTime (event timestamps, microseconds). The actual size (in events) of the resulting packet depends on the event rate in the requested time interval. The resulting packet may be empty, if there is no event that happened in the requested interval.

As a slice is a shallow representation, no EventData gets copied by this operation. The resulting EventStore receives shared ownership over the relevant parts of the data. Should the original EventStore get out of scope, memory that is not relevant to the sliced EventStore will get freed.

The sliced output will be in the time range [startTime, endTime), endTime is exclusive.

Parameters

- **startTime** – The start time of the required slice (inclusive)
- **endTime** – The end time of the required time (exclusive)
- **retStart** – parameter that will get set to the actual index (in number of events) at which the start of the slice occurred.
- **retEnd** – parameter that will get set to the actual index (in number of events) at which the end of the slice occurred

Returns

A new EventStore object that is a shallow representation to the sliced, shared data. No data is copied over.

```
inline AddressableEventStorage sliceTime (const int64_t startTime, const int64_t endTime) const
```

Returns a new EventStore which is a shallow representation of a slice of this EventStore. The slice is from a specific startTime (in event timestamps, microseconds) to a specific endTime (event timestamps, microseconds). The actual size (in events) of the resulting packet depends on the event rate in the requested time interval. The resulting packet may be empty, if there is no event that happend in the requested interval.

As a slice is a shallow representation, no EventData gets copied by this operation. The resulting EventStore receives shared ownership over the relevant parts of the data. Should the original EventStore get out of scope, memory that is not relevant to the sliced EventStore will get freed.

The sliced output will be in the time range [startTime, endTime), endTime is exclusive.

Parameters

- **startTime** – The start time of the required slice (inclusive)
- **endTime** – The end time of the required time (exclusive)

Returns

A new EventStore object that is a shallow representation to the sliced, shared data. No data is copied over.

```
inline AddressableEventStorage sliceBack (const size_t length) const
```

Returns a new EventStore which is a shallow representation of a slice of this EventStore. Returns a slice which contains events from the back of the storage, it will contain no more events than given length variable.

As a slice is a shallow representation, no EventData gets copied by this operation. The resulting EventStore receives shared ownership over the relevant parts of the data. Should the original EventStore get out of scope, memory that is not relevant to the sliced EventStore will get freed.

Parameters

length – Maximum number of events contained in the resulting slice.

Returns

A new EventStore object that is a shallow representation to the sliced, shared data. No data is copied over.

```
inline AddressableEventStorage sliceTime (const int64_t startTime) const
```

Returns a new EventStore which is a shallow representation of a slice of this EventStore. The slice is from a specific startTime (in event timestamps, microseconds) to the end of the packet. The actual size (in events) of the resulting packet depends on the event rate in the requested time interval. The resulting packet may be empty, if there is no event that happened in the requested interval.

As a slice is a shallow representation, no EventData gets copied by this operation. The resulting EventStore receives shared ownership over the relevant parts of the data. Should the original EventStore get out of scope, memory that is not relevant to the sliced EventStore will get freed.

Parameters

startTime – The start time of the required slice, if positive. If negative, the number of microseconds from the end of the store

Returns

A new EventStore object that is a shallow representation to the sliced, shared data. No data is copied over.

```
inline AddressableEventStorage sliceRate (const double targetRate) const
```

Slices events from back of the EventStore, so that the EventStore would only contain a number of events of a given event rate. Useful for performance limited applications when it is required to limit the rate of events to maintain stable execution time.

Parameters

targetRate – Target event rate in events per second.

Returns

New event store which contains number of events within the target event rate.

```
inline const_iterator begin () const noexcept
```

Returns an iterator to the begin of the EventStore

Returns

an iterator to the begin of the EventStore

```
inline const_iterator end () const noexcept
```

Returns an iterator to the end of the EventStore

Returns

an iterator to the end of the EventStore

```
inline const_reference front () const
```

Returns a reference to the first element of the packet

Returns

a reference to the first element to the packet

```
inline const_reference back () const
```

Returns a reference to the last element of the packet

Returns

a reference to the last element to the packet

inline int64_t **getLowestTime** () const

Returns the timestamp of the first event in the packet. This is also the lowest timestamp in the packet, as the events are required to be monotonic.

Returns

The lowest timestamp present in the packet. 0 if the packet is empty.

inline int64_t **getHighestTime** () const

Returns the timestamp of the last event in the packet. This is also the highest timestamp in the packet, as the events are required to be monotonic.

Returns

The highest timestamp present in the packet. 0 if the packet is empty

inline bool **isEmpty** () const

Returns true if the packet is empty (does not contain any events).

Returns

Returns true if the packet is empty (does not contain any events).

inline void **erase** (const size_t start, const size_t length)

Erase given range of events from the event store. This does not necessarily delete the underlying data since event store maps the data using smart pointers, the data will be cleared only in the case that none of the stores is mapping the data. This erase function does not affect data shared with other event stores.

Parameters

- **start** – Start index of events to erase
- **length** – Number of events to erase

inline size_t **eraseTime** (const int64_t startTime, const int64_t endTime)

Erase events in the range between given timestamps. This does not necessarily delete the underlying data since event store maps the data using smart pointers, the data will be cleared only in the case that none of the stores is mapping the data. This erase function does not affect data shared with other event stores.

Parameters

- **startTime** – Start timestamp for events to be erased, including this exact timestamp
- **endTime** – End timestamp for events to be erased, up to this time, events with this exact timestamp are not going to be erased.

Returns

Number of events deleted

inline const *EventType* &**operator**[] (const size_t index) const

Return an event at given index.

Parameters

index – Index of the event

Returns

Reference to the event at the index.

inline const *EventType* &**at** (const size_t index) const

Return an event at given index.

Parameters

index – Index of the event

Returns

Reference to the event at the index.

```
inline void retainDuration (const dv::Duration duration)
```

Retain a certain duration of event data in the event store. This will retain latest events and delete oldest data. The duration is just a hint of minimum amount of duration to keep, the exact duration will always be slightly greater (depending on event rate and memory allocation).

Parameters

duration – Minimum amount of time to keep in the event store. Events are erased in batches, so this guarantees only to maintain the batches of events within this duration.

```
inline dv::Duration duration () const
```

Get the duration of events contained.

Returns

Duration of stored events in microseconds.

```
inline bool isWithinStoreTimeRange (const int64_t timestamp) const
```

Checks whether given timestamp is within the time range of the event store.

Parameters

timestamp – Microsecond Unix timestamp to check.

Returns

True if the timestamp is within the time of event store, false otherwise.

```
inline size_t getShardCapacity () const
```

Get currently used default shard (data partial) capacity value.

Returns

Default capacity for new shards.

```
inline void setShardCapacity (const size_t shardCapacity)
```

Set a new capacity for shards (data partials). Setting this value does not affect already allocated shards and will be used only when a new shard needs to be allocated. If passed in capacity is set to 0, the setter will use a capacity value of 1, because that is the lowest allowed capacity value.

Parameters

shardCapacity – Capacity of events for newly allocated shards.

```
inline size_t getShardCount () const
```

Get the amount of shards that are currently referenced by the event store.

Returns

Number of referenced shards (data partials).

```
inline double rate () const
```

Get the event rate (events per second) for the events stored in this storage.

Returns

Events per second within this storage.

```
inline EventPacketType toPacket () const
```

Convert event store into a continuous memory packet. This performs a deep copy of underlying data.

Returns

Event packet with a copy of all stored events in this event store.

Protected Types

using **PartialEventData**Type = *PartialEventData*<Event**Type**, EventPacket**Type**>

Protected Functions

inline explicit **AddressableEventStorage** (const *std::vector*<*PartialEventData***Type**> &dataPartials)

INTERNAL USE ONLY Creates a new EventStore based on the supplied *PartialEventData* objects. Offsets and meta information is recomputed from the supplied list. The packet gets shared ownership of all underlying data of the *PartialEventData* slices in dataPartials.

Parameters

dataPartials – vector of *PartialEventData* to construct this package from.

inline *PartialEventData***Type** &**_getLastNonFullPartial** ()

Retrieve the last partial that can store events. If available partial is full or no partials available at all, this function will instantiate, add the partial to the store, and return a reference to that partial.

Returns

Last data partial that can store an additional event.

Protected Attributes

std::vector<*PartialEventData***Type**> **dataPartials_**

internal list of the shards.

std::vector<size_t> **partialOffsets_**

The exact number-of-events global offsets of the shards

size_t **totalLength_** = {0}

The total length of the event package

size_t **shardCapacity_** = {10000}

Default capacity for the data partials

Friends

friend class dv::io::MonoCameraWriter

friend class dv::io::NetworkWriter

inline friend *std::ostream* &**operator**<< (*std::ostream* &os, const *AddressableEventStorage* &storage)

template<*concepts::AddressableEvent* Event**Type**, class EventPacket**Type**>

class **AddressableEventStorageIterator**

#include </builds/inivation/dv/dv-processing/include/dv-processing/core/core.hpp> Iterator for the EventStore class.

Public Types

using **iterator_category** = *std::bidirectional_iterator_tag*

```
using value_type = const EventType
```

```
using pointer = const EventType*
```

```
using reference = const EventType&
```

```
using size_type = size_t
```

```
using difference_type = ptrdiff_t
```

Public Functions

```
inline AddressableEventStorageIterator ()
```

Default constructor. Creates a new iterator at the beginning of the packet

```
inline explicit AddressableEventStorageIterator (const std::vector<PartialEventData<EventType,  
                                                    EventPacketType>> *dataPartialsPtr, const bool front)
```

Creates a new Iterator either at the beginning or at the end of the package

Parameters

- **dataPartialsPtr** – to the partials (shards) of the packet
- **front** – iterator will be at the beginning (true) of the packet, or at the end (false) of the packet.

```
inline AddressableEventStorageIterator (const std::vector<PartialEventData<EventType,  
                                                    EventPacketType>> *dataPartialsPtr, const size_t partialIndex,  
                                                    const size_t offset)
```

INTERNAL USE ONLY Creates a new iterator at the specific internal position supplied

Parameters

- **dataPartialsPtr** – Pointer to the partials (shards) of the packet
- **partialIndex** – Index pointing to the active shard
- **offset** – Offset in the active shard

```
inline reference operator* () const noexcept
```

Returns

A reference to the Event at the current iterator position

```
inline pointer operator--> () const noexcept
```

Returns

A pointer to the Event at current iterator position

```
inline AddressableEventStorageIterator &operator++ () noexcept
```

Increments the iterator by one

Returns

A reference to the the same iterator, incremented by one

inline const *AddressableEventStorageIterator* **operator++** (int) noexcept

Post-increments the iterator by one

Returns

A new iterator at the current position. Increments original iterator by one.

inline *AddressableEventStorageIterator* &**operator+=** (const *size_type* add) noexcept

Increments iterator by a fixed number and returns reference to itself

Parameters

add – amount one wishes to increment the iterator

Returns

reference to itself incremented by **add**

inline *AddressableEventStorageIterator* &**operator--** () noexcept

Decrements the iterator by one

Returns

A reference to the the same iterator, decremented by one

inline const *AddressableEventStorageIterator* **operator--** (int) noexcept

Post-decrement the iterator by one

Returns

A new iterator at the current position. Decrements original iterator by one.

inline *AddressableEventStorageIterator* &**operator-=** (const *size_type* sub) noexcept

Decrements iterator by a fixed number and returns reference to itself

Parameters

sub – amount one wishes to decrement the iterator

Returns

reference to itself decremented by **sub**

inline bool **operator==** (const *AddressableEventStorageIterator* &rhs) const noexcept

Parameters

rhs – iterator to compare to

Returns

true if both iterators point to the same element

inline bool **operator!=** (const *AddressableEventStorageIterator* &rhs) const noexcept

Parameters

rhs – iterator to compare to

Returns

true if both iterators point to different elements

Private Functions

inline void **increment** ()

Increments the iterator to the next event. If the iterator goes beyond available data, it remains at this position.

inline void **decrement** ()

Decrements the iterator to the previous event. If the iterator goes below zero, it remains at zero.

Private Members

const *std::vector*<*PartialEventData*<*EventType*, *EventPacketType*>> ***dataPartialsPtr_**

size_t **partialIndex_**

The current partial (shard) we point to

size_t **offset_**

The current offset inside the shard we point to

template<class **EventStoreType**>

class **AddressableStereoEventStreamSlicer**

Public Functions

inline void **accept** (const *std::optional*<*EventStoreType*> &left, const *std::optional*<*EventStoreType*> &right)

Adds EventStores from the left and right camera. Performs job evaluation immediately.

Parameters

- **leftEvents** – the EventStore from left camera.
- **rightEvents** – the EventStore from right camera.

inline int **doEveryNumberOfEvents** (const size_t n, *std::function*<void(const *EventStoreType*&, const *EventStoreType*&> callback)

Perform an action on the stereo stream data every given amount of events. Event count is evaluated on the left camera stream and according time interval of data is sliced from the right camera event stream. Sliced data is passed into the callback function as soon as it arrived, first argument is left camera events and second is right camera events. Since right camera events are sliced by the time interval of left camera, the amount of events on right camera can be different.

See also

AddressableEventStreamSlicer::doEveryNumberOfEvents

Parameters

- **n** – the interval (in number of events) in which the callback should be called.
- **callback** – the callback function that gets called on the data every interval.

Returns

Job identifier

inline int **doEveryTimeInterval** (const *dv::Duration* interval, *std::function*<void(const *EventStoreType*&, const *EventStoreType*&> callback)

Perform an action on the stereo stream data every given time interval. Event period is evaluated on the left camera stream and according time interval of data is sliced from the right camera event stream. Sliced data is passed into the callback function as soon as it arrived, first argument is left camera events and second is right camera events.

 See also

AddressableEventStreamSlicer::doEveryTimeInterval

Parameters

- **interval** – Time interval to call the callback function. The callback is called based on timestamps of left camera.
- **callback** – Function to be executed

Returns

Job identifier.

```
inline bool hasJob (const int job)
```

Returns true if the slicer contains the slicejob with the provided id

Parameters

job – the id of the slicejob in question

Returns

true, if the slicer contains the given slicejob

```
inline void removeJob (const int job)
```

Removes the given job from the list of current jobs.

Parameters

job – The job id to be removed

Protected Functions

```
inline void clearRightEventsBuffer (const int64_t timestampFrom)
```

Perform book-keeping of the right camera buffer by retaining data from a given timestamp. Events are “forgot” only if minimum amount and time duration values are maintained according to slicing configuration.

Parameters

timestampFrom – Perform book-keeping by retaining data from this timestamp onward.

Protected Attributes

```
std::optional<size_t> minimumEvents = std::nullopt
```

```
std::optional<dv::Duration> minimumTime = std::nullopt
```

```
StreamSlicer<EventStoreType> slicer
```

```
EventStoreType leftEvents
```

```
EventStoreType rightEvents
```

```
int64_t rightEventSeek = -1
```

```
struct AedatFileError
```

Public Types

```
using Info = std::filesystem::path
```

```
struct AedatFileParseError
```

Public Types

```
using Info = std::filesystem::path
```

Public Static Functions

```
static inline std::string format (const Info &info)
```

```
struct AedatVersionError
```

Public Types

```
using Info = int32_t
```

Public Static Functions

```
static inline std::string format (const Info &info)
```

```
template<dv::concepts::TimeSurface<dv::EventStore> TimeSurface = dv::TimeSurface, size_t radius1 = 5, size_t  
radius2 = 6>
```

```
class ArcCornerDetector
```

Public Types

```
using UniquePtr = std::unique_ptr<ArcCornerDetector>
```

```
using SharedPtr = std::shared_ptr<ArcCornerDetector>
```

Public Functions

```
ArcCornerDetector () = delete
```

```
template<typename ...TIME_SURFACE_ADDITIONAL_ARGS>
```

```
inline ArcCornerDetector (const cv::Size resolution, const typename TimeSurface::Scalar range, const bool  
resetTsAtEachIteration, TIME_SURFACE_ADDITIONAL_ARGS&&...  
timeSurfaceAdditionalArgs)
```

Constructor

Template Parameters

TIME_SURFACE_ADDITIONAL_ARGS – Types of the additional arguments passed to the time surface constructor

Parameters

- **resolution** – camera dimensions

- **range** – the range within which the timestamps of a corner should be for it to be detected as a corner
- **resetTsAtEachIteration** – set to true if the time surface should be reset at each iteration
- **timeSurfaceAdditionalArgs** – arguments passed to the time surface constructor in addition to the resolution

```
inline std::vector<dv::TimedKeyPoint> detect (const dv::EventStore &events, const cv::Rect &roi, const cv::Mat &mask)
```

Runs the detection algorithm.

A corner is defined by two arcs of different radii containing timestamps which satisfy the following conditions:

- All timestamps that are on the corner are within a range of mCornerRange.
- No timestamp that is outside of this corner is greater than or equal to the minimum timestamp within the corner
- Length of the arc is within the ranges [ArcLimits::MIN_ARC_SIZE_FACTOR * circumference, ArcLimits::MAX_ARC_SIZE_FACTOR * circumference].

See also

ArcLimits.

Parameters

- **events** – events
- **roi** – region of interest
- **mask** – mask containing zeros for all pixels which should be ignored and nonzero for all others

Returns

a vector containing the detected keypoints. The response is defined as the difference between the minimum timestamp within the arc and the maximum timestamp outside of the arc.

```
inline auto getTimeSurface (const bool polarity) const
```

Returns the TimeSurface for a given polarity

Parameters

polarity – the polarity

Returns

the requested time surface

Private Functions

```
inline auto insideCorner (const int64_t ts1, const int64_t ts2)
```

```
template<typename ITERATOR>
```

```
inline auto expandArc (const ITERATOR &maxTimestampLoc, const int64_t maxTimestampValue, const  
dv::Event &event, const CircularTimeSurfaceView &circle)
```

```
template<typename ITERATOR>
```

```
inline auto checkSurroundingTimestamps (const ITERATOR &arcBegin, const ITERATOR arcEnd, const  
int64_t minTimestampInArc, const dv::Event &event, const  
CircularTimeSurfaceView &circle)
```

Private Members

```
std::array<TimeSurface, 2> mTimeSurfaces
```

```
int64_t mCornerRange
```

```
bool mResetTsAfterDetection
```

```
std::array<CircularTimeSurfaceView, 2> mCircles
```

```
std::array<ArcLimits, 2> mArcLimits
```

```
class ArcLimits
```

Public Functions

```
inline explicit ArcLimits (const size_t circumference)
```

```
inline auto satisfied (const size_t arcSize) const
```

Private Members

```
const size_t mCircumference
```

```
const size_t mMinSize
```

```
const size_t mMaxSize
```

Private Static Attributes

```
static constexpr float MIN_ARC_SIZE_FACTOR = 0.125f
```

```
static constexpr float MAX_ARC_SIZE_FACTOR = 0.4f
```

```
template<dv::concepts::EventStorage EventStoreClass = dv::EventStore>
```

```
class BackgroundActivityNoiseFilter : public dv::EventFilterBase<dv::EventStore>
```

Public Functions

```
inline explicit BackgroundActivityNoiseFilter (const cv::Size &resolution, const dv::Duration timeDelta
                                              = dv::Duration(2 * 1000))
```

Initiate a background activity noise filter, which test the neighbourhoods of incoming events for other supporting events that happened within the background activity period.

Parameters

- **resolution** – Sensor resolution.
- **timeDelta** – Background activity duration.

```
inline virtual bool retain (const typename EventStoreClass::value_type &evt) noexcept override
```

Test the background activity, if the event neighbourhood has at least one event that was triggered within the background activity duration, the event will not be considered noise and should be retained, and discarded otherwise.

Parameters

evt – Event to be checked.

Returns

True to retain event, false to discard.

```
inline BackgroundActivityNoiseFilter &operator<< (const EventStoreClass &events)
```

Accept events using the input stream operator.

Parameters

events – Input events.

Returns

```
inline dv::Duration getBackgroundActivityDuration () const
```

Get currently configured background activity duration value.

Returns

Background activity duration value.

```
inline void setBackgroundActivityDuration (const dv::Duration timeDelta)
```

Set new background activity duration value.

Parameters

timeDelta – Background activity duration value.

Protected Functions

```
inline bool doBackgroundActivityLookup_unsafe (const int16_t x, int16_t y, const int64_t timestamp)
```

```
inline bool doBackgroundActivityLookup (const int16_t x, int16_t y, const int64_t timestamp)
```

Protected Attributes

```
cv::Size mResolutionLimits
```

```
dv::TimeSurface mTimeSurface
```

```
int64_t mBackgroundActivityDuration = 2000
```

```
struct BadAlloc : public dv::exceptions::info::EmptyException

template<dv::concepts::EventStorage EventStoreClass = dv::EventStore>

class BandCutFilter : public dv::noise::BaseFrequencyFilter<dv::EventStore>
```

Public Functions

```
inline explicit BandCutFilter (const cv::Size &resolution, const float lowerCutOffFrequency, const float
                                upperCutOffFrequency)
```

A band-cut event frequency filter. Discards events at a pixel location with a frequency inside a given frequency band defined by a lower cutoff frequency and an upper cutoff frequency.

Parameters

- **resolution** – Sensor resolution.
- **lowerCutOffFrequency** – Lower filter cutoff frequency. Together with #highCutOffFrequency, defines the frequency band for the band-pass filter. All events with a frequency outside of this band are discarded.
- **upperCutOffFrequency** – Upper filter cutoff frequency. Together with #lowCutOffFrequency, defines the frequency band for the band-pass filter. All events with a frequency outside of this band are discarded.

```
inline BandCutFilter &operator<< (const EventStoreClass &events)
```

Accept events using the input stream operator.

Parameters

events – Input events.

Returns

```
inline float getLowerCutOffFrequency () const
```

Get the lower cutoff frequency of the frequency band.

Returns

Currently configured low cutoff frequency of the frequency band.

```
inline float getUpperCutOffFrequency () const
```

Get the upper cutoff frequency of the frequency band.

Returns

Currently configured upper cutoff frequency of the frequency band.

```
inline void setLowerCutOffFrequency (const float frequency)
```

Set a new value for the lower cutoff frequency of the frequency band.

Parameters

frequency – New lower cutoff frequency value.

```
inline void setUpperCutOffFrequency (const float frequency)
```

Set a new value for the upper cutoff frequency of the frequency band.

Parameters

frequency – New upper cutoff frequency value.

```
template<dv::concepts::EventStorage EventStoreClass = dv::EventStore>
```

```
class BandPassFilter : public dv::noise::BaseFrequencyFilter<dv::EventStore>
```

#include </builds/inivation/dv/dv-processing/include/dv-processing/noise/frequency_filters.hpp> A band-pass event frequency filter. Discards events at a pixel location with a frequency outside a given frequency band defined by a lower cutoff frequency and an upper cutoff frequency.

Template Parameters

EventStoreClass – Type of event store.

Public Functions

```
inline explicit BandPassFilter (const cv::Size &resolution, const float lowerCutOffFrequency, const float  
                                upperCutOffFrequency)
```

A band-pass event frequency filter. Discards events at a pixel location with a frequency outside a given frequency band defined by a lower cutoff frequency and an upper cutoff frequency.

Parameters

- **resolution** – Sensor resolution.
- **lowerCutOffFrequency** – Lower filter cutoff frequency. Together with **highCutOffFrequency**, defines the frequency band for the band-pass filter. All events with a frequency outside of this band are discarded.
- **upperCutOffFrequency** – Upper filter cutoff frequency. Together with **lowCutOffFrequency**, defines the frequency band for the band-pass filter. All events with a frequency outside of this band are discarded.

```
inline BandPassFilter &operator<< (const EventStoreClass &events)
```

Accept events using the input stream operator.

Parameters

events – Input events.

Returns

```
inline float getLowerCutOffFrequency () const
```

Get the lower cutoff frequency of the frequency band.

Returns

Currently configured low cutoff frequency of the frequency band.

```
inline float getUpperCutOffFrequency () const
```

Get the upper cutoff frequency of the frequency band.

Returns

Currently configured upper cutoff frequency of the frequency band.

```
inline void setLowerCutOffFrequency (const float frequency)
```

Set a new value for the lower cutoff frequency of the frequency band.

Parameters

frequency – New lower cutoff frequency value.

```
inline void setUpperCutOffFrequency (const float frequency)
```

Set a new value for the upper cutoff frequency of the frequency band.

Parameters

frequency – New upper cutoff frequency value.

```
template<dv::concepts::EventStorage EventStoreClass = dv::EventStore>
```

class **BaseFrequencyFilter** : public *dv::EventFilterBase*<*dv::EventStore*>

#include </builds/inivation/dv/dv-processing/include/dv-processing/noise/frequency_filters.hpp> A base class for basic frequency filters. Different frequency filters (low-pass, high-pass, band-pass, etc.) are derived as special cases of this base filter. Handles data input and output, as well as checking if an event should be retained. Derived classes only have to define the required behavior by passing correct arguments to the constructor.

Template Parameters

EventStoreClass – Type of event store.

Subclassed by *dv::noise::BandCutFilter*< *EventStoreClass* >, *dv::noise::BandPassFilter*< *EventStoreClass* >, *dv::noise::HighPassFilter*< *EventStoreClass* >, *dv::noise::LowPassFilter*< *EventStoreClass* >

Public Functions

inline explicit **BaseFrequencyFilter** (const cv::Size &resolution, const *std::optional*<float> lowerCutOffFrequency, const *std::optional*<float> upperCutOffFrequency, const *FrequencyFilterType* filterType)

Construct a base event frequency filter for filtering events based on defined thresholds for the upper/lower cutoff frequencies. Different filter behaviors can be implemented by passing different arguments to the constructor (low-pass, high-pass, band-pass, etc.)

Parameters

- **resolution** – Sensor resolution.
- **lowerCutOffFrequency** – Lower cutoff frequency for filter. Ignored if *std::nullopt*
- **upperCutOffFrequency** – Upper cutoff frequency for filter. Ignored if *std::nullopt*
- **filterType** – Filter type, whether the filter behaves as a cut or a pass for the given cutoff frequencies

inline virtual bool **retain** (const typename *EventStoreClass::value_type* &event) noexcept override

Test whether the frequency of an event is less than the cutoff frequency.

Parameters

event – Event to be tested.

Returns

True - the frequency of the tested event at that pixel location is less than the cutoff frequency, false otherwise.

Protected Attributes

dv::TimeSurface **mTimeSurface**

std::optional<int64_t> **mLowCutOffPeriod**

std::optional<int64_t> **mHighCutOffPeriod**

FrequencyFilterType **mFilterType**

struct **BoundingBox** : public *flatbuffers::NativeTable*

Public Types

```
typedef BoundingBoxFlatbuffer TableType
```

Public Functions

```
inline BoundingBox ()
```

```
inline BoundingBox (int64_t _timestamp, float _topLeftX, float _topLeftY, float _bottomRightX, float  
_bottomRightY, float _confidence, const std::string &_label)
```

Public Members

```
int64_t timestamp
```

```
float topLeftX
```

```
float topLeftY
```

```
float bottomRightX
```

```
float bottomRightY
```

```
float confidence
```

```
std::string label
```

Public Static Functions

```
static inline constexpr const char *GetFullyQualifiedName ()
```

```
struct BoundingBoxBuilder
```

Public Functions

```
inline void add_timestamp (int64_t timestamp)
```

```
inline void add_topLeftX (float topLeftX)
```

```
inline void add_topLeftY (float topLeftY)
```

```
inline void add_bottomRightX (float bottomRightX)
```

```
inline void add_bottomRightY (float bottomRightY)
```

```
inline void add_confidence (float confidence)
```

```
inline void add_label (flatbuffers::Offset<flatbuffers::String> label)
```

```
inline explicit BoundingBoxBuilder (flatbuffers::FlatBufferBuilder &_fbb)
```

```
BoundingBoxBuilder &operator= (const BoundingBoxBuilder&)
```

```
inline flatbuffers::Offset<BoundingBoxFlatbuffer> Finish ()
```

Public Members

```
flatbuffers::FlatBufferBuilder &fbb_
```

```
flatbuffers::uoffset_t start_
```

```
struct BoundingBoxFlatbuffer : private flatbuffers::Table
```

Public Types

```
typedef BoundingBox NativeTableType
```

Public Functions

```
inline int64_t timestamp () const
```

Timestamp (µs).

```
inline float topLeftX () const
```

top left corner of bounding box x-coordinate.

```
inline float topLeftY () const
```

top left corner of bounding box y-coordinate.

```
inline float bottomRightX () const
```

bottom right corner of bounding box x-coordinate.

```
inline float bottomRightY () const
```

bottom right corner of bounding box y-coordinate.

```
inline float confidence () const
```

confidence of the given bounding box.

```
inline const flatbuffers::String *label () const
```

Label for the given bounding box.

```
inline bool Verify (flatbuffers::Verifier &verifier) const
```

```
inline BoundingBox *UnPack (const flatbuffers::resolver_function_t *_resolver = nullptr) const
```

```
inline void UnPackTo (BoundingBox *_o, const flatbuffers::resolver_function_t *_resolver = nullptr) const
```

Public Static Functions

```
static inline const flatbuffers::TypeTable *MiniReflectTypeTable ()
```

```
static inline constexpr const char *GetFullyQualifiedName ()
```

```
static inline void UnPackToFrom (BoundingBox *_o, const BoundingBoxFlatbuffer *_fb, const  
flatbuffers::resolver_function_t *_resolver = nullptr)
```

```
static inline flatbuffers::Offset<BoundingBoxFlatbuffer> Pack (flatbuffers::FlatBufferBuilder &_fbb, const  
                                                         BoundingBox *_o, const  
                                                         flatbuffers::rehasher_function_t *_rehasher =  
                                                         nullptr)
```

```
struct BoundingBoxPacket : public flatbuffers::NativeTable
```

Public Types

```
typedef BoundingBoxPacketFlatbuffer TableType
```

Public Functions

```
inline BoundingBoxPacket ()
```

```
inline BoundingBoxPacket (const std::vector<BoundingBox> &_elements)
```

Public Members

```
std::vector<BoundingBox> elements
```

Public Static Functions

```
static inline constexpr const char *GetFullyQualifiedName ()
```

Friends

```
inline friend std::ostream &operator<< (std::ostream &os, const BoundingBoxPacket &packet)
```

```
struct BoundingBoxPacketBuilder
```

Public Functions

```
inline void add_elements (flatbuffers::Offset<flatbuffers::Vector<flatbuffers::Offset<BoundingBoxFlatbuffer>>>  
                        elements)
```

```
inline explicit BoundingBoxPacketBuilder (flatbuffers::FlatBufferBuilder &_fbb)
```

```
BoundingBoxPacketBuilder &operator= (const BoundingBoxPacketBuilder &)
```

```
inline flatbuffers::Offset<BoundingBoxPacketFlatbuffer> Finish ()
```

Public Members

```
flatbuffers::FlatBufferBuilder &fbb_
```

```
flatbuffers::uoffset_t start_
```

```
struct BoundingBoxPacketFlatbuffer : private flatbuffers::Table
```

Public Types

```
typedef BoundingBoxPacket NativeTableType
```

Public Functions

```
inline const flatbuffers::Vector<flatbuffers::Offset<BoundingBoxFlatbuffer>> *elements () const
```

```
inline bool Verify (flatbuffers::Verifier &verifier) const
```

```
inline BoundingBoxPacket *UnPack (const flatbuffers::resolver_function_t *_resolver = nullptr) const
```

```
inline void UnPackTo (BoundingBoxPacket *_o, const flatbuffers::resolver_function_t *_resolver = nullptr) const
```

Public Static Functions

```
static inline const flatbuffers::TypeTable *MiniReflectTypeTable ()
```

```
static inline constexpr const char *GetFullyQualifiedName ()
```

```
static inline void UnPackToFrom (BoundingBoxPacket *_o, const BoundingBoxPacketFlatbuffer *_fb, const  
                                flatbuffers::resolver_function_t *_resolver = nullptr)
```

```
static inline flatbuffers::Offset<BoundingBoxPacketFlatbuffer> Pack (flatbuffers::FlatBufferBuilder &_fbb, const  
                                BoundingBoxPacket *_o, const  
                                flatbuffers::rehasher_function_t *_rehasher  
                                = nullptr)
```

Public Static Attributes

```
static constexpr const char *identifier = "BBOX"
```

class **CalibrationSet**

#include </builds/inivation/dv/dv-processing/include/dv-processing/camera/calibration_set.hpp> *CalibrationSet*
class is used to store, serialize and deserialize various camera related calibrations - intrinsic, extrinsic, *IMU* calibrations. Supports multi-camera and multi sensor setups.

Each calibration for each sensor received a designation string which consist of a letter determining the type of sensor and a numeric index automatically generated for each sensor. Designation string look like this: "C0" - camera with index 0 "S0" - *IMU* sensor with index 0 "C0C1" - stereo calibration where C0 is the left camera and C1 is the right camera in the camera rig setup.

Designation indexes are automatically incremented by the order they are added to the calibration set.

Public Types

```
using CameraCalibrationMap = std::map<std::string, calibrations::CameraCalibration, std::less<>>
```

```
using IMUCalibrationMap = std::map<std::string, calibrations::IMUCalibration, std::less<>>
```

```
using StereoCalibrationMap = std::map<std::string, calibrations::StereoCalibration, std::less<>>
```

Public Functions

CalibrationSet () = default

inline boost::property_tree::ptree **toPropertyTree** () const

Serialize calibration data into a property tree that can be saved into a file using boost::property_tree::write_json or other property_tree serialization method.

Returns

Property tree containing calibration data.

inline std::vector<std::string> **getCameraList** () const

Get a list of cameras available by their designation.

Returns

Vector of available camera designations.

inline std::vector<std::string> **getImuList** () const

Get a list camera designations which have imu calibrations available in this calibration set.

Returns

Vector of available imu designations.

inline std::vector<std::string> **getStereoList** () const

Get a list of designations of stereo calibrations available here.

Returns

Vector of available stereo calibrations designations.

inline std::optional<calibrations::CameraCalibration> **getCameraCalibration** (const std::string_view designation) const

Retrieve a camera calibration by designation (e.g. "C0").

Designation string consists of a letter determining the type of sensor and a numeric index automatically generated for each sensor. Designation string look like this: "C0" - camera with index 0 "S0" - *IMU* sensor with index 0 "C0C1" - stereo calibration where C0 is the left camera and C1 is the right camera in the camera rig setup.

Parameters

designation – Camera designation string.

Returns

Camera intrinsics calibration, std::nullopt if given designation is not found.

inline std::optional<calibrations::IMUCalibration> **getImuCalibration** (const std::string_view designation) const

Get *IMU* calibration by *IMU* sensor designation (e.g. "S0").

Designation string consists of a letter determining the type of sensor and a numeric index automatically generated for each sensor. Designation string look like this: "C0" - camera with index 0 "S0" - *IMU* sensor with index 0 "C0C1" - stereo calibration where C0 is the left camera and C1 is the right camera in the camera rig setup.

Parameters

designation – *IMU* designation string.

Returns

IMU extrinsic calibration, std::nullopt if given designation is not found.

```
inline std::optional<calibrations::StereoCalibration> getStereoCalibration (const std::string_view
                                                                    designation) const
```

Get stereo calibration by stereo rig designation (e.g. “C0C1”).Retrieve the full list of *IMU* extrinsic calibrations.

Designation string consists of a letter determining the type of sensor and a numeric index automatically generated for each sensor. Designation string look like this: “C0” - camera with index 0 “S0” - *IMU* sensor with index 0 “C0C1” - stereo calibration where C0 is the left camera and C1 is the right camera in the camera rig setup.

Parameters

designation – Stereo rig designation string.

Returns

Stereo extrinsic calibration, `std::nullopt` if given designation is not found.

```
inline std::optional<calibrations::CameraCalibration> getCameraCalibrationByName (const std::string_view
                                                                    camera) const
```

Retrieve a camera calibration by camera name, which consist of model and serial number concatenation with an underscore separator (e.g. “DVXplorer_DXA00000”).

Camera name is usually available in recording files and when connected directly to a camera.

Parameters

camera – Name of the camera.

Returns

Camera intrinsic calibration, `std::nullopt` if given camera name is not found.

```
inline std::optional<calibrations::IMUCalibration> getImuCalibrationByName (const std::string_view
                                                                    camera) const
```

Retrieve an *IMU* calibration by camera name, which consist of model and serial number concatenation with an underscore separator (e.g. “DVXplorer_DXA00000”).

Camera name is usually available in recording files and when connected directly to a camera.

Parameters

camera – Name of the camera.

Returns

IMU extrinsics calibration, `std::nullopt` if given camera name is not found.

```
inline std::optional<calibrations::StereoCalibration> getStereoCalibrationByLeftCameraName (const
                                                                    std::string_view
                                                                    camera)
                                                                    const
```

Retrieve a stereo calibration by matching camera name to left camera name in the stereo calibrations. Camera name consist of model and serial number concatenation with an underscore separator (e.g. “DVXplorer_DXA00000”).

Camera name is usually available in recording files and when connected directly to a camera.

Parameters

camera – Name of the camera.

Returns

Stereo extrinsic calibration, `std::nullopt` if given camera name is not found.

```
inline std::optional<calibrations::StereoCalibration> getStereoCalibrationByRightCameraName (const  
                                                                                          std::string_view  
                                                                                          camera)  
                                                                                          const
```

Retrieve a stereo calibration by matching camera name to right camera name in the stereo calibrations. Camera name consist of model and serial number concatenation with an underscore separator (e.g. “DVX-plorer_DXA00000”).

Camera name is usually available in recording files and when connected directly to a camera.

Parameters

camera – Name of the camera.

Returns

Stereo extrinsic calibration, `std::nullopt` if given camera name is not found.

```
inline void updateCameraCalibration (const calibrations::CameraCalibration &calibration)
```

Update Camera calibration for the given camera name.

Parameters

calibration – Camera calibration instance.

```
inline void updateImuCalibration (const calibrations::IMUCalibration &calibration)
```

Update *IMU* calibration for the camera name.

Parameters

calibration – *IMU* calibration instance.

```
inline void updateStereoCameraCalibration (const calibrations::StereoCalibration &calibration)
```

Update Stereo Camera calibration for the given camera name.

Parameters

calibration – Stereo calibration instance.

```
inline void addCameraCalibration (const calibrations::CameraCalibration &calibration)
```

Add an intrinsic calibration to the camera calibration set. Camera designation is going to be generated automatically.

Parameters

calibration – Camera intrinsics calibration.

```
inline void addImuCalibration (const calibrations::IMUCalibration &calibration)
```

Add an *IMU* extrinsics calibration to the calibration set.

Parameters

calibration – *IMU* extrinsic calibration.

```
inline void addStereoCalibration (const calibrations::StereoCalibration &calibration)
```

Add a stereo calibration to the calibration set. Intrinsic calibrations of the sensors should already be added using `addCameraCalibration` prior to adding the stereo extrinsic calibration.

Parameters

calibration – Stereo calibration.

Throws

Throws – an invalid argument exception if the intrinsic calibration of given camera sensors are not available in the set or stereo calibration for the given cameras already exist/

```
inline const CameraCalibrationMap &getCameraCalibrations () const
```

Retrieve the full list of camera intrinsic calibrations.

Returns

`std::map` containing camera calibrations where keys are camera designation strings.

inline const *IMUCalibrationMap* &**getImuCalibrations** () const

Retrieve the full list of *IMU* extrinsic calibrations.

Returns

`std::map` containing *IMU* calibrations where keys are *IMU* sensor designation strings.

inline const *StereoCalibrationMap* &**getStereoCalibrations** () const

Retrieve the full list of stereo extrinsic calibrations.

Returns

`std::map` containing stereo calibrations where keys are stereo rig camera designation strings.

inline void **writeToFile** (const *std::filesystem::path* &outputFile) const

Write the contents of this calibration set into a file at given path.

This function requires that supplied path contains “.json” extension.

Parameters

outputFile – Output file path with “.json” extension to write the contents of the calibration set.

Public Static Functions

static inline *CalibrationSet* **LoadFromFile** (const *std::filesystem::path* &path)

Create a calibration file representation from a persistent file. Supports legacy “.xml” calibration files produced by DV as well as JSON files containing calibration of a new format.

The file format is distinguished using the file path extension.

Parameters

path – Path to calibration file.

Returns

CalibrationFile instanced containing parsed calibration values.

Private Functions

inline explicit **CalibrationSet** (const boost::property_tree::ptree &tree)

Private Members

size_t **cameraIndex** = 0

size_t **imuIndex** = 0

CameraCalibrationMap **cameras**

IMUCalibrationMap **imus**

StereoCalibrationMap **stereo**

Private Static Functions

```
static inline CalibrationSet cameraRigCalibrationFromJsonFile (const std::filesystem::path &path)

static inline calibrations::CameraCalibration oneCameraCalibrationFromXML (const cv::FileNode &node,
                                                                           const std::string_view
                                                                           cameraName, const bool
                                                                           cameraIsMaster)

static inline CalibrationSet cameraRigCalibrationFromXmlFile (const std::filesystem::path &path)
```

```
struct CameraCalibration
```

Public Functions

CameraCalibration () = default

```
inline explicit CameraCalibration (const std::string_view name_, const std::string_view position_, const bool
                                     master_, const cv::Size &resolution_, const cv::Point2f &principalPoint_,
                                     const cv::Point2f &focalLength_, const std::span<const float> distortion_,
                                     const DistortionModel distortionModel_, const
                                     dv::kinematics::Transformationf &transformationToC0_, const
                                     std::optional<Metadata> &metadata_)
```

Construct the camera calibration

Parameters

- **name_** – Camera name (e.g. “DVXplorer_DXA02137”)
- **position_** – Description of the location of the camera in the camera rig (e.g. “left”)
- **master_** – Whether camera was a master camera during calibration
- **resolution_** – Camera resolution
- **principalPoint_** – Principal point
- **focalLength_** – Focal length
- **distortion_** – Distortion coefficients
- **distortionModel_** – Distortion model used (can be empty string or “radialTangential”)
- **transformationToC0_** – Transformation from this camera to camera zero
- **metadata_** – *Metadata*

```
inline explicit CameraCalibration (const boost::property_tree::ptree &tree)
```

Parse a property tree and initialize camera calibration out of it.

Parameters

tree – Serialized property tree containing camera intrinsics calibration.

```
inline boost::property_tree::ptree toPropertyTree () const
```

Serialize the *CameraCalibration* structure into a property tree.

Returns

Serialized property tree.

inline bool **operator==** (const *CameraCalibration* &rhs) const

Equality operator for the class, compares each member of the class.

Parameters

rhs – Other instance of this class

Returns

inline cv::Matx33f **getCameraMatrix** () const

Get camera matrix in the format: | mFx 0 mCx || 0 mFy mCy || 0 0 1 | for direct OpenCV compatibility.

Returns

3x3 Camera matrix with pixel length values

inline *dv::camera::CameraGeometry* **getCameraGeometry** () const

Retrieve camera geometry instance from this calibration instance. Distortion model is going to be ignored if the *CameraGeometry* class doesn't support the distortion model.

CameraGeometry class only supports “radialTangential” distortion model.

Returns

Camera geometry class that implements geometrical transformations of pixel coordinates.

inline *std::string* **getDistortionModelString** () const

Get distortion model name as a string.

Returns

Distortion model name.

Public Members

std::string **name**

Camera name (e.g. “DVXplorer_DXA02137”)

std::string **position**

Description of the location of the camera in the camera rig (e.g. “left”)

bool **master** = false

Indicate whether it is the master camera in a multi-camera rig.

cv::Size **resolution**

Camera resolution width.

cv::Point2f **principalPoint**

Intersection of optical axis and image plane.

cv::Point2f **focalLength**

Focal length.

std::vector<float> **distortion**

Distortion coefficients.

DistortionModel **distortionModel** = *DistortionModel::RADIAL_TANGENTIAL*

Distortion model used.

dv::kinematics::Transformationf **transformationToC0**

Transformation from camera zero to this camera.

std::optional<Metadata> **metadata**

Metadata.

Protected Static Functions

template<typename T>

static inline void **pushVectorToTree** (const *std::string* &key, const *std::vector<T>* &vals,
boost::property_tree::ptree &tree)

Push a vector of the given type to the property tree at the given key.

Template Parameters

T – Datatype of vector

Parameters

- **key** – Key in property tree where data will be added
- **transform** – Transform to add to property tree
- **tree** – Property tree to add data to

static inline void **pushTransformToTree** (const *std::string* &key, const *dv::kinematics::Transformationf*
&transform, boost::property_tree::ptree &tree)

Push kinematics transformation to the property tree at the given key.

Parameters

- **key** – Key in property tree where data will be added
- **transform** – Transform to add to property tree
- **tree** – Property tree to add data to

template<typename T>

static inline *std::vector<T>* **getVectorFromTree** (const *std::string* &key, const boost::property_tree::ptree
&tree)

Retrieve a vector of the given type from the property tree from the given key.

Template Parameters

T – Datatype of vector

Parameters

- **key** – Key in property tree where to get data from
- **tree** – Property tree to get data from

Returns

A sequence value in a *std::vector* container.

static inline *dv::kinematics::Transformationf* **getTransformFromTree** (const *std::string* &key, const
boost::property_tree::ptree &tree)

Retrieve a kinematics transformation from the property tree from the given key.

Parameters

- **key** – Key where kinematics transform is stored in property tree
- **tree** – Property tree to get data from

Returns

A kinematics transform from the property tree.

```
template<class Container, typename Scalar>
static inline Container parsePair (const boost::property_tree::ptree &child, const std::string &name,
                                   std::optional<Scalar> defaults = std::nullopt)

template<class Container, typename Scalar>
static inline Container parseTripple (const boost::property_tree::ptree &child, const std::string &name,
                                       std::optional<Scalar> defaults = std::nullopt)

template<class MetadataClass>
static inline std::optional<MetadataClass> getOptionalMetadata (const boost::property_tree::ptree &tree,
                                                             const std::string &path)
```

Friends

```
friend struct IMUCalibration
```

```
friend struct StereoCalibration
```

```
inline friend std::ostream &operator<< (std::ostream &os, const CameraCalibration &calibration)
    Serialize the object into a stream.
```

Parameters

- **os** –
- **calibration** –

Returns

```
class CameraGeometry
```

Public Types

```
enum class FunctionImplementation
```

Values:

```
enumerator LUT
```

```
enumerator SUB_PIXEL
```

```
using SharedPtr = std::shared_ptr<CameraGeometry>
```

```
using UniquePtr = std::unique_ptr<CameraGeometry>
```

Public Functions

inline **CameraGeometry** (const *std::span*<const float> distortion, const float fx, const float fy, const float cx, const float cy, const cv::Size &resolution, const *DistortionModel* distortionModel)

Create a camera geometry model with distortion model. Currently only radial tangential model is supported.

Parameters

- **distortion** – Distortion coefficient (4 or 5 coefficient radtan model).
- **fx** – Focal length X measured in pixels.
- **fy** – Focal length Y measured in pixels.
- **cx** – Central point coordinate X in pixels.
- **cy** – Central point coordinate Y in pixels.
- **resolution** – Sensor resolution.

inline **CameraGeometry** (const float fx, const float fy, const float cx, const float cy, const cv::Size &resolution)

Create a camera geometry model without distortion model. Currently only radial tangential model is supported.

Any calls to function dependent on distortion will cause exceptions or segfaults.

Parameters

- **fx** – Focal length X measured in pixels.
- **fy** – Focal length Y measured in pixels.
- **cx** – Central point coordinate X in pixels.
- **cy** – Central point coordinate Y in pixels.
- **resolution** – Sensor resolution.

template<*concepts::Coordinate2DConstructible* **Output**, *concepts::Coordinate2D* **Input**>

inline *Output* **undistort** (const *Input* &point) const

Returns pixel coordinates of given point with applied back projection, undistortion, and projection. This function uses look-up table and is designed for minimal execution speed.

WARNING: will cause a segfault if coordinates are out-of-bounds or if distortion model is not available.

Parameters

point – Pixel coordinate

Returns

Undistorted pixel coordinate

inline *dv::EventStore* **undistortEvents** (const *dv::EventStore* &events) const

Undistort event coordinates, discards events which fall beyond camera resolution.

Parameters

events – Input events

Returns

A new event store containing the same events with undistorted coordinates

template<*concepts::Coordinate2DMutableIterable* **Output**, *concepts::Coordinate2DIterable* **Input**>

inline *Output* **undistortSequence** (const *Input* &coordinates) const

Undistort point coordinates.

Parameters

coordinates – Input point coordinates

Returns

A new vector containing the points with undistorted coordinates

```
template<concepts::Coordinate3DConstructible Output, concepts::Coordinate3D Input>
```

```
inline Output distort (const Input &undistortedPoint) const
```

Apply distortion to a 3D point.

Parameters

point – Point in 3D space

Returns

Distorted point

```
template<concepts::Coordinate3DMutableIterable Output, concepts::Coordinate3DIterable Input>
```

```
inline Output distortSequence (const Input &points) const
```

Apply direct distortion on the 3D points.

Parameters

points – Input points

Returns

Distorted points

```
template<concepts::Coordinate3DConstructible Output, concepts::Coordinate2D Input, FunctionImplementation
```

```
implementation = FunctionImplementation::LUT>
```

```
inline Output backProject (const Input &pixel) const
```

Back-project pixel coordinates into a unit ray vector of depth = 1.0 meters.

Parameters

pixel – Pixel to be projected

Template Parameters

implementation – Specify the internal implementation to perform the computations, Sub-Pixel performs all computations without any optimization, LUT option avoids computation by performing a look-up table operation instead, but rounds input coordinate values.

Returns

Back projected unit ray

```
template<concepts::Coordinate3DMutableIterable Output, concepts::Coordinate2DIterable Input,
```

```
FunctionImplementation implementation = FunctionImplementation::LUT>
```

```
inline Output backProjectSequence (const Input &points) const
```

Back project a sequence of 2D point into 3D unit ray-vectors.

Parameters

points – Input points.

Template Parameters

implementation – Specify the internal implementation to perform the computations, Sub-Pixel performs all computations without any optimization, LUT option avoids computation by performing a look-up table operation instead, but rounds input coordinate values.

Returns

A sequence of back-projected unit ray vectors.

```
template<concepts::Coordinate3DConstructible Output, concepts::Coordinate2D Input>
```

inline *Output* **backProjectUndistort** (const *Input* &pixel) const

Returns a unit ray of given coordinates with applied back projection and undistortion. This function uses look-up table and is designed for minimal execution speed.

WARNING: will cause a segfault if coordinates are out-of-bounds or if distortion model is not available.

Parameters

pixel – Pixel coordinate

Returns

Back projected and undistorted unit ray

template<concepts::Coordinate3DMutableIterable *Output*, concepts::Coordinate2DIterable *Input*>

inline *Output* **backProjectUndistortSequence** (const *Input* &points) const

Undistort and back project a batch of points. Output is normalized point coordinates as unit rays.

Parameters

points – Input points.

Returns

Undistorted and back projected points.

template<concepts::Coordinate2DConstructible *Output*, concepts::Coordinate3D *Input*>

inline *Output* **project** (const *Input* &points) const

Project a 3D point into pixel plane.

WARNING: Does not perform range checking!

Parameters

points – 3D points to be projected

Returns

Projected pixel coordinates

template<concepts::Coordinate2DMutableIterable *Output*, concepts::Coordinate3DIterable *Input*>

inline *Output* **projectSequence** (const *Input* &points, const bool dimensionCheck = true) const

Project a batch of 3D points into pixel plane.

Parameters

- **points** – Points to be projected.
- **dimensionCheck** – Whether to perform resolution check, if true, output points outside of valid frame resolution will be omitted. If disabled, output point count and order will be the same as input points.

Returns

Projected points in pixel plane.

template<concepts::Coordinate2D *Input*>

inline bool **isWithinDimensions** (const *Input* &point) const

Check whether given coordinates are within valid range.

Parameters

point – Pixel coordinates

Returns

True if the coordinate values are within camera resolution, false otherwise.

inline bool **isUndistortionAvailable** () const

Checks whether this camera geometry calibration contains coefficient for an undistortion model.

Returns

True if undistortion is available, false otherwise

```
inline cv::Matx33f getCameraMatrix() const
```

Get camera matrix in the format: | mFx 0 mCx || 0 mFy mCy || 0 0 1 |

Returns

3x3 Camera matrix with pixel length values

```
template<concepts::Coordinate2DConstructible Output = cv::Point2f>
```

```
inline Output getFocalLength() const
```

Focal length

Returns

Focal length in pixels

```
template<concepts::Coordinate2DConstructible Output = cv::Point2f>
```

```
inline Output getCentralPoint() const
```

Central point coordinates

Returns

Central point coordinates in pixels

```
inline std::vector<float> getDistortion() const
```

Get distortion coefficients

Returns

Vector containing distortion coefficients

```
inline DistortionModel getDistortionModel() const
```

Get distortion model

Returns

DistortionModel type

```
inline cv::Size getResolution() const
```

Get the camera resolution.

Returns

Camera sensor resolution

Private Functions

```
inline void generateLUTs()
```

Generates internal distortion look-up table to speed up undistortion.

```
template<concepts::Coordinate3DConstructible Output, concepts::Coordinate3D Input>
```

```
inline Output distortRadialTangential(const Input &point) const
```

Distort the Input point according to the Radial Tangential distortion model.

Template Parameters

- **Output** –
- **Input** –

Parameters

point –

Returns

the distorted point in the 3D space

```
template<concepts::Coordinate3DConstructible Output, concepts::Coordinate3D Input>
inline Output distortEquidistant (const Input &point) const
```

Distort the Input point according to the Equidistant distortion model.

Template Parameters

- **Output** –
- **Input** –

Parameters

point –

Returns

the distorted point in the 3D space

Private Members

```
std::vector<cv::Point3f> mDistortionLUT
```

Row-based distortion look-up table. Access index by: $\text{index} = (y * \text{width}) + x$

```
std::vector<cv::Point3f> mBackProjectLUT
```

Row-based distortion look-up table. Access index by: $\text{index} = (y * \text{width}) + x$

```
std::vector<cv::Point2f> mDistortionPixelLUT
```

Row-based undistorted coordinate look-up table, containing undistorted points in pixel space. Access index by: $\text{index} = (y * \text{width}) + x$

```
std::vector<float> mDistortion
```

Distortion coefficients

```
float mFx
```

Focal length on x axis in pixels

```
float mFy
```

Focal length on y axis in pixels

```
float mCx
```

Central point coordinates on x axis

```
float mCy
```

Central point coordinates on x axis

```
cv::Size mResolution
```

Sensor resolution

```
float mMaxX
```

Max floating point coordinate x address value

float **mMaxY**

Max floating point coordinate y address value

DistortionModel **mDistortionModel**

Distortion model used

Friends

inline friend *std::ostream* &**operator<<** (*std::ostream* &os, const *FunctionImplementation* &var)

class **CameraInputBase** : public *dv::io::InputBase*

#include </builds/inivation/dv/dv-processing/include/dv-processing/io/camera/camera_input_base.hpp> Camera input base class to abstract live camera and recorded files with a common interface.

Subclassed by *dv::io::camera::DVXplorerM*, *dv::io::camera::SyncCameraInputBase*

Public Types

enum class **Flatten**

Event flattening modes, used to change polarity of events returned by the camera.

Values:

enumerator **NONE**

No change.

enumerator **FLATTEN_ON**

All events become ON events.

enumerator **ON_ONLY**

enumerator **OFF_ONLY**

Public Functions

virtual bool **getFlipHorizontal** () const = 0

Status of horizontal events flip.

Returns

status of horizontal events flip.

virtual void **setFlipHorizontal** (bool flipHorizontalEvents) = 0

Flip events horizontally.

Parameters

flipHorizontalEvents – flip events horizontally.

virtual bool **getFlipVertical** () const = 0

Status of vertical events flip.

Returns

status of vertical events flip.

virtual void **setFlipVertical** (bool flipVerticalEvents) = 0

Flip events vertically.

Parameters

flipVerticalEvents – flip events vertically.

virtual *Flatten* **getFlatten** () const = 0

Status of event polarity flattening.

Returns

status of event polarity flattening.

virtual void **setFlatten** (*Flatten* flattenEvents) = 0

Flatten events polarity.

Parameters

flattenEvents – flattening mode.

virtual cv::Rect **getCropArea** () const = 0

Get events Region of Interest (ROI).

Returns

get events Region of Interest (ROI).

virtual void **setCropArea** (cv::Rect cropAreaEvents) = 0

Set events Region of Interest (ROI). Usually hardware accelerated.

Parameters

cropAreaEvents – region of interest (ROI) position and size.

virtual *imu::ImuModel* **getImuModel** () const = 0

Return *IMU* model used on device.

Returns

IMU model in use.

virtual float **getPixelPitch** () const = 0

Return pixel pitch distance for the connected camera model. The value is returned in meters.

Returns

Pixel pitch distance in meters according to the connected device.

virtual *std::chrono::microseconds* **getTimeInterval** () const = 0

Get the time interval for data commit.

Returns

Time interval in microseconds.

virtual void **setTimeInterval** (*std::chrono::microseconds* timeInterval) = 0

Set a new time interval value for data commit. Data is put in the queues for *getNextEventBatch()*, *readNext()*, ... at this interval's rate.

Parameters

timeInterval – New time interval value in microseconds.

virtual *std::chrono::microseconds* **getTimestampOffset** () const = 0

Get the timestamp offset.

Returns

Absolute timestamp offset value in microseconds.

inline *std::optional<dv::EventPacket>* getNextEventPacket ()

Parse and retrieve next event packet (internal format).

Returns

Event packet or *std::nullopt* if no events were received since last read.

inline virtual *std::optional<dv::EventStore>* getNextEventBatch () override

Parse and retrieve next event batch.

Returns

Event batch or *std::nullopt* if no events were received since last read.

inline virtual *std::optional<dv::Frame>* getNextFrame () override

Parse and retrieve next frame.

Returns

Frame or *std::nullopt* if no frames were received since last read.

inline virtual *std::optional<std::vector<dv::IMU>>* getNextImuBatch () override

Parse and retrieve next *IMU* data batch.

Returns

IMU data batch or *std::nullopt* if no *IMU* data was received since last read.

inline virtual *std::optional<std::vector<dv::Trigger>>* getNextTriggerBatch () override

Parse and retrieve next trigger data batch.

Returns

Trigger data batch or *std::nullopt* if no triggers were received since last read.

inline virtual bool isStreamAvailable (const *std::string_view* streamName) const override

Check whether a stream with given name is available.

Returns

True if data stream is available, false otherwise.

inline int64_t getEventSeekTime () const

Get latest timestamp of event data stream that has been read from the capture class.

Returns

Latest processed event timestamp; returns -1 if no data was processed or stream is unavailable.

inline int64_t getFrameSeekTime () const

Get latest timestamp of frames stream that has been read from the capture class.

Returns

Latest processed frame timestamp; returns -1 if no data was processed or stream is unavailable.

inline int64_t getImuSeekTime () const

Get latest timestamp of imu data that has been read from the capture class.

Returns

Latest processed imu data timestamp; returns -1 if no data was processed or stream is unavailable.

inline int64_t getTriggerSeekTime () const

Get latest timestamp of trigger data stream that has been read from the capture class.

Returns

Latest processed trigger timestamp; returns -1 if no data was processed or stream is unavailable.

inline *DataReadVariant* **readNext** ()

Read a packet from the camera and return a variant of any packet. You can use `std::visit` with *dv::io::DataReadHandler* to handle each type of packet using callback methods. This method might not maintain timestamp monotonicity between different stream types.

Returns

A variant containing data packet from the camera.

inline bool **handleNext** (*DataReadHandler* &handler)

Read next packet from the camera and use a handler object to handle all types of packets. The function returns a true if end-of-file was not reached, so this function call can be used in a while loop like so:

```
while (camera.handleNext(handler)) {  
    // While-loop executes after each packet  
}
```

Parameters

handler – Handler instance that contains callback functions to handle different packets.

Returns

False to indicate end of data stream, true to continue.

Protected Attributes

SortedPacketBuffers **mBuffers**

Friends

inline friend *std::ostream* &**operator<<** (*std::ostream* &os, const *Flatten* &var)

template<size_t **radius**>

struct **CircleCoordinates**

template<>

struct **CircleCoordinates**<3>

Public Static Attributes

```
static std::vector<Eigen::Vector2i, Eigen::aligned_allocator<Eigen::Vector2i>> coords{{Eigen::Vector2i{0, 3},  
Eigen::Vector2i{1, 3}, Eigen::Vector2i{2, 2}, Eigen::Vector2i{3, 1}, Eigen::Vector2i{3, 0}, Eigen::Vector2i{3, -1},  
Eigen::Vector2i{2, -2}, Eigen::Vector2i{1, -3}, Eigen::Vector2i{0, -3}, Eigen::Vector2i{-1, -3}, Eigen::Vector2i{-2,  
-2}, Eigen::Vector2i{-3, -1}, Eigen::Vector2i{-3, 0}, Eigen::Vector2i{-3, 1}, Eigen::Vector2i{-2, 2},  
Eigen::Vector2i{-1, 3}}}
```

template<>

struct **CircleCoordinates**<4>

Public Static Attributes

```
static std::vector<Eigen::Vector2i, Eigen::aligned_allocator<Eigen::Vector2i>> coords{{Eigen::Vector2i{0, 4},
Eigen::Vector2i{1, 4}, Eigen::Vector2i{2, 3}, Eigen::Vector2i{3, 2}, Eigen::Vector2i{4, 1}, Eigen::Vector2i{4, 0},
Eigen::Vector2i{4, -1}, Eigen::Vector2i{3, -2}, Eigen::Vector2i{2, -3}, Eigen::Vector2i{1, -4}, Eigen::Vector2i{0,
-4}, Eigen::Vector2i{-1, -4}, Eigen::Vector2i{-2, -3}, Eigen::Vector2i{-3, -2}, Eigen::Vector2i{-4, -1},
Eigen::Vector2i{-4, 0}, Eigen::Vector2i{-4, 1}, Eigen::Vector2i{-3, 2}, Eigen::Vector2i{-2, 3}, Eigen::Vector2i{-1,
4}}}
```

```
template<>
```

```
struct CircleCoordinates<5>
```

Public Static Attributes

```
static std::vector<Eigen::Vector2i, Eigen::aligned_allocator<Eigen::Vector2i>> coords{{Eigen::Vector2i{0, 5},
Eigen::Vector2i{1, 5}, Eigen::Vector2i{2, 5}, Eigen::Vector2i{3, 4}, Eigen::Vector2i{4, 3}, Eigen::Vector2i{5, 2},
Eigen::Vector2i{5, 1}, Eigen::Vector2i{5, 0}, Eigen::Vector2i{5, -1}, Eigen::Vector2i{5, -2}, Eigen::Vector2i{4,
-3}, Eigen::Vector2i{3, -4}, Eigen::Vector2i{2, -5}, Eigen::Vector2i{1, -5}, Eigen::Vector2i{0, -5},
Eigen::Vector2i{-1, -5}, Eigen::Vector2i{-2, -5}, Eigen::Vector2i{-3, -4}, Eigen::Vector2i{-4, -3},
Eigen::Vector2i{-5, -2}, Eigen::Vector2i{-5, -1}, Eigen::Vector2i{-5, 0}, Eigen::Vector2i{-5, 1},
Eigen::Vector2i{-5, 2}, Eigen::Vector2i{-4, 3}, Eigen::Vector2i{-3, 4}, Eigen::Vector2i{-2, 5}, Eigen::Vector2i{-1,
5}}}
```

```
template<>
```

```
struct CircleCoordinates<6>
```

Public Static Attributes

```
static std::vector<Eigen::Vector2i, Eigen::aligned_allocator<Eigen::Vector2i>> coords{{Eigen::Vector2i{0, 6},
Eigen::Vector2i{1, 6}, Eigen::Vector2i{2, 6}, Eigen::Vector2i{3, 5}, Eigen::Vector2i{4, 4}, Eigen::Vector2i{5, 3},
Eigen::Vector2i{6, 2}, Eigen::Vector2i{6, 1}, Eigen::Vector2i{6, 0}, Eigen::Vector2i{6, -1}, Eigen::Vector2i{6, -2},
Eigen::Vector2i{5, -3}, Eigen::Vector2i{4, -4}, Eigen::Vector2i{3, -5}, Eigen::Vector2i{2, -6}, Eigen::Vector2i{1,
-6}, Eigen::Vector2i{0, -6}, Eigen::Vector2i{-1, -6}, Eigen::Vector2i{-2, -6}, Eigen::Vector2i{-3, -5},
Eigen::Vector2i{-4, -4}, Eigen::Vector2i{-5, -3}, Eigen::Vector2i{-6, -2}, Eigen::Vector2i{-6, -1},
Eigen::Vector2i{-6, 0}, Eigen::Vector2i{-6, 1}, Eigen::Vector2i{-6, 2}, Eigen::Vector2i{-5, 3}, Eigen::Vector2i{-4,
4}, Eigen::Vector2i{-3, 5}, Eigen::Vector2i{-2, 6}, Eigen::Vector2i{-1, 6}}}
```

```
template<>
```

```
struct CircleCoordinates<7>
```

Public Static Attributes

```
static std::vector<Eigen::Vector2i, Eigen::aligned_allocator<Eigen::Vector2i>> coords{{Eigen::Vector2i{0, 7},
Eigen::Vector2i{1, 7}, Eigen::Vector2i{2, 7}, Eigen::Vector2i{3, 7}, Eigen::Vector2i{4, 6}, Eigen::Vector2i{5, 5},
Eigen::Vector2i{6, 4}, Eigen::Vector2i{7, 3}, Eigen::Vector2i{7, 2}, Eigen::Vector2i{7, 1}, Eigen::Vector2i{7, 0},
Eigen::Vector2i{7, -1}, Eigen::Vector2i{7, -2}, Eigen::Vector2i{7, -3}, Eigen::Vector2i{6, -4}, Eigen::Vector2i{5,
-5}, Eigen::Vector2i{4, -6}, Eigen::Vector2i{3, -7}, Eigen::Vector2i{2, -7}, Eigen::Vector2i{1, -7},
Eigen::Vector2i{0, -7}, Eigen::Vector2i{-1, -7}, Eigen::Vector2i{-2, -7}, Eigen::Vector2i{-3, -7},
Eigen::Vector2i{-4, -6}, Eigen::Vector2i{-5, -5}, Eigen::Vector2i{-6, -4}, Eigen::Vector2i{-7, -3},
Eigen::Vector2i{-7, -2}, Eigen::Vector2i{-7, -1}, Eigen::Vector2i{-7, 0}, Eigen::Vector2i{-7, 1},
Eigen::Vector2i{-7, 2}, Eigen::Vector2i{-7, 3}, Eigen::Vector2i{-6, 4}, Eigen::Vector2i{-5, 5}, Eigen::Vector2i{-4,
6}, Eigen::Vector2i{-3, 7}, Eigen::Vector2i{-2, 7}, Eigen::Vector2i{-1, 7}, Eigen::Vector2i{0, 7}}}
```

class **CircularTimeSurfaceView**

Public Types

using **CoordVector** = *std::vector*<Eigen::Vector2i, Eigen::aligned_allocator<Eigen::Vector2i>>

Public Functions

inline explicit **CircularTimeSurfaceView** (*CoordVector* &coords)

inline explicit **CircularTimeSurfaceView** (*CoordVector* &&coords)

inline auto **getTimestamp** (const *dv::Event* &e, const Eigen::Vector2i &circleCoords, const *TimeSurface* &ts)
const

template<typename **ITERATOR**>

inline auto **circularIncrement** (const *ITERATOR* it) const

template<typename **ITERATOR**>

inline auto **circularDecrement** (const *ITERATOR* it) const

Public Members

CoordVector **mCoords**

struct **CoarseFineBias**

On-chip coarse-fine bias current configuration. See <https://docs.inivation.com/hardware/hardware-advanced-usage/biasing.html>²⁴ for more details.

Public Functions

constexpr **CoarseFineBias** () = default

inline constexpr **CoarseFineBias** (const *CoarseFineBiasSex* sex)

inline constexpr **CoarseFineBias** (const uint8_t coarse, const uint8_t fine, const *CoarseFineBiasSex* sex, const *CoarseFineBiasType* type = *CoarseFineBiasType::NORMAL*, const bool enable = true)

Public Members

uint8_t **coarseValue** = {0}

Coarse current, from 0 to 7, creates big variations in output current.

uint8_t **fineValue** = {0}

Fine current, from 0 to 255, creates small variations in output current.

bool **enabled** = {false}

Whether this bias is enabled or not.

CoarseFineBiasSex **sex** = {*CoarseFineBiasSex*::*N_TYPE*}

Bias sex: true for 'N' type, false for 'P' type.

CoarseFineBiasType **type** = {*CoarseFineBiasType*::*NORMAL*}

Bias type: true for 'Normal', false for 'Cascode'.

CoarseFineBiasCurrentLevel **currentLevel** = {*CoarseFineBiasCurrentLevel*::*NORMAL*}

Bias current level: true for 'Normal', false for 'Low'.

class **CompressionSupport**

Subclassed by *dv::io::compression::Lz4CompressionSupport*, *dv::io::compression::NoneCompressionSupport*, *dv::io::compression::ZstdCompressionSupport*

Public Functions

inline explicit **CompressionSupport** (const *CompressionType* type)

virtual ~**CompressionSupport** () = default

virtual void **compress** (*dv::io::support::IODataBuffer* &packet) = 0

inline *CompressionType* **getCompressionType** () const

Private Members

CompressionType **mType**

class **Config**

#include </builds/inivation/dv/dv-processing/include/dv-processing/io/mono_camera_writer.hpp> A configuration structure for the *MonoCameraWriter*.

Public Functions

inline void **addStreamMetadata** (const *std::string* &name, const *std::pair*<*std::string*, *dv::io::support::VariantValueOwning*> &metadataEntry)

Add a metadata entry for a data type stream.

Parameters

- **name** – Name of the stream.
- **metadataEntry** – Metadata entry consisting of a pair, where first element is the key name of the stream and second element is the value.

inline void **addEventStream** (const cv::Size &resolution, const *std::string* &name = "events", const *std::optional*<*std::string*> &source = *std::nullopt*)

Add an event stream with a given resolution.

Parameters

- **resolution** – Resolution of the event sensor.

²⁴ <https://docs.inivation.com/hardware/hardware-advanced-usage/biasing.html>

- **name** – Name of the stream
- **source** – Name of the source camera.

```
inline void addFrameStream (const cv::Size &resolution, const std::string &name = "frames", const  
                           std::optional<std::string> &source = std::nullopt)
```

Add a frame stream with a given resolution.

Parameters

- **resolution** – Resolution of the frame sensor.
- **name** – Name of the stream
- **source** – Name of the source camera.

```
inline void addImuStream (const std::string &name = "imu", const std::optional<std::string> &source =  
                          std::nullopt)
```

Add an imu data stream.

Parameters

name – *Stream* name, with a default value of “imu”.

```
inline void addTriggerStream (const std::string &name = "triggers", const std::optional<std::string> &source  
                             = std::nullopt)
```

Add a trigger stream.

Parameters

name – *Stream* name, with a default value of “triggers”.

```
template<class PacketType>
```

```
inline void addStream (const std::string &name, const std::optional<std::string> &source = std::nullopt)
```

Add a stream of given data type.

Template Parameters

PacketType – *Stream* data packet type.

Parameters

- **name** – Name for the stream.
- **source** – Camera name for the source of the data, usually a concatenation of “MODEL_SERIAL”, e.g. “DVXplorer_DXA000000”

```
inline std::optional<cv::Size> findStreamResolution (const std::string &name) const
```

Parse resolution of the stream from metadata of the stream. Resolution should be set as two metadata parameters: “sizeX” and “sizeY” parameters.

Parameters

name – *Stream* name.

Returns

Configured resolution. `std::nullopt` if unavailable or incorrectly configured.

```
inline explicit Config (const std::string &cameraName, CompressionType compression =  
                       CompressionType::LZ4)
```

Create a config instance

Parameters

- **cameraName** –
- **compression** –

Public Members

dv::CompressionType **compression**

Compression type for this file.

std::string **cameraName**

Camera name that produces the data, usually contains production serial number.

Private Members

std::map<std::string, std::string> **customDataStreams**

std::map<std::string, std::map<std::string, dv::io::support::VariantValueOwning>>
customDataStreamsMetadata

Friends

friend class *dv::io::MonoCameraWriter*

friend class *dv::io::StereoCameraWriter*

class **Connection** : public *std::enable_shared_from_this<Connection>*

Connection helper class that maintains shared pointer to itself when called on the public API methods.

This class should be wrapped in a shared pointer and start method should be called. This will intrinsically increment the reference count to maintain the pointer to itself even if the wrapper *shared_ptr* goes out-of-scope until the instance gets API calls to write data into the buffer. During destruction, the instance will remove it's own pointer from a connection list in the top-level class.

(Personal comment by Rokas): this seems over-engineered and unnecessary, but it's the way ASIO works and, although there are other ways to implement it, it just doesn't work with other approaches leading to undefined behaviors.

Public Functions

inline **Connection** (*WriteOrderedSocket &&socket, NetworkWriter *const server*)

inline **~Connection** ()

inline void **start** ()

inline void **close** ()

inline void **writePacket** (const *std::shared_ptr<const dv::io::support::IODataBuffer>* &packet)

inline bool **isOpen** () const

Private Functions

inline void **writeIOHeader** (const *std::shared_ptr<const dv::io::support::IODataBuffer>* &ioHeader)

inline void **keepAliveByReading** ()

inline void **handleError** (const *boost::system::error_code* &error, const *std::string_view* message)

Private Members

Network Writer *mParent

WriteOrderedSocket mSocket

uint8_t mKeepAliveReadSpace = {0}

template<class Functor>

class ContrastMaximizationWrapper

#include </builds/inivation/dv/dv-processing/include/dv-processing/optimization/contrast_maximization_wrapper.hpp>
Wrapper for all contrast maximization algorithms. For more information about contrast maximization please check “contrast_maximization_rotation.hpp” or “contrast_maximization_translation_and_depth.hpp”. This wrapper is mainly meant to set the non linear differentiation parameters (see constructor for more information). In addition, the class expose to user only “optimize” function which returns a struct containing the result of the non-linear optimization (successful or not), number of iteration of the optimization and optimized parameters.

Template Parameters

Functor – Functor that handles optimization. Cost is computed by overriding operator() method. For an example of a functor please check “contrast_maximization_rotation.hpp” or “contrast_maximization_translation_and_depth.hpp”.

Public Functions

inline ContrastMaximizationWrapper (*std::unique_ptr<Functor>* functor_, float learningRate, float epsfcn = 0, float ftol = 0.000345267, float gtol = 0, float xtol = 0.000345267, int maxfev = 400)

Parameters

- **functor_** – functor handling contrast maximization optimization. the functor should inherit “OptimizationFunctor” and overload the “int operator()” method to compute cost for contrast maximization and optimize pre-defined parameters.
- **learningRate** – constant multiplying input value to find new value at which function will be evaluated. E.g. assuming function is evaluated at $x \rightarrow f(x)$, next input sample x' is computed as $x' = \text{abs}(x) * \text{learningRate}$.
- **epsfcn** – error precision
- **ftol** – tolerance for the norm of the vector function
- **gtol** – tolerance for the norm of the gradient of the error vector
- **xtol** – tolerance for the norm of the solution vector
- **maxfev** – max number of function evaluations Note that default parameters are taken from default parameters of LevenbergMarquardt optimizer.

inline optimizationOutput **optimize** (const Eigen::VectorXf &initialValues)

Function optimizing cost defined in mFunctor (inside operator() method).

Parameters

initialValues – Initial values of variables to be optimized.

Returns

optimized variable that minimize cost.

Private Members

`std::unique_ptr<Functor> mFunctor = nullptr`

`optimizationParameters mParams`

struct **controlInBuffer**

Public Members

`uint8_t setup[LIBUSB_CONTROL_SETUP_SIZE] = {}`

`uint8_t buffer[MAX_CONTROL_TRANSFER_SIZE] = {}`

`controlInCallbackType callback = {}`

struct **controlOutBuffer**

Public Members

`uint8_t setup[LIBUSB_CONTROL_SETUP_SIZE] = {}`

`uint8_t buffer[MAX_CONTROL_TRANSFER_SIZE] = {}`

`controlOutCallbackType callback = {}`

struct **DataReadHandler**

`#include </builds/inivation/dv/dv-processing/include/dv-processing/io/data_read_handler.hpp>` Read handler that can handle all supported types in *MonoCameraRecording*.

Public Types

enum class **OutputFlag**

Values:

enumerator **END_OF_FILE**

enumerator **CONTINUE**

Public Functions

inline void **operator ()** (const *dv::EventStore* &events)

Internal call to handle input data

Parameters

events –

```
inline void operator () (const dv::Frame &frame)
```

Internal call to handle input data

Parameters

frame –

```
inline void operator () (const std::vector<dv::Trigger> &triggers)
```

Internal call to handle input data

Parameters

triggers –

```
inline void operator () (const std::vector<dv::IMU> &imu)
```

Internal call to handle input data

Parameters

imu –

```
inline void operator () (const OutputFlag flag)
```

Internal call to handle input data

Parameters

flag –

Public Members

```
std::optional<std::function<void(const dv::EventStore&)>> mEventHandler = std::nullopt
```

Event handler that is going to be called on each arriving event batch.

```
std::optional<std::function<void(const dv::Frame&)>> mFrameHandler = std::nullopt
```

Frame handler that is called on each arriving frame.

```
std::optional<std::function<void(const std::vector<dv::IMU>&)>> mImuHandler = std::nullopt
```

IMU data handler that is going to be called on each arriving imu data batch.

```
std::optional<std::function<void(const std::vector<dv::Trigger>&)>> mTriggersHandler = std::nullopt
```

Trigger data handler that is going to be called on each arriving trigger data batch.

```
std::optional<std::function<void(const OutputFlag)>> mOutputFlagHandler = std::nullopt
```

A handler for output flags that can indicate some file behaviour, e.g. end-of-file.

```
bool eof = false
```

Is end of file reached.

```
int64_t seek = -1
```

Timestamp holding latest seek position of the recording

```
class DAVIS : public dv::io::camera::USBDevice, public dv::io::camera::SyncCameraInputBase
```

Public Types

enum class **Davis240BiasCF**

Values:

enumerator **Diff**

enumerator **On**

enumerator **Off**

enumerator **ApsCascode**

enumerator **DiffCascode**

enumerator **ApsReadoutSourceFollower**

enumerator **LocalBuffer**

enumerator **PixelInverter**

enumerator **Photoreceptor**

enumerator **PhotoreceptorSourceFollower**

enumerator **Refractory**

enumerator **AERPullDown**

enumerator **LCOLTimeout**

enumerator **AERPullUpX**

enumerator **AERPullUpY**

enumerator **PadFollower**

enumerator **ApsOverflowLevel**

enumerator **BiasBuffer**

enum class **Davis346BiasVDAC**

Values:

enumerator **ApsOverflowLevel**

enumerator **ApsCascode**

enumerator **ADCReferenceHigh**

enumerator **ADCReferenceLow**

enumerator **ADCTestVoltage**

enum class **Davis346BiasCF**

Values:

enumerator **LocalBuffer**

enumerator **PadFollower**

enumerator **Diff**

enumerator **On**

enumerator **Off**

enumerator **PixelInverter**

enumerator **Photoreceptor**

enumerator **PhotoreceptorSourceFollower**

enumerator **Refractory**

enumerator **ReadoutBuffer**

enumerator **ApsReadoutSourceFollower**

enumerator **ADCComparator**

enumerator **COLSelectLow**

enumerator **DACBuffer**

enumerator **LCOLTimeout**

enumerator **AERPullDown**

enumerator **AERPullUpX**

enumerator **AERPullUpY**

enumerator **BiasBuffer**

enum class **CDavisBiasVDAC**

Values:

enumerator **ApsCascode**

enumerator **OVG1Low**

enumerator **OVG2Low**

enumerator **TX2OVG2High**

enumerator **Gnd07**

enumerator **ADCTestVoltage**

enumerator **ADCReferenceHigh**

enumerator **ADCReferenceLow**

enum class **CDavisBiasCF**

Values:

enumerator **LocalBuffer**

enumerator **PadFollower**

enumerator **PixelInverter**

enumerator **Diff**

enumerator **On**

enumerator **Off**

enumerator **Photoreceptor**

enumerator **PhotoreceptorSourceFollower**

enumerator **Refractory**

enumerator **ArrayBiasBuffer**

enumerator **ArrayLogicBuffer**

enumerator **FallTime**

enumerator **RiseTime**

enumerator **ReadoutBuffer**

enumerator **ApsReadoutSourceFollower**

enumerator **ADCComparator**

enumerator **DACBuffer**

enumerator **LCOLTimeout**

enumerator **AERPullDown**

enumerator **AERPullUpX**

enumerator **AERPullUpY**

enumerator **BiasBuffer**

using **AutoExposureCallback** = *std::function*<*dv::Duration*(const *dv::Frame* &frame)>

Public Functions

inline explicit **DAVIS** (loggerCallbackType deviceLogger = {}, const LogLevel deviceLogLevel =
LogLevel::LVL_WARNING)

Open the first *DAVIS* camera that can be found. Throws if device cannot be opened.

Parameters

- **deviceLogger** – per-device logging callback, must be thread-safe
- **deviceLogLevel** – initial log-level

```
inline explicit DAVIS (const std::string_view filterBySerialNumber, loggerCallbackType deviceLogger = {}, const
    LogLevel deviceLogLevel = LogLevel::LVL_WARNING)
```

Open the *DAVIS* camera with the specified serial number. Throws if device cannot be opened.

Parameters

- **filterBySerialNumber** – serial number to search for
- **deviceLogger** – per-device logging callback, must be thread-safe
- **deviceLogLevel** – initial log-level

```
inline explicit DAVIS (const DeviceDescriptor &deviceToOpen, loggerCallbackType deviceLogger = {}, const
    LogLevel deviceLogLevel = LogLevel::LVL_WARNING)
```

Open the *DAVIS* camera corresponding to the specified descriptor. Throws if device cannot be opened.

Parameters

- **deviceToOpen** – device descriptor structure
- **deviceLogger** – per-device logging callback, must be thread-safe
- **deviceLogLevel** – initial log-level

```
inline ~DAVIS () override
```

```
inline virtual std::string getCameraName () const override
```

Get camera name, which is a combination of the camera model and the serial number.

Returns

String containing the camera model and serial number separated by an underscore character.

```
inline uint32_t getLogicVersion () const
```

Get camera FPGA logic version.

Returns

camera FPGA logic version

```
inline uint32_t getLogicPatchLevel () const
```

Get camera FPGA logic patch level.

Returns

camera FPGA logic patch level

```
inline parser::DAVIS::SensorModel getSensorModel () const
```

Get the exact model of the *DAVIS* sensor chip mounted in this camera.

Returns

camera *DAVIS* sensor chip model

```
inline virtual std::optional<cv::Size> getEventResolution () const override
```

Get event stream resolution.

Returns

Event stream resolution, `std::nullopt` if event stream is unavailable.

```
inline virtual std::optional<cv::Size> getFrameResolution () const override
```

Retrieve frame stream resolution.

Returns

Frame stream resolution or `std::nullopt` if the frame stream is not available.

inline virtual *imu::ImuModel* **getImuModel** () const override

Return *IMU* model used on device.

Returns

IMU model in use.

inline virtual float **getPixelPitch** () const override

Return pixel pitch distance for the connected camera model. The value is returned in meters.

Returns

Pixel pitch distance in meters according to the connected device.

inline virtual bool **isMaster** () const override

Report if this camera is a clock synchronization master.

Returns

true if clock master, false otherwise.

inline virtual bool **isEventStreamAvailable** () const override

Check whether event stream is available.

Returns

True if event stream is available, false otherwise.

inline virtual bool **isFrameStreamAvailable** () const override

Check whether frame stream is available.

Returns

True if frame stream is available, false otherwise.

inline virtual bool **isImuStreamAvailable** () const override

Check whether *IMU* data is available.

Returns

True if *IMU* data stream is available, false otherwise.

inline virtual bool **isTriggerStreamAvailable** () const override

Check whether trigger data is available.

Returns

True if trigger data stream is available, false otherwise.

inline virtual bool **isRunning** () const override

Check whether any input data streams have terminated. For a live camera this should check if the device is still connected and functioning, while for a recording file this should check if any of the data streams have reached end-of-file (EOF). For a network input, this indicates the network stream is still connected.

Returns

True if data read on all streams is still possible, false otherwise.

inline virtual bool **isRunning** (const *std::string_view* streamName) const override

Check whether the input data stream with the specified name is still active.

Returns

True if data read on this stream is possible, false otherwise.

inline virtual bool **isRunningAny** () const override

Check whether any input data streams are still available. For a live camera this should check if the device is still connected and functioning and at least one data stream is active (different than *isRunning()*), while for a recording file this should check if any of the data streams have not yet reached end-of-file (EOF) and are still readable. For a network input, this indicates the network stream is still connected.

Returns

True if data read on at least one stream is still possible, false otherwise.

inline virtual `std::chrono::microseconds` **getTimeInterval** () const override

Get the time interval for data commit.

Returns

Time interval in microseconds.

inline virtual void **setTimeInterval** (const `std::chrono::microseconds` timeInterval) override

Set a new time interval value for data commit. Data is put in the queues for `getNextEventBatch()`, `readNext()`, ... at this interval's rate.

Parameters

timeInterval – New time interval value in microseconds.

inline virtual `std::chrono::microseconds` **getTimestampOffset** () const override

Get the timestamp offset.

Returns

Absolute timestamp offset value in microseconds.

inline `dv::PixelArrangement` **getPixelArrangementEvents** () const

Get pixel color filter arrangement for event readout on color `DAVIS`. Takes flips into account.

Returns

current color filter

inline `dv::PixelArrangement` **getPixelArrangementFrames** () const

Get pixel color filter arrangement for frame readout on color `DAVIS`. Takes flips into account.

Returns

current color filter

inline `dv::EventColor` **getEventColor** (const `dv::Event` &event) const

Determine the color of the Bayer color filter for a specific event, based on its pixel address. WHITE means White/No Filter. Please take into account that there are usually twice as many green pixels as there are red or blue ones.

Parameters

event – event pixel to get color for.

Returns

event color filter value.

inline bool **isEventsRunning** () const

Report if the event output is running.

Returns

true if events active, false otherwise.

inline void **setEventsRunning** (const bool run)

Enable or disable the event output.

Parameters

run – whether to enable the event output or not.

inline virtual bool **getFlipHorizontal** () const override

Status of horizontal events flip.

Returns

status of horizontal events flip.

inline virtual void **setFlipHorizontal** (const bool flipHorizontalEvents) override

Flip events horizontally.

Parameters

flipHorizontalEvents – flip events horizontally.

inline virtual bool **getFlipVertical** () const override

Status of vertical events flip.

Returns

status of vertical events flip.

inline virtual void **setFlipVertical** (const bool flipVerticalEvents) override

Flip events vertically.

Parameters

flipVerticalEvents – flip events vertically.

inline bool **getFlipHorizontalEvents** () const

Status of horizontal events flip.

Returns

status of horizontal events flip.

inline void **setFlipHorizontalEvents** (const bool flipHorizontalEvents)

Flip events horizontally.

Parameters

flipHorizontalEvents – flip events horizontally.

inline bool **getFlipVerticalEvents** () const

Status of vertical events flip.

Returns

status of vertical events flip.

inline void **setFlipVerticalEvents** (const bool flipVerticalEvents)

Flip events vertically.

Parameters

flipVerticalEvents – flip events vertically.

inline bool **getExternalAERControl** () const

Get external AER control status. If true, the AER handshake with the sensor chip is not done by our camera's FPGA, but by some external system provided by the customer.

Returns

external AER control status

inline void **setExternalAERControl** (const bool externalAERControl)

Enable or disable external AER control feature. If true, the AER handshake with the sensor chip is not done by our camera's FPGA, but by some external system provided by the customer. This also disables the normal event output (*isEventsRunning()* returns false). Disabling this will not re-enable the normal event output automatically!

Parameters

externalAERControl – true to enable external AER control, false to disable.

inline uint16_t **getBackgroundActivityFilter** () const

Get background activity noise filter time delta value. If 0, filtering is disabled.

Returns

time delta value in 250µs multiples

inline void **setBackgroundActivityFilter** (const uint16_t timeIn250us)

Set background activity noise filter time delta value. If 0, filtering is disabled.

Parameters

timeIn250us – time delta value in 250µs multiples

inline uint16_t **getRefractoryPeriodFilter** () const

Get refractory period filter time delta value. If 0, filtering is disabled.

Returns

time delta value in 250µs multiples

inline void **setRefractoryPeriodFilter** (const uint16_t timeIn250us)

Set refractory period filter time delta value. If 0, filtering is disabled.

Parameters

timeIn250us – time delta value in 250µs multiples

inline uint8_t **getSkipFilter** () const

Get event skip filter skip value. If 0, filtering is disabled.

Returns

drop every Nth event

inline void **setSkipFilter** (const uint8_t skipEvery)

Set event skip filter skip value. Drops every Nth event. If 0, filtering is disabled.

Parameters

skipEvery – drop every Nth event

inline virtual Flatten **getFlatten** () const override

Status of event polarity flattening.

Returns

status of event polarity flattening.

inline virtual void **setFlatten** (const Flatten flatten) override

Flatten events polarity.

Parameters

flattenEvents – flattening mode.

inline virtual cv::Rect **getCropArea** () const override

Get events Region of Interest (ROI).

Returns

get events Region of Interest (ROI).

inline virtual void **setCropArea** (const cv::Rect cropAreaEvents) override

Set events Region of Interest (ROI). Usually hardware accelerated.

Parameters

cropAreaEvents – region of interest (ROI) position and size.

inline cv::Rect **getCropAreaEvents** () const

Get events Region of Interest (ROI).

Returns

get events Region of Interest (ROI).

inline void **setCropAreaEvents** (cv::Rect cropAreaEvents)

Set events Region of Interest (ROI). Usually hardware accelerated.

Parameters

cropAreaEvents – region of interest (ROI) position and size.

inline bool **isFramesRunning** () const

Report if the frame output is running.

Returns

true if frames active, false otherwise.

inline void **setFramesRunning** (const bool run)

Enable or disable the frame output.

Parameters

run – whether to enable the frame output or not.

inline void **snapshot** ()

Takes a snapshot (one frame), like a photo-camera. More efficient implementation that just toggling *setFramesRunning()* on and off. The frame output should not be running prior to calling this, as it only makes sense if frames are not being generated at the time. Also, *setFrameInterval()* should be set to zero if only doing snapshots, to ensure a quicker readiness for the next one, since the delay is always observed after taking a frame.

inline *dv::io::camera::parser::DAVIS::ColorMode* **getColorMode** () const

Get frames color mode.

Returns

frame color mode

inline void **setColorMode** (const *dv::io::camera::parser::DAVIS::ColorMode* colorMode)

Set frames color mode. For monochrome sensors, all modes are equivalent. For color sensors, DEFAULT will output an RGB frame after debayering, GRAYSCALE will debayer to one channel, and ORIGINAL will return the unchanged readout from the pixels, in case you want to do debayering yourself.

Parameters

colorMode – frame color mode

inline bool **getAutoExposure** () const

Report if auto-exposure feature is enabled.

Returns

true if auto-exposure enabled, false otherwise.

inline void **setAutoExposure** (const bool enabled)

Enable auto-exposure feature. Automatic exposure control, tries to set the exposure value automatically to an appropriate value to maximize information in the scene and minimize under- and over-exposure.

Parameters

enabled – true to enable auto-exposure, false to disable.

inline void **setAutoExposureCallback** (*AutoExposureCallback* callback)

Set custom callback function to compute new exposure value based on current frame output.

Parameters

callback –

inline `std::chrono::microseconds` **getExposureDuration** () const

Get current frame exposure duration. Also works with auto-exposure, reporting last automatically set value.

Returns

current exposure duration in microseconds.

inline void **setExposureDuration** (const `std::chrono::microseconds` exposureDurationUs)

Set frame exposure duration. Auto-exposure must be disabled.

Parameters

exposureDurationUs – exposure duration in microseconds.

inline `std::chrono::microseconds` **getFrameInterval** () const

Get frame interval, the time between two consecutive frames.

Returns

frame interval in microseconds.

inline void **setFrameInterval** (const `std::chrono::microseconds` frameIntervalUs)

Set frame interval, the time between two consecutive frames.

Parameters

frameIntervalUs – frame interval in microseconds.

inline bool **getFlipHorizontalFrames** () const

Status of horizontal frames flip.

Returns

status of horizontal frames flip.

inline void **setFlipHorizontalFrames** (const bool flipHorizontalFrames)

Flip frames horizontally.

Parameters

flipHorizontalFrames – flip frames horizontally.

inline bool **getFlipVerticalFrames** () const

Status of vertical frames flip.

Returns

status of vertical frames flip.

inline void **setFlipVerticalFrames** (const bool flipVerticalFrames)

Flip frames vertically.

Parameters

flipVerticalFrames – flip frames vertically.

inline `cv::Rect` **getCropAreaFrames** () const

Get frames Region of Interest (ROI).

Returns

get frames Region of Interest (ROI).

inline void **setCropAreaFrames** (`cv::Rect` cropAreaFrames)

Set frames Region of Interest (ROI). Usually hardware accelerated.

Parameters

cropAreaFrames – region of interest (ROI) position and size.

inline bool **isImuRunningAccelerometer** () const

Status of *IMU* accelerometer.

Returns

true if enabled, false otherwise.

inline void **setImuRunningAccelerometer** (const bool run)

Enable or disable *IMU* accelerometer.

Parameters

run – true to enable, false to disable.

inline bool **isImuRunningGyroscope** () const

Status of *IMU* gyroscope.

Returns

true if enabled, false otherwise.

inline void **setImuRunningGyroscope** (const bool run)

Enable or disable *IMU* gyroscope.

Parameters

run – true to enable, false to disable.

inline bool **isImuRunningTemperature** () const

Status of *IMU* temperature measurement.

Returns

true if enabled, false otherwise.

inline void **setImuRunningTemperature** (const bool run)

Enable or disable *IMU* temperature measurement.

Parameters

run – true to enable, false to disable.

inline bool **getIMUFlipX** () const

Status of *IMU* X axis flipping.

Returns

true if enabled, false otherwise.

inline void **setIMUFlipX** (const bool flipX)

Enable or disable *IMU* X axis flipping. Will negate (flip) all returned X axis values.

Parameters

flipX – true to enable, false to disable.

inline bool **getIMUFlipY** () const

Status of *IMU* Y axis flipping.

Returns

true if enabled, false otherwise.

inline void **setIMUFlipY** (const bool flipY)

Enable or disable *IMU* Y axis flipping. Will negate (flip) all returned Y axis values.

Parameters

flipY – true to enable, false to disable.

inline bool **getIMUFlipZ** () const

Status of *IMU* Z axis flipping.

Returns

true if enabled, false otherwise.

inline void **setIMUFlipZ** (const bool flipZ)

Enable or disable *IMU* Z axis flipping. Will negate (flip) all returned Z axis values.

Parameters

flipZ – true to enable, false to disable.

inline *imu::InvensenseAccelRange* **getImuAccelRange** () const

Get current *IMU* accelerometer range.

Returns

accelerometer range.

inline void **setImuAccelRange** (const *imu::InvensenseAccelRange* range)

Set *IMU* accelerometer range.

Parameters

range – accelerometer range.

inline *imu::InvensenseGyroRange* **getImuGyroRange** () const

Get current *IMU* gyroscope range.

Returns

gyroscope range.

inline void **setImuGyroRange** (const *imu::InvensenseGyroRange* range)

Set *IMU* gyroscope range.

Parameters

range – gyroscope range.

inline bool **isDetectorRunning** () const

Report status of external signal detector.

Returns

true if running, false otherwise.

inline void **setDetectorRunning** (const bool run)

Enable or disable external signal detector.

Parameters

run – true to enable, false to disable.

inline bool **getDetectorRisingEdges** () const

Report status of rising edge detection on the SIGNAL_IN line.

Returns

true if enabled, false otherwise.

inline void **setDetectorRisingEdges** (const bool detectRising)

Detect rising edges (low to high transitions) on the SIGNAL_IN line.

Parameters

detectRising – true to enable, false to disable.

inline bool **getDetectorFallingEdges** () const

Report status of falling edge detection on the SIGNAL_IN line.

Returns

true if enabled, false otherwise.

inline void **setDetectorFallingEdges** (const bool detectFalling)

Detect falling edges (high to low transitions) on the SIGNAL_IN line.

Parameters

detectFalling – true to enable, false to disable.

inline bool **isGeneratorRunning** () const

Report status of external signal generator.

Returns

true if running, false otherwise.

inline void **setGeneratorRunning** (const bool run)

Enable or disable external signal generator (PWM-like output).

Parameters

run – true to enable, false to disable.

inline [std::chrono::microseconds](#) **getGeneratorLowTime** () const

Get current PWM low time.

Returns

low time in microseconds.

inline void **setGeneratorLowTime** (const [std::chrono::microseconds](#) lowTimeUs)

Set PWM low time for external signal generator.

Parameters

lowTimeUs – low time in microseconds.

inline [std::chrono::microseconds](#) **getGeneratorHighTime** () const

Get current PWM high time.

Returns

high time in microseconds.

inline void **setGeneratorHighTime** (const [std::chrono::microseconds](#) highTimeUs)

Set PWM high time for external signal generator.

Parameters

highTimeUs – high time in microseconds.

inline bool **getGeneratorInjectTriggerOnRisingEdge** () const

Report status of trigger event injection feature for external signal generator rising edges.

Returns

true if enabled, false otherwise.

inline void **setGeneratorInjectTriggerOnRisingEdge** (const bool injectRising)

Inject a trigger event of type EXTERNAL_GENERATOR_RISING_EDGE into the event stream from the device, every time a rising edge is generated by the PWM-like output of the external signal generator.

Parameters

injectRising – true to inject trigger event, false to disable.

inline bool **getGeneratorInjectTriggerOnFallingEdge** () const

Report status of trigger event injection feature for external signal generator falling edges.

Returns

true if enabled, false otherwise.

inline void **setGeneratorInjectTriggerOnFallingEdge** (const bool injectFalling)

Inject a trigger event of type EXTERNAL_GENERATOR_FALLING_EDGE into the event stream from the device, every time a falling edge is generated by the PWM-like output of the external signal generator.

Parameters

injectFalling – true to inject trigger event, false to disable.

inline *std::*pair<uint8_t, uint8_t> **getDavis240BiasCoarseFine** (const *Davis240BiasCF* bias) const

Get low-level coarse/fine values for a specific bias.

Parameters

bias – bias to query.

Returns

coarse/fine values.

inline void **setDavis240BiasCoarseFine** (const *Davis240BiasCF* bias, const uint8_t coarse, const uint8_t fine)

Set specific bias to new low-level coarse/fine values.

Parameters

- **bias** – bias to configure.
- **coarse** – coarse bias value.
- **fine** – fine bias value.

inline uint8_t **getDavis346BiasVoltage** (const *Davis346BiasVDAC* bias) const

Get voltage value in 52.4 milliVolt increments for a specific *voltage* bias.

Parameters

bias – voltage bias to query.

Returns

voltage value in 52.4 milliVolt increments, from 0 to 63.

inline void **setDavis346BiasVoltage** (const *Davis346BiasVDAC* bias, const uint8_t voltageValue)

Set specific *voltage* bias to a new value in 52.4 milliVolt increments.

Parameters

- **bias** – voltage bias to configure.
- **voltageValue** – voltage value in 52.4 milliVolt increments, from 0 to 63.

inline *std::*pair<uint8_t, uint8_t> **getDavis346BiasCoarseFine** (const *Davis346BiasCF* bias) const

Get low-level coarse/fine values for a specific bias.

Parameters

bias – bias to query.

Returns

coarse/fine values.

```
inline void setDavis346BiasCoarseFine (const Davis346BiasCF bias, const uint8_t coarse, const uint8_t fine)
```

Set specific bias to new low-level coarse/fine values.

Parameters

- **bias** – bias to configure.
- **coarse** – coarse bias value.
- **fine** – fine bias value.

```
inline uint8_t getCDavisBiasVoltage (const CDavisBiasVDAC bias) const
```

Get voltage value in 52.4 milliVolt increments for a specific *voltage* bias.

Parameters

bias – voltage bias to query.

Returns

voltage value in 52.4 milliVolt increments, from 0 to 63.

```
inline void setCDavisBiasVoltage (const CDavisBiasVDAC bias, const uint8_t voltageValue)
```

Set specific *voltage* bias to a new value in 52.4 milliVolt increments.

Parameters

- **bias** – voltage bias to configure.
- **voltageValue** – voltage value in 52.4 milliVolt increments, from 0 to 63.

```
inline std::pair<uint8_t, uint8_t> getCDavisBiasCoarseFine (const CDavisBiasCF bias) const
```

Get low-level coarse/fine values for a specific bias.

Parameters

bias – bias to query.

Returns

coarse/fine values.

```
inline void setCDavisBiasCoarseFine (const CDavisBiasCF bias, const uint8_t coarse, const uint8_t fine)
```

Set specific bias to new low-level coarse/fine values.

Parameters

- **bias** – bias to configure.
- **coarse** – coarse bias value.
- **fine** – fine bias value.

```
inline bool getCDavisAdjustOVG1Low () const
```

```
inline void setCDavisAdjustOVG1Low (const bool enable)
```

```
inline bool getCDavisAdjustOVG2Low () const
```

```
inline void setCDavisAdjustOVG2Low (const bool enable)
```

```
inline bool getCDavisAdjustTX2OVGHigh () const
```

```
inline void setCDavisAdjustTX2OVGHigh (const bool enable)
```

inline *std::chrono::microseconds* **getUSBEarlyPacketDelay** () const

Get value of USB early packet timeout.

Returns

timeout in microseconds.

inline void **setUSBEarlyPacketDelay** (const *std::chrono::microseconds* earlyPacketDelayUs)

Send data over USB early if this timeout is reached, instead of waiting on buffers being full. Timeout on device is in 125µs time-slices.

Parameters

earlyPacketDelayUs – timeout in microseconds.

Public Members

bool **muxHasStatistics**

Feature test: Multiplexer statistics support (event drops).

bool **dvsHasPixelFilter**

Feature test: DVS pixel-level filtering.

bool **dvsHasBackgroundActivityFilter**

Feature test: DVS Background Activity filter (and Refractory Period filter).

bool **dvsHasROIFilter**

Feature test: DVS ROI filter.

bool **dvsHasSkipFilter**

Feature test: DVS event skip filter.

bool **dvsHasPolarityFilter**

Feature test: DVS polarity suppression filter.

bool **dvsHasStatistics**

Feature test: DVS statistics support.

bool **apsHasGlobalShutter**

Feature test: APS supports Global Shutter.

bool **extInputHasGenerator**

Feature test: External Input module supports Signal-Generation.

cv::Mat **pixelHistogram**

cv::Mat **msvHistogram**

Public Static Functions

```
static inline auto findDevices (const std::string_view filterBySerialNumber = {}, loggerCallbackType logger = {})
```

Find connected *DAVIS* cameras.

Parameters

- **filterBySerialNumber** – only search for devices with this serial number
- **logger** – discovery logging callback, must be thread-safe

Returns

a descriptor structure describing a compatible device

Protected Functions

```
inline virtual void sendTimestampReset () override
```

Send a timestamp reset command to the device.

```
inline virtual void setTimestampOffset (const std::chrono::microseconds timestampOffset) override
```

Set a new timestamp offset value for the camera.

Parameters

timestampOffset – New timestamp offset value in microseconds.

```
inline std::chrono::microseconds getGeneratorHighTimeInternal () const
```

```
inline std::chrono::microseconds getGeneratorLowTimeInternal () const
```

```
inline void setCropAreaEventsInternal (cv::Rect cropAreaEvents)
```

```
inline void setCropAreaFramesInternal (cv::Rect cropAreaFrames)
```

Private Types

```
enum class CoarseFineBiasSex
```

Coarse-fine bias sex: true for 'N' type, false for 'P' type.

Values:

enumerator **N_TYPE**

enumerator **P_TYPE**

```
enum class CoarseFineBiasType
```

Coarse-fine bias type: true for 'Normal', false for 'Cascode'.

Values:

enumerator **NORMAL**

enumerator **CASCADE**

enum class **CoarseFineBiasCurrentLevel**

Coarse-fine bias current level: true for 'Normal', false for 'Low'.

Values:

enumerator **NORMAL**

enumerator **LOW**

enum class **ShiftedSourceBiasOperatingMode**

Shifted-source bias operating mode.

Values:

enumerator **SHIFTED_SOURCE**

Standard mode.

enumerator **HI_Z**

High impedance (driven from outside).

enumerator **TIED_TO_RAIL**

Tied to ground (SSN) or VDD (SSP).

enum class **ShiftedSourceBiasVoltageLevel**

Shifted-source bias voltage level.

Values:

enumerator **SPLIT_GATE**

Standard mode (200-400mV).

enumerator **SINGLE_DIODE**

Higher shifted-source voltage (one cascode).

enumerator **DOUBLE_DIODE**

Even higher shifted-source voltage (two cascodes).

Private Functions

inline void **shutdownCallback** ()

inline void **usbDataCallback** (const *std::span*<const uint8_t> data)

inline void **dataParserCallback** (*parser::ParsedData* data)

inline void **timeInitCallback** ()

inline void **sendVDACBias** (const int address)

inline void **sendCoarseFineBias** (const int address)

```
inline void sendShiftedSourceBias (const int address)

inline dv::Duration computeAutomaticExposure (const dv::Frame &frame)
```

Private Members

```
uint32_t mLogicVersion

uint32_t mLogicPatch

cv::Size mEventResolution

cv::Size mFrameResolution

parser::DAVIS::SensorModel mSensorModel

imu::ImuModel mImuModel

float mLogicClockActual

float mUSBClockActual

float mADCClockActual

std::unique_ptr<parser::DAVIS::Parser> mParser

std::atomic<dv::PixelArrangement> mDvsColorFilter

mutable std::mutex mConfigLock

cv::Rect mCropAreaEvents

cv::Rect mCropAreaFrames

std::array<VDACBias, 8> mVDACBiases = {}

std::array<CoarseFineBias, 35> mCoarseFineBiases = {}

ShiftedSourceBias mShiftedSourceNBias = {}

ShiftedSourceBias mShiftedSourcePBias = {}

bool mAutoExposureEnabled = {false}
```

```
AutoExposureCallback mAutoExposureCallback = {}
```

```
std::chrono::microseconds mAutoExposureLast = {1}
```

```
mutable std::mutex mCallbackConfigLock
```

```
std::atomic<bool> mIsRunning = {true}
```

```
std::atomic<bool> mTimestampMaster = {true}
```

```
struct dv::io::camera::DAVIS mInfo
```

```
struct dv::io::camera::DAVIS mAutoExposure
```

Private Static Functions

```
static inline constexpr uint16_t caerBiasVDACGenerate (const VDACBias vdacBias)
```

Transform VDAC bias structure into internal integer representation, suited for sending directly to the device via caerDeviceConfigSet().

Parameters

vdacBias – VDAC bias structure.

Returns

internal integer representation for device configuration.

```
static inline constexpr VDACBias caerBiasVDACParse (const uint16_t vdacBias)
```

Transform internal integer representation, as received by calls to caerDeviceConfigGet(), into a VDAC bias structure, for easier handling and understanding of the various parameters.

Parameters

vdacBias – internal integer representation from device.

Returns

VDAC bias structure.

```
static inline constexpr uint16_t caerBiasCoarseFineGenerate (const CoarseFineBias coarseFineBias)
```

Transform coarse-fine bias structure into internal integer representation, suited for sending directly to the device via caerDeviceConfigSet().

Parameters

coarseFineBias – coarse-fine bias structure.

Returns

internal integer representation for device configuration.

```
static inline constexpr CoarseFineBias caerBiasCoarseFineParse (const uint16_t coarseFineBias)
```

Transform internal integer representation, as received by calls to caerDeviceConfigGet(), into a coarse-fine bias structure, for easier handling and understanding of the various parameters.

Parameters

coarseFineBias – internal integer representation from device.

Returns

coarse-fine bias structure.

```
static inline constexpr uint16_t caerBiasShiftedSourceGenerate (const ShiftedSourceBias
                                                                shiftedSourceBias)
```

Transform shifted-source bias structure into internal integer representation, suited for sending directly to the device via caerDeviceConfigSet().

Parameters

shiftedSourceBias – shifted-source bias structure.

Returns

internal integer representation for device configuration.

```
static inline constexpr ShiftedSourceBias caerBiasShiftedSourceParse (const uint16_t shiftedSourceBias)
```

Transform internal integer representation, as received by calls to caerDeviceConfigGet(), into a shifted-source bias structure, for easier handling and understanding of the various parameters.

Parameters

shiftedSourceBias – internal integer representation from device.

Returns

shifted-source bias structure.

Private Static Attributes

```
static constexpr uint16_t MODULE_MULTIPLEXER = {0}
```

Module address: device-side Multiplexer configuration. The Multiplexer is responsible for mixing, times-tamping and outputting (via USB) the various event types generated by the device. It is also responsible for timestamp generation and synchronization.

```
static constexpr uint16_t MODULE_DVS = {1}
```

Module address: device-side DVS configuration. The DVS state machine handshakes with the chip's AER bus and gets the polarity events from it. It supports various configurable delays, as well as advanced filtering capabilities on the polarity events.

```
static constexpr uint16_t MODULE_APS = {2}
```

Module address: device-side APS (*Frame*) configuration. The APS (Active-Pixel-Sensor) is responsible for getting the normal, synchronous frame from the camera chip. It supports various options for very precise timing control, as well as Region of Interest imaging.

```
static constexpr uint16_t MODULE_IMU = {3}
```

Module address: device-side *IMU* (Inertial Measurement Unit) configuration. The *IMU* module connects to the external *IMU* chip and sends data on the device's movement in space. It can configure various options on the external chip, such as accelerometer range or gyroscope refresh rate.

```
static constexpr uint16_t MODULE_EXTERNAL_INPUT = {4}
```

Module address: device-side External Input (signal detector/generator) configuration. The External Input module is used to detect external signals on the external input jack and inject an event into the event stream when this happens. It can detect pulses of a specific length or rising and falling edges. On some systems, a signal generator module is also present, which can generate PWM-like pulsed signals with configurable timing.

```
static constexpr uint16_t MODULE_BIAS = {5}
```

Module address: device-side chip bias configuration. Shared with DAVIS_CONFIG_CHIP. This state machine is responsible for configuring the chip's bias generator.

```
static constexpr uint16_t MODULE_CHIP = {5}
```

Module address: device-side chip control configuration. Shared with DAVIS_CONFIG_BIAS. This state machine is responsible for configuring the chip's internal control shift registers, to set special options.

```
static constexpr uint16_t MODULE_SYSINFO = {6}
```

Module address: device-side system information. The system information module provides various details on the device, such as currently installed logic revision or clock speeds. All its parameters are read-only. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct caer_davis_info' documentation for more details on what information is available.

```
static constexpr uint16_t MODULE_USB = {9}
```

Module address: device-side USB output configuration. The USB output module forwards the data from the device and the FPGA/CPLD to the USB chip, usually a Cypress FX2 or FX3.

```
static constexpr uint16_t MUX_RUN = {0}
```

Parameter address for module DAVIS_CONFIG_MUX: run the Multiplexer state machine, which is responsible for mixing the various event types at the device level, timestamping them and outputting them via USB or other connectors.

```
static constexpr uint16_t MUX_TIMESTAMP_RUN = {1}
```

Parameter address for module DAVIS_CONFIG_MUX: run the Timestamp Generator inside the Multiplexer state machine, which will provide microsecond accurate timestamps to the events passing through.

```
static constexpr uint16_t MUX_TIMESTAMP_RESET = {2}
```

Parameter address for module DAVIS_CONFIG_MUX: reset the Timestamp Generator to zero. This also sends a reset pulse to all connected slave devices, resetting their timestamp too.

```
static constexpr uint16_t MUX_RUN_CHIP = {3}
```

Parameter address for module DAVIS_CONFIG_MUX: power up the chip's bias generator, enabling the chip to work.

```
static constexpr uint16_t MUX_DROP_EXTINPUT_ON_TRANSFER_STALL = {4}
```

Parameter address for module DAVIS_CONFIG_MUX: drop External Input events if the USB output FIFO is full, instead of having them pile up at the input FIFOs.

```
static constexpr uint16_t MUX_DROP_DVS_ON_TRANSFER_STALL = {5}
```

Parameter address for module DAVIS_CONFIG_MUX: drop DVS events if the USB output FIFO is full, instead of having them pile up at the input FIFOs.

```
static constexpr uint16_t MUX_HAS_STATISTICS = {80}
```

Parameter address for module DAVIS_CONFIG_MUX: read-only parameter, information about the presence of the statistics feature. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct caer_davis_info' documentation to get this information.

```
static constexpr uint16_t MUX_STATISTICS_EXTINPUT_DROPPED = {81}
```

Parameter address for module DAVIS_CONFIG_MUX: read-only parameter, representing the number of dropped External Input events on the device due to full USB buffers. This is a 64bit value, and should always be read using the function: caerDeviceConfigGet64().

static constexpr uint16_t **MUX_STATISTICS_DVS_DROPPED** = {83}

Parameter address for module DAVIS_CONFIG_MUX: read-only parameter, representing the number of dropped DVS events on the device due to full USB buffers. This is a 64bit value, and should always be read using the function: caerDeviceConfigGet64().

static constexpr uint16_t **DVS_SIZE_COLUMNS** = {0}

Parameter address for module DAVIS_CONFIG_DVS: read-only parameter, contains the X axis resolution of the DVS events returned by the camera. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct caer_davis_info' documentation to get proper size information that already considers the rotation and orientation settings.

static constexpr uint16_t **DVS_SIZE_ROWS** = {1}

Parameter address for module DAVIS_CONFIG_DVS: read-only parameter, contains the Y axis resolution of the DVS events returned by the camera. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct caer_davis_info' documentation to get proper size information that already considers the rotation and orientation settings.

static constexpr uint16_t **DVS_ORIENTATION_INFO** = {2}

Parameter address for module DAVIS_CONFIG_DVS: read-only parameter, contains information on the orientation of the X/Y axes, whether they should be inverted or not on the host when parsing incoming events. Bit 2: dvsInvertXY Bit 1: reserved Bit 0: reserved This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct caer_davis_info' documentation to get proper size information that already considers the rotation and orientation settings.

static constexpr uint16_t **DVS_RUN** = {3}

Parameter address for module DAVIS_CONFIG_DVS: run the DVS state machine and get polarity events from the chip by handshaking with its AER bus.

static constexpr uint16_t **DVS_WAIT_ON_TRANSFER_STALL** = {4}

Parameter address for module DAVIS_CONFIG_DVS: if the output FIFO for this module is full, stall the AER handshake with the chip and wait until it's free again, instead of just continuing the handshake and dropping the resulting events.

static constexpr uint16_t **DVS_EXTERNAL_AER_CONTROL** = {5}

Parameter address for module DAVIS_CONFIG_DVS: enable external AER control. This ensures the chip and the DVS pixel array are running, but doesn't do the handshake and leaves the ACK pin in high-impedance, to allow for an external system to take over the AER communication with the chip. DAVIS_CONFIG_DVS_RUN has to be turned off for this to work.

static constexpr uint16_t **DVS_HAS_PIXEL_FILTER** = {10}

Parameter address for module DAVIS_CONFIG_DVS: read-only parameter, information about the presence of the pixel filter feature. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct caer_davis_info' documentation to get this information.

static constexpr uint16_t **DVS_FILTER_PIXEL_0_ROW** = {11}

Parameter address for module DAVIS_CONFIG_DVS: the pixel filter completely suppresses up to eight pixels in the DVS array, filtering out all events produced by them. This is the pixel 0, Y axis setting.

```
static constexpr uint16_t DVS_FILTER_PIXEL_0_COLUMN = {12}
```

Parameter address for module DAVIS_CONFIG_DVS: the pixel filter completely suppresses up to eight pixels in the DVS array, filtering out all events produced by them. This is the pixel 0, X axis setting.

```
static constexpr uint16_t DVS_FILTER_PIXEL_1_ROW = {13}
```

Parameter address for module DAVIS_CONFIG_DVS: the pixel filter completely suppresses up to eight pixels in the DVS array, filtering out all events produced by them. This is the pixel 1, Y axis setting.

```
static constexpr uint16_t DVS_FILTER_PIXEL_1_COLUMN = {14}
```

Parameter address for module DAVIS_CONFIG_DVS: the pixel filter completely suppresses up to eight pixels in the DVS array, filtering out all events produced by them. This is the pixel 1, X axis setting.

```
static constexpr uint16_t DVS_FILTER_PIXEL_2_ROW = {15}
```

Parameter address for module DAVIS_CONFIG_DVS: the pixel filter completely suppresses up to eight pixels in the DVS array, filtering out all events produced by them. This is the pixel 2, Y axis setting.

```
static constexpr uint16_t DVS_FILTER_PIXEL_2_COLUMN = {16}
```

Parameter address for module DAVIS_CONFIG_DVS: the pixel filter completely suppresses up to eight pixels in the DVS array, filtering out all events produced by them. This is the pixel 2, X axis setting.

```
static constexpr uint16_t DVS_FILTER_PIXEL_3_ROW = {17}
```

Parameter address for module DAVIS_CONFIG_DVS: the pixel filter completely suppresses up to eight pixels in the DVS array, filtering out all events produced by them. This is the pixel 3, Y axis setting.

```
static constexpr uint16_t DVS_FILTER_PIXEL_3_COLUMN = {18}
```

Parameter address for module DAVIS_CONFIG_DVS: the pixel filter completely suppresses up to eight pixels in the DVS array, filtering out all events produced by them. This is the pixel 3, X axis setting.

```
static constexpr uint16_t DVS_FILTER_PIXEL_4_ROW = {19}
```

Parameter address for module DAVIS_CONFIG_DVS: the pixel filter completely suppresses up to eight pixels in the DVS array, filtering out all events produced by them. This is the pixel 4, Y axis setting.

```
static constexpr uint16_t DVS_FILTER_PIXEL_4_COLUMN = {20}
```

Parameter address for module DAVIS_CONFIG_DVS: the pixel filter completely suppresses up to eight pixels in the DVS array, filtering out all events produced by them. This is the pixel 4, X axis setting.

```
static constexpr uint16_t DVS_FILTER_PIXEL_5_ROW = {21}
```

Parameter address for module DAVIS_CONFIG_DVS: the pixel filter completely suppresses up to eight pixels in the DVS array, filtering out all events produced by them. This is the pixel 5, Y axis setting.

```
static constexpr uint16_t DVS_FILTER_PIXEL_5_COLUMN = {22}
```

Parameter address for module DAVIS_CONFIG_DVS: the pixel filter completely suppresses up to eight pixels in the DVS array, filtering out all events produced by them. This is the pixel 5, X axis setting.

```
static constexpr uint16_t DVS_FILTER_PIXEL_6_ROW = {23}
```

Parameter address for module DAVIS_CONFIG_DVS: the pixel filter completely suppresses up to eight pixels in the DVS array, filtering out all events produced by them. This is the pixel 6, Y axis setting.

static constexpr uint16_t **DVS_FILTER_PIXEL_6_COLUMN** = {24}

Parameter address for module DAVIS_CONFIG_DVS: the pixel filter completely suppresses up to eight pixels in the DVS array, filtering out all events produced by them. This is the pixel 6, X axis setting.

static constexpr uint16_t **DVS_FILTER_PIXEL_7_ROW** = {25}

Parameter address for module DAVIS_CONFIG_DVS: the pixel filter completely suppresses up to eight pixels in the DVS array, filtering out all events produced by them. This is the pixel 7, Y axis setting.

static constexpr uint16_t **DVS_FILTER_PIXEL_7_COLUMN** = {26}

Parameter address for module DAVIS_CONFIG_DVS: the pixel filter completely suppresses up to eight pixels in the DVS array, filtering out all events produced by them. This is the pixel 7, X axis setting.

static constexpr uint16_t **DVS_HAS_BACKGROUND_ACTIVITY_FILTER** = {30}

Parameter address for module DAVIS_CONFIG_DVS: read-only parameter, information about the presence of the background-activity filter feature. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct caer_davis_info' documentation to get this information.

static constexpr uint16_t **DVS_FILTER_BACKGROUND_ACTIVITY** = {31}

Parameter address for module DAVIS_CONFIG_DVS: enable the background-activity filter, which tries to remove events caused by transistor leakage, by rejecting uncorrelated events.

static constexpr uint16_t **DVS_FILTER_BACKGROUND_ACTIVITY_TIME** = {32}

Parameter address for module DAVIS_CONFIG_DVS: specify the time difference constant for the background-activity filter. Range: 0 - 4095, in 250µs units. Events that are correlated within this time-frame are let through, while others are filtered out.

static constexpr uint16_t **DVS_FILTER_REFRACTORY_PERIOD** = {33}

Parameter address for module DAVIS_CONFIG_DVS: enable the refractory period filter, which limits the firing rate of pixels. This is supported together with the background-activity filter.

static constexpr uint16_t **DVS_FILTER_REFRACTORY_PERIOD_TIME** = {34}

Parameter address for module DAVIS_CONFIG_DVS: specify the time constant for the refractory period filter. Range: 0 - 4095, in 250µs units. Pixels will be inhibited from generating new events during this time after the last even has fired.

static constexpr uint16_t **DVS_HAS_ROI_FILTER** = {40}

Parameter address for module DAVIS_CONFIG_DVS: read-only parameter, information about the presence of the ROI filter feature. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct caer_davis_info' documentation to get this information.

static constexpr uint16_t **DVS_FILTER_ROI_START_COLUMN** = {41}

Parameter address for module DAVIS_CONFIG_DVS: start position on the X axis for Region of Interest. Must be between 0 and DVS_SIZE_X-1, and be smaller or equal to DAVIS_CONFIG_DVS_FILTER_ROI_END_COLUMN.

static constexpr uint16_t **DVS_FILTER_ROI_START_ROW** = {42}

Parameter address for module DAVIS_CONFIG_DVS: start position on the Y axis for Region of Inter-

est. Must be between 0 and DVS_SIZE_Y-1, and be smaller or equal to DAVIS_CONFIG_DVS_FILTER_ROI_END_ROW.

static constexpr uint16_t DVS_FILTER_ROI_END_COLUMN = {43}

Parameter address for module DAVIS_CONFIG_DVS: end position on the X axis for Region of Interest. Must be between 0 and DVS_SIZE_X-1, and be greater or equal to DAVIS_CONFIG_DVS_FILTER_ROI_START_COLUMN.

static constexpr uint16_t DVS_FILTER_ROI_END_ROW = {44}

Parameter address for module DAVIS_CONFIG_DVS: end position on the Y axis for Region of Interest. Must be between 0 and DVS_SIZE_Y-1, and be greater or equal to DAVIS_CONFIG_DVS_FILTER_ROI_START_ROW.

static constexpr uint16_t DVS_HAS_SKIP_FILTER = {50}

Parameter address for module DAVIS_CONFIG_DVS: read-only parameter, information about the presence of the event skip filter feature. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct caer_davis_info' documentation to get this information.

static constexpr uint16_t DVS_FILTER_SKIP_EVENTS = {51}

Parameter address for module DAVIS_CONFIG_DVS: enable the event skip filter, which simply throws away one event every N events (decimation filter).

static constexpr uint16_t DVS_FILTER_SKIP_EVENTS_EVERY = {52}

Parameter address for module DAVIS_CONFIG_DVS: number of events to let through before skipping one. Range: 0 - 255 events.

static constexpr uint16_t DVS_HAS_POLARITY_FILTER = {60}

Parameter address for module DAVIS_CONFIG_DVS: read-only parameter, information about the presence of the polarity suppression filter feature. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct caer_davis_info' documentation to get this information.

static constexpr uint16_t DVS_FILTER_POLARITY_FLATTEN = {61}

Parameter address for module DAVIS_CONFIG_DVS: flatten all polarities to OFF (0).

static constexpr uint16_t DVS_FILTER_POLARITY_SUPPRESS = {62}

Parameter address for module DAVIS_CONFIG_DVS: suppress one of the two ON/OFF polarities completely. Use DAVIS_CONFIG_DVS_FILTER_POLARITY_IGNORE to select which.

static constexpr uint16_t DVS_FILTER_POLARITY_SUPPRESS_TYPE = {63}

Parameter address for module DAVIS_CONFIG_DVS: polarity to suppress (0=OFF, 1=ON). Use DAVIS_CONFIG_DVS_FILTER_POLARITY_IGNORE to enable.

static constexpr uint16_t DVS_HAS_STATISTICS = {80}

Parameter address for module DAVIS_CONFIG_DVS: read-only parameter, information about the presence of the statistics feature. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct caer_davis_info' documentation to get this information.

static constexpr uint16_t DVS_STATISTICS_EVENTS_ROW = {81}

Parameter address for module DAVIS_CONFIG_DVS: read-only parameter, representing the number of row event transactions completed on the device. This is a 64bit value, and should always be read using the function: caerDeviceConfigGet64().

static constexpr uint16_t **DVS_STATISTICS_EVENTS_COLUMN** = {83}

Parameter address for module DAVIS_CONFIG_DVS: read-only parameter, representing the number of column event transactions completed on the device. This is a 64bit value, and should always be read using the function: caerDeviceConfigGet64().

static constexpr uint16_t **DVS_STATISTICS_EVENTS_DROPPED** = {85}

Parameter address for module DAVIS_CONFIG_DVS: read-only parameter, representing the number of dropped transaction sequences on the device due to full buffers. This is a 64bit value, and should always be read using the function: caerDeviceConfigGet64().

static constexpr uint16_t **DVS_STATISTICS_FILTERED_PIXELS** = {87}

Parameter address for module DAVIS_CONFIG_DVS: read-only parameter, representing the number of dropped events due to the pixel filter. This is a 64bit value, and should always be read using the function: caerDeviceConfigGet64().

static constexpr uint16_t **DVS_STATISTICS_FILTERED_BACKGROUND_ACTIVITY** = {89}

Parameter address for module DAVIS_CONFIG_DVS: read-only parameter, representing the number of dropped events due to the background-activity filter. This is a 64bit value, and should always be read using the function: caerDeviceConfigGet64().

static constexpr uint16_t **DVS_STATISTICS_FILTERED_REFRACTORY_PERIOD** = {91}

Parameter address for module DAVIS_CONFIG_DVS: read-only parameter, representing the number of dropped events due to the refractory period filter. This is a 64bit value, and should always be read using the function: caerDeviceConfigGet64().

static constexpr uint16_t **APS_SIZE_COLUMNS** = {0}

Parameter address for module DAVIS_CONFIG_APS: read-only parameter, contains the X axis resolution of the APS frames returned by the camera. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct caer_davis_info' documentation to get proper size information that already considers the rotation and orientation settings.

static constexpr uint16_t **APS_SIZE_ROWS** = {1}

Parameter address for module DAVIS_CONFIG_APS: read-only parameter, contains the Y axis resolution of the APS frames returned by the camera. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct caer_davis_info' documentation to get proper size information that already considers the rotation and orientation settings.

static constexpr uint16_t **APS_ORIENTATION_INFO** = {2}

Parameter address for module DAVIS_CONFIG_APS: read-only parameter, contains information on the orientation of the X/Y axes, whether they should be inverted or not on the host when parsing incoming pixels, as well as if the X or Y axes need to be flipped when reading the pixels. Bit 2: apsInvertXY Bit 1: apsFlipX Bit 0: apsFlipY This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct caer_davis_info' documentation to get proper size information that already considers the rotation and orientation settings.

```
static constexpr uint16_t APS_COLOR_FILTER = {3}
```

Parameter address for module DAVIS_CONFIG_APS: read-only parameter, contains information on the type of color filter present on the device. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct caer_davis_info' documentation to get proper color filter information.

```
static constexpr uint16_t APS_RUN = {4}
```

Parameter address for module DAVIS_CONFIG_APS: enable the APS module and take intensity images of the scene. While this parameter is enabled, frames will be taken continuously. To slow down the frame-rate, see DAVIS_CONFIG_APS_FRAME_DELAY. To only take snapshots, see DAVIS_CONFIG_APS_SNAPSHOT.

```
static constexpr uint16_t APS_WAIT_ON_TRANSFER_STALL = {5}
```

Parameter address for module DAVIS_CONFIG_APS: if the output FIFO for this module is full, stall the APS state machine and wait until it's free again, instead of just dropping the pixels as they are being read out. This guarantees a complete frame readout, at the possible cost of slight timing differences between pixels. If disabled, incomplete frames may be transmitted and will then be dropped on the host, resulting in lower frame-rates, especially during high DVS traffic.

```
static constexpr uint16_t APS_HAS_GLOBAL_SHUTTER = {6}
```

Parameter address for module DAVIS_CONFIG_APS: read-only parameter, information about the presence of the global shutter feature. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct caer_davis_info' documentation to get this information.

```
static constexpr uint16_t APS_GLOBAL_SHUTTER = {7}
```

Parameter address for module DAVIS_CONFIG_APS: enable Global Shutter mode instead of Rolling Shutter. The Global Shutter eliminates motion artifacts, but is noisier than the Rolling Shutter (worse quality).

```
static constexpr uint16_t APS_START_COLUMN_0 = {8}
```

Parameter address for module DAVIS_CONFIG_APS: start position on the X axis for Region of Interest 0. Must be between 0 and APS_SIZE_X-1, and be smaller or equal to DAVIS_CONFIG_APS_END_COLUMN_0.

```
static constexpr uint16_t APS_START_ROW_0 = {9}
```

Parameter address for module DAVIS_CONFIG_APS: start position on the Y axis for Region of Interest 0. Must be between 0 and APS_SIZE_Y-1, and be smaller or equal to DAVIS_CONFIG_APS_END_ROW_0.

```
static constexpr uint16_t APS_END_COLUMN_0 = {10}
```

Parameter address for module DAVIS_CONFIG_APS: end position on the X axis for Region of Interest 0. Must be between 0 and APS_SIZE_X-1, and be greater or equal to DAVIS_CONFIG_APS_START_COLUMN_0.

```
static constexpr uint16_t APS_END_ROW_0 = {11}
```

Parameter address for module DAVIS_CONFIG_APS: end position on the Y axis for Region of Interest 0. Must be between 0 and APS_SIZE_Y-1, and be greater or equal to DAVIS_CONFIG_APS_START_ROW_0.

```
static constexpr uint16_t APS_EXPOSURE = {12}
```

Parameter address for module DAVIS_CONFIG_APS: frame exposure time. Range: 0-4194303, in microseconds (maximum ~4s). Very precise for Global Shutter, slightly less exact for Rolling Shutter due to column-based timing constraints.

static constexpr uint16_t **APS_FRAME_INTERVAL** = {13}

Parameter address for module DAVIS_CONFIG_APS: time between consecutive frames. Range: 0-8388607, in microseconds (maximum ~8s). This can be used to set a frame-rate. Please note the frame-rate is best-effort, and may not be met if readout and exposure times exceed this value.

static constexpr uint16_t **APS_TRANSFER** = {14}

Parameter address for module DAVIS_CONFIG_APS (only for CDAVIS chip): charge transfer time in ADCClock cycles.

static constexpr uint16_t **APS_RSFDSETTLE** = {15}

Parameter address for module DAVIS_CONFIG_APS (only for CDAVIS chip): Rolling Shutter FD settle time in ADCClock cycles.

static constexpr uint16_t **APS_GSPDRESET** = {16}

Parameter address for module DAVIS_CONFIG_APS (only for CDAVIS chip): Global Shutter PD reset time in ADCClock cycles.

static constexpr uint16_t **APS_GSRESETFALL** = {17}

Parameter address for module DAVIS_CONFIG_APS (only for CDAVIS chip): Global Shutter Reset Fall time in ADCClock cycles.

static constexpr uint16_t **APS_GSTXFALL** = {18}

Parameter address for module DAVIS_CONFIG_APS (only for CDAVIS chip): Global Shutter Transfer Fall time in ADCClock cycles.

static constexpr uint16_t **APS_GSFDRESET** = {19}

Parameter address for module DAVIS_CONFIG_APS (only for CDAVIS chip): Global Shutter FD reset time in ADCClock cycles.

static constexpr uint16_t **IMU_TYPE** = {0}

Parameter address for module DAVIS_CONFIG_IMU: read-only parameter, contains information on the type of *IMU* chip being used in this device: 0 - no *IMU* present 1 - InvenSense MPU 6050/6150 2 - InvenSense MPU 9250 This is reserved for internal use and should not be used by anything other than libcaer.

static constexpr uint16_t **IMU_ORIENTATION_INFO** = {1}

Parameter address for module DAVIS_CONFIG_IMU: read-only parameter, contains information on the orientation of the X/Y/Z axes, whether they should be flipped or not on the host when parsing incoming *IMU* data samples. Bit 2: imuFlipX Bit 1: imuFlipY Bit 0: imuFlipZ This is reserved for internal use and should not be used by anything other than libcaer. Generated *IMU* events are already properly flipped when returned to the user.

static constexpr uint16_t **IMU_RUN_ACCELEROMETER** = {2}

Parameter address for module DAVIS_CONFIG_IMU: enable the *IMU*'s accelerometer. This takes the *IMU* chip out of sleep.

```
static constexpr uint16_t IMU_RUN_GYROSCOPE = {3}
```

Parameter address for module DAVIS_CONFIG_IMU: enable the *IMU*'s gyroscope. This takes the *IMU* chip out of sleep.

```
static constexpr uint16_t IMU_RUN_TEMPERATURE = {4}
```

Parameter address for module DAVIS_CONFIG_IMU: enable the *IMU*'s temperature sensor. This takes the *IMU* chip out of sleep.

```
static constexpr uint16_t IMU_SAMPLE_RATE_DIVIDER = {5}
```

Parameter address for module DAVIS_CONFIG_IMU: this specifies the divider from the Gyroscope Output Rate used to generate the Sample Rate for the *IMU*. Valid values are from 0 to 255. The Sample Rate is generated like this: Sample Rate = Gyroscope Output Rate / (1 + DAVIS_CONFIG_IMU_SAMPLE_RATE_DIVIDER) where Gyroscope Output Rate = 8 kHz when DAVIS_CONFIG_IMU_DIGITAL_LOW_PASS_FILTER is disabled (set to 0 or 7), and 1 kHz when enabled. Note: the accelerometer output rate is 1 kHz. This means that for a Sample Rate greater than 1 kHz, the same accelerometer sample may be output multiple times.

```
static constexpr uint16_t IMU_ACCEL_DLPF = {6}
```

Parameter address for module DAVIS_CONFIG_IMU: this configures the digital low-pass filter for both the accelerometer and the gyroscope on InvenSense MPU 6050/6150 *IMU* devices, or for the accelerometer only on InvenSense MPU 9250. Valid values are from 0 to 7 and have the following meaning:

On InvenSense MPU 6050/6150: 0 - Accel: BW=260Hz, Delay=0ms, FS=1kHz - Gyro: BW=256Hz, Delay=0.98ms, FS=8kHz 1 - Accel: BW=184Hz, Delay=2.0ms, FS=1kHz - Gyro: BW=188Hz, Delay=1.9ms, FS=1kHz 2 - Accel: BW=94Hz, Delay=3.0ms, FS=1kHz - Gyro: BW=98Hz, Delay=2.8ms, FS=1kHz 3 - Accel: BW=44Hz, Delay=4.9ms, FS=1kHz - Gyro: BW=42Hz, Delay=4.8ms, FS=1kHz 4 - Accel: BW=21Hz, Delay=8.5ms, FS=1kHz - Gyro: BW=20Hz, Delay=8.3ms, FS=1kHz 5 - Accel: BW=10Hz, Delay=13.8ms, FS=1kHz - Gyro: BW=10Hz, Delay=13.4ms, FS=1kHz 6 - Accel: BW=5Hz, Delay=19.0ms, FS=1kHz - Gyro: BW=5Hz, Delay=18.6ms, FS=1kHz 7 - Accel: RESERVED, FS=1kHz - Gyro: RESERVED, FS=8kHz

On InvenSense MPU 9250: 0 - Accel: BW=218.1Hz, Delay=1.88ms, FS=1kHz 1 - Accel: BW=218.1Hz, Delay=1.88ms, FS=1kHz 2 - Accel: BW=99Hz, Delay=2.88ms, FS=1kHz 3 - Accel: BW=44.8Hz, Delay=4.88ms, FS=1kHz 4 - Accel: BW=21.2Hz, Delay=8.87ms, FS=1kHz 5 - Accel: BW=10.2Hz, Delay=16.83ms, FS=1kHz 6 - Accel: BW=5.05Hz, Delay=32.48ms, FS=1kHz 7 - Accel: BW=420Hz, Delay=1.38ms, FS=1kHz

```
static constexpr uint16_t IMU_ACCEL_FULL_SCALE = {7}
```

Parameter address for module DAVIS_CONFIG_IMU: select the full scale range of the accelerometer outputs. Valid values are: 0 - +- 2 g 1 - +- 4 g 2 - +- 8 g 3 - +- 16 g

```
static constexpr uint16_t IMU_GYRO_DLPF = {9}
```

Parameter address for module DAVIS_CONFIG_IMU: this configures the digital low-pass filter for the gyroscope on devices using the InvenSense MPU 9250. Valid values are from 0 to 7 and have the following meaning:

0 - Gyro: BW=250Hz, Delay=0.97ms, FS=8kHz 1 - Gyro: BW=184Hz, Delay=2.9ms, FS=1kHz 2 - Gyro: BW=92Hz, Delay=3.9ms, FS=1kHz 3 - Gyro: BW=41Hz, Delay=5.9ms, FS=1kHz 4 - Gyro: BW=20Hz, Delay=9.9ms, FS=1kHz 5 - Gyro: BW=10Hz, Delay=17.85ms, FS=1kHz 6 - Gyro: BW=5Hz, Delay=33.48ms, FS=1kHz 7 - Gyro: BW=3600Hz, Delay=0.17ms, FS=8kHz

static constexpr uint16_t **IMU_GYRO_FULL_SCALE** = {10}

Parameter address for module DAVIS_CONFIG_IMU: select the full scale range of the gyroscope outputs. Valid values are: 0 - +- 250 °/s 1 - +- 500 °/s 2 - +- 1000 °/s 3 - +- 2000 °/s

static constexpr uint16_t **EXTINPUT_RUN_DETECTOR** = {0}

Parameter address for module DAVIS_CONFIG_EXTINPUT: enable the signal detector module. It generates events when it sees certain types of signals, such as edges or pulses of a defined length, on the IN JACK signal. This can be useful to inject events into the event stream in response to external stimuli or controls, such as turning on a LED lamp.

static constexpr uint16_t **EXTINPUT_DETECT_RISING_EDGES** = {1}

Parameter address for module DAVIS_CONFIG_EXTINPUT: send a special EXTERNAL_INPUT_RISING_EDGE event when a rising edge is detected (transition from low voltage to high).

static constexpr uint16_t **EXTINPUT_DETECT_FALLING_EDGES** = {2}

Parameter address for module DAVIS_CONFIG_EXTINPUT: send a special EXTERNAL_INPUT_FALLING_EDGE event when a falling edge is detected (transition from high voltage to low).

static constexpr uint16_t **EXTINPUT_DETECT_PULSES** = {3}

Parameter address for module DAVIS_CONFIG_EXTINPUT: send a special EXTERNAL_INPUT_PULSE event when a pulse, of a specified, configurable polarity and length, is detected. See DAVIS_CONFIG_EXTINPUT_DETECT_PULSE_POLARITY and DAVIS_CONFIG_EXTINPUT_DETECT_PULSE_LENGTH for more details.

static constexpr uint16_t **EXTINPUT_DETECT_PULSE_POLARITY** = {4}

Parameter address for module DAVIS_CONFIG_EXTINPUT: the polarity the pulse must exhibit to be detected as such. '1' means active high; a pulse will start when the signal goes from low to high and will continue to be seen as the same pulse as long as it stays high. '0' means active low; a pulse will start when the signal goes from high to low and will continue to be seen as the same pulse as long as it stays low.

static constexpr uint16_t **EXTINPUT_DETECT_PULSE_LENGTH** = {5}

Parameter address for module DAVIS_CONFIG_EXTINPUT: the minimal length that a pulse must have to trigger the sending of a special event. Range: 1-1048575, in microseconds.

static constexpr uint16_t **EXTINPUT_HAS_GENERATOR** = {10}

Parameter address for module DAVIS_CONFIG_EXTINPUT: read-only parameter, information about the presence of the signal generator feature. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct caer_davis_info' documentation to get this information.

static constexpr uint16_t **EXTINPUT_RUN_GENERATOR** = {11}

Parameter address for module DAVIS_CONFIG_EXTINPUT: enable the signal generator module. It generates a PWM-like signal based on configurable parameters and outputs it on the OUT JACK signal.

static constexpr uint16_t **EXTINPUT_GENERATE_PULSE_POLARITY** = {12}

Parameter address for module DAVIS_CONFIG_EXTINPUT: polarity of the PWM-like signal to be generated. '1' means active high, '0' means active low.

```
static constexpr uint16_t EXTINPUT_GENERATE_PULSE_INTERVAL = {13}
```

Parameter address for module `DAVIS_CONFIG_EXTINPUT`: the interval between the start of two consecutive pulses. Range: 1-1048575, in microseconds. This must be bigger or equal to `DAVIS_CONFIG_EXTINPUT_GENERATE_PULSE_LENGTH`. To generate a signal with 50% duty cycle, this would have to be exactly double of `DAVIS_CONFIG_EXTINPUT_GENERATE_PULSE_LENGTH`.

```
static constexpr uint16_t EXTINPUT_GENERATE_PULSE_LENGTH = {14}
```

Parameter address for module `DAVIS_CONFIG_EXTINPUT`: the length a pulse stays active. Range: 1-1048575, in microseconds. This must be smaller or equal to `DAVIS_CONFIG_EXTINPUT_GENERATE_PULSE_INTERVAL`. To generate a signal with 50% duty cycle, this would have to be exactly half of `DAVIS_CONFIG_EXTINPUT_GENERATE_PULSE_INTERVAL`.

```
static constexpr uint16_t EXTINPUT_GENERATE_INJECT_ON_RISING_EDGE = {15}
```

Parameter address for module `DAVIS_CONFIG_EXTINPUT`: enables event injection when a rising edge occurs in the generated signal; a special event `EXTERNAL_GENERATOR_RISING_EDGE` is emitted into the event stream.

```
static constexpr uint16_t EXTINPUT_GENERATE_INJECT_ON_FALLING_EDGE = {16}
```

Parameter address for module `DAVIS_CONFIG_EXTINPUT`: enables event injection when a falling edge occurs in the generated signal; a special event `EXTERNAL_GENERATOR_FALLING_EDGE` is emitted into the event stream.

```
static constexpr uint16_t SYSINFO_LOGIC_VERSION = {0}
```

Parameter address for module `DAVIS_CONFIG_SYSINFO`: read-only parameter, the version of the logic currently running on the device's FPGA/CPLD. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct caer_davis_info' documentation to get this information.

```
static constexpr uint16_t SYSINFO_CHIP_IDENTIFIER = {1}
```

Parameter address for module `DAVIS_CONFIG_SYSINFO`: read-only parameter, an integer used to identify the different types of sensor chips used on the device. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct caer_davis_info' documentation to get this information.

```
static constexpr uint16_t SYSINFO_DEVICE_IS_MASTER = {2}
```

Parameter address for module `DAVIS_CONFIG_SYSINFO`: read-only parameter, whether the device is currently a timestamp master or slave when synchronizing multiple devices together. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct caer_davis_info' documentation to get this information.

```
static constexpr uint16_t SYSINFO_LOGIC_CLOCK = {3}
```

Parameter address for module `DAVIS_CONFIG_SYSINFO`: read-only parameter, the frequency in MHz at which the main FPGA/CPLD logic is running. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct caer_davis_info' documentation to get this information.

```
static constexpr uint16_t SYSINFO_ADC_CLOCK = {4}
```

Parameter address for module `DAVIS_CONFIG_SYSINFO`: read-only parameter, the frequency in MHz at which the FPGA/CPLD logic related to APS frame grabbing is running. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct caer_davis_info' documentation to get this information.

```
static constexpr uint16_t SYSINFO_USB_CLOCK = {5}
```

Parameter address for module `DAVIS_CONFIG_SYSINFO`: read-only parameter, the frequency in MHz at which the FPGA/CPLD logic related to USB data transmission is running. This is reserved for internal use and should not be used by anything other than `libcaer`.

```
static constexpr uint16_t SYSINFO_CLOCK_DEVIATION = {6}
```

Parameter address for module `DAVIS_CONFIG_SYSINFO`: read-only parameter, the deviation factor for the clocks. Due to how FX3 generates the clocks, which are then used by FPGA/CPLD, they are not integers but have a fractional part. This is reserved for internal use and should not be used by anything other than `libcaer`.

```
static constexpr uint16_t SYSINFO_LOGIC_PATCH = {7}
```

Parameter address for module `DAVIS_CONFIG_SYSINFO`: read-only parameter, the patch version of the logic currently running on the device's FPGA/CPLD. This is reserved for internal use and should not be used by anything other than `libcaer`.

```
static constexpr uint16_t USB_RUN = {0}
```

Parameter address for module `DAVIS_CONFIG_USB`: enable the USB FIFO module, which transfers the data from the FPGA/CPLD to the USB chip, to be then sent to the host. Turning this off will suppress any USB data communication!

```
static constexpr uint16_t USB_EARLY_PACKET_DELAY = {1}
```

Parameter address for module `DAVIS_CONFIG_USB`: the time delay after which a packet of data is committed to USB, even if it is not full yet (short USB packet). The value is in 125 μ s time-slices, corresponding to how USB schedules its operations (a value of 4 for example would mean waiting at most 0.5ms until sending a short USB packet to the host).

```
static constexpr uint16_t DAVIS240_CHIP_DIGITALMUX0 = {128}
```

Parameter address for module `DAVIS240_CONFIG_CHIP`: DAVIS240 chip configuration. These are for expert control and should never be used or changed unless for advanced debugging purposes. To change the Global Shutter configuration, please use `DAVIS_CONFIG_APS_GLOBAL_SHUTTER` instead. On DAVIS240B cameras, `DAVIS240_CONFIG_CHIP_SPECIALPIXELCONTROL` can be used to enable the test pixel array.

```
static constexpr uint16_t DAVIS240_CHIP_DIGITALMUX1 = {129}
```

```
static constexpr uint16_t DAVIS240_CHIP_DIGITALMUX2 = {130}
```

```
static constexpr uint16_t DAVIS240_CHIP_DIGITALMUX3 = {131}
```

```
static constexpr uint16_t DAVIS240_CHIP_ANALOGMUX0 = {132}
```

```
static constexpr uint16_t DAVIS240_CHIP_ANALOGMUX1 = {133}
```

```
static constexpr uint16_t DAVIS240_CHIP_ANALOGMUX2 = {134}
```

```
static constexpr uint16_t DAVIS240_CHIP_BIASMUX0 = {135}
```

```
static constexpr uint16_t DAVIS240_CHIP_RESETCALIBNEURON = {136}
```

```
static constexpr uint16_t DAVIS240_CHIP_TYPENCALIBNEURON = {137}
```

```
static constexpr uint16_t DAVIS240_CHIP_RESETTESTPIXEL = {138}
```

```
static constexpr uint16_t DAVIS240_CHIP_SPECIALPIXELCONTROL = {139}
```

```
static constexpr uint16_t DAVIS240_CHIP_AERNAROW = {140}
```

```
static constexpr uint16_t DAVIS240_CHIP_USEAOUT = {141}
```

```
static constexpr uint16_t DAVIS240_CHIP_GLOBAL_SHUTTER = {142}
```

```
static constexpr uint16_t DAVIS346_CHIP_DIGITALMUX0 = {128}
```

Parameter address for module DAVIS346_CONFIG_CHIP: DAVIS346 chip configuration. These are for expert control and should never be used or changed unless for advanced debugging purposes. To change the Global Shutter configuration, please use DAVIS_CONFIG_APS_GLOBAL_SHUTTER instead.

```
static constexpr uint16_t DAVIS346_CHIP_DIGITALMUX1 = {129}
```

```
static constexpr uint16_t DAVIS346_CHIP_DIGITALMUX2 = {130}
```

```
static constexpr uint16_t DAVIS346_CHIP_DIGITALMUX3 = {131}
```

```
static constexpr uint16_t DAVIS346_CHIP_ANALOGMUX0 = {132}
```

```
static constexpr uint16_t DAVIS346_CHIP_ANALOGMUX1 = {133}
```

```
static constexpr uint16_t DAVIS346_CHIP_ANALOGMUX2 = {134}
```

```
static constexpr uint16_t DAVIS346_CHIP_BIASMUX0 = {135}
```

```
static constexpr uint16_t DAVIS346_CHIP_RESETCALIBNEURON = {136}
```

```
static constexpr uint16_t DAVIS346_CHIP_TYPENCALIBNEURON = {137}
```

```
static constexpr uint16_t DAVIS346_CHIP_RESETTESTPIXEL = {138}
```

```
static constexpr uint16_t DAVIS346_CHIP_AERNAROW = {140}
```

```
static constexpr uint16_t DAVIS346_CHIP_USEAOUT = {141}
```

```
static constexpr uint16_t DAVIS346_CHIP_GLOBAL_SHUTTER = {142}
```

```
static constexpr uint16_t DAVIS346_CHIP_SELECTGRAYCOUNTER = {143}
```

```
static constexpr uint16_t DAVIS346_CHIP_TESTADC = {144}
```

```
static constexpr uint16_t CDAVIS_CHIP_DIGITALMUX0 = {128}
```

Parameter address for module CDAVIS_CONFIG_CHIP: CDAVIS chip configuration. These are for expert control and should never be used or changed unless for advanced debugging purposes. To change the Global Shutter configuration, please use DAVIS_CONFIG_APS_GLOBAL_SHUTTER instead.

```
static constexpr uint16_t CDAVIS_CHIP_DIGITALMUX1 = {129}
```

```
static constexpr uint16_t CDAVIS_CHIP_DIGITALMUX2 = {130}
```

```
static constexpr uint16_t CDAVIS_CHIP_DIGITALMUX3 = {131}
```

```
static constexpr uint16_t CDAVIS_CHIP_ANALOGMUX0 = {132}
```

```
static constexpr uint16_t CDAVIS_CHIP_ANALOGMUX1 = {133}
```

```
static constexpr uint16_t CDAVIS_CHIP_ANALOGMUX2 = {134}
```

```
static constexpr uint16_t CDAVIS_CHIP_BIASMUX0 = {135}
```

```
static constexpr uint16_t CDAVIS_CHIP_RESETCALIBNEURON = {136}
```

```
static constexpr uint16_t CDAVIS_CHIP_TYPENCALIBNEURON = {137}
```

```
static constexpr uint16_t CDAVIS_CHIP_RESETTESTPIXEL = {138}
```

```
static constexpr uint16_t CDAVIS_CHIP_AERNAROW = {140}
```

```
static constexpr uint16_t CDAVIS_CHIP_USEAOUT = {141}
```

```
static constexpr uint16_t CDAVIS_CHIP_SELECTGRAYCOUNTER = {143}
```

```
static constexpr uint16_t CDAVIS_CHIP_TESTADC = {144}
```

```
static constexpr uint16_t CDAVIS_CHIP_ADJUSTOVG1LO = {145}
```

```
static constexpr uint16_t CDAVIS_CHIP_ADJUSTOVG2LO = {146}

static constexpr uint16_t CDAVIS_CHIP_ADJUSTTX2OVG2HI = {147}

static constexpr uint16_t DAVIS240_BIAS_IFTHRBN = {15}

static constexpr uint16_t DAVIS240_BIAS_IFREFRBN = {16}

static constexpr uint16_t DAVIS240_BIAS_SSP = {20}

static constexpr uint16_t DAVIS240_BIAS_SSN = {21}

static constexpr uint16_t DAVIS346_BIAS_IFTHRBN = {27}

static constexpr uint16_t DAVIS346_BIAS_IFREFRBN = {26}

static constexpr uint16_t DAVIS346_BIAS_SSP = {35}

static constexpr uint16_t DAVIS346_BIAS_SSN = {36}

static constexpr uint16_t CDAVIS_BIAS_IFTHRBN = {9}

static constexpr uint16_t CDAVIS_BIAS_IFREFRBN = {8}

static constexpr uint16_t CDAVIS_BIAS_SSP = {35}

static constexpr uint16_t CDAVIS_BIAS_SSN = {36}

static constexpr uint16_t PID_DAVIS_FX2 = {0x841B}

static constexpr uint8_t FX2_FIRMWARE_REQUIRED_VERSION = {4}

static constexpr uint8_t FX2_LOGIC_REQUIRED_VERSION = {18}

static constexpr uint8_t FX2_LOGIC_MINIMUM_PATCH = {1}

static constexpr uint16_t PID_DAVIS_FX3 = {0x841A}

static constexpr uint8_t FX3_FIRMWARE_REQUIRED_VERSION = {6}

static constexpr uint8_t FX3_LOGIC_REQUIRED_VERSION = {18}
```

```
static constexpr uint8_t FX3_LOGIC_MINIMUM_PATCH = {1}

static constexpr size_t PIXEL_FILTER_MAX_SIZE = {8}

static constexpr uint32_t EXPOSURE_MAX = {(1 << 22) - 1}

static constexpr uint32_t FRAME_INTERVAL_MAX = {(1 << 23) - 1}

static constexpr uint32_t USB_EARLY_PACKET_DELAY_MAX = {(1 << 20) - 1}

static constexpr uint32_t EXT_INPUT_TIME_MAX = {(1 << 20) - 1}

static constexpr auto COMPATIBLE_CAMERA = [] (const uint16_t vid, const uint16_t
pid, [[maybe_unused]] const USBDeviceType deviceType) -> std::optional<CameraModel>
{if ((vid == VID_INIVATION) && ((pid == PID_DAVIS_FX2) || (pid ==
PID_DAVIS_FX3)) && (deviceType != USBDeviceType::FX3_GEN2)) {return
CameraModel::DAVIS;}return std::nullopt;}

static constexpr int AUTOEXPOSURE_HISTOGRAM_SIZE = {256}

static constexpr int AUTOEXPOSURE_HISTOGRAM_MSV_SIZE = {5}

static constexpr std::array<float, 2> AUTOEXPOSURE_HISTOGRAM_RANGE = {0, UINT8_MAX + 1}

static constexpr std::array<const float*, 1> AUTOEXPOSURE_HISTOGRAM_RANGES =
{AUTOEXPOSURE_HISTOGRAM_RANGE.data()}

static constexpr float AUTOEXPOSURE_LOW_BOUNDARY = {0.10f}

static constexpr float AUTOEXPOSURE_HIGH_BOUNDARY = {0.90f}

static constexpr float AUTOEXPOSURE_UNDEROVER_FRAC = {0.33f}

static constexpr float AUTOEXPOSURE_UNDEROVER_CORRECTION = {14000.0f}

static constexpr float AUTOEXPOSURE_MSV_CORRECTION = {100.0f}

static constexpr int
AUTOEXPOSURE_HISTOGRAM_BINS_LOW{static_cast<int>(AUTOEXPOSURE_LOW_BOUNDARY *
static_cast<float>(AUTOEXPOSURE_HISTOGRAM_SIZE))}

static constexpr int
AUTOEXPOSURE_HISTOGRAM_BINS_HIGH{static_cast<int>(AUTOEXPOSURE_HIGH_BOUNDARY *
static_cast<float>(AUTOEXPOSURE_HISTOGRAM_SIZE))}
```

class **DecompressionSupport**

Subclassed by *dv::io::compression::Lz4DecompressionSupport*, *dv::io::compression::NoneDecompressionSupport*, *dv::io::compression::ZstdDecompressionSupport*

Public Functions

inline explicit **DecompressionSupport** (const *CompressionType* type)

virtual **~DecompressionSupport** () = default

virtual void **decompress** (*std::vector<std::byte>* &source, *std::vector<std::byte>* &target) = 0

inline *CompressionType* **getCompressionType** () const

Private Members

CompressionType **mType**

struct **Depth**

#include </builds/inivation/dv/dv-processing/include/dv-processing/measurements/depth.hpp> A depth measurement structure that contains a timestamped measurement of depth.

Public Functions

inline **Depth** (int64_t timestamp, float depth)

Public Members

int64_t **mTimestamp**

UNIX Microsecond timestamp

float **mDepth**

Depth measurement value, expected to be in meters.

struct **DepthEventPacket** : public *flatbuffers::NativeTable*

Public Types

typedef *DepthEventPacketFlatbuffer* **TableType**

Public Functions

inline **DepthEventPacket** ()

inline **DepthEventPacket** (const *std::vector<DepthEvent>* &_elements)

Public Members

std::vector<DepthEvent> **elements**

Public Static Functions

static inline constexpr const char ***GetFullyQualifiedName** ()

Friends

inline friend *std::ostream* &**operator**<< (*std::ostream* &os, const *DepthEventPacket* &packet)

struct **DepthEventPacketBuilder**

Public Functions

inline void **add_elements** (*flatbuffers::Offset*<*flatbuffers::Vector*<const DepthEvent*>> elements)

inline explicit **DepthEventPacketBuilder** (*flatbuffers::FlatBufferBuilder* &_fbb)

DepthEventPacketBuilder &**operator**= (const *DepthEventPacketBuilder*&)

inline *flatbuffers::Offset*<*DepthEventPacketFlatbuffer*> **Finish** ()

Public Members

flatbuffers::FlatBufferBuilder &**fbb_**

flatbuffers::uoffset_t **start_**

struct **DepthEventPacketFlatbuffer** : private *flatbuffers::Table*

Public Types

typedef *DepthEventPacket* **NativeTableType**

Public Functions

inline const *flatbuffers::Vector*<const DepthEvent*> ***elements** () const

inline bool **Verify** (*flatbuffers::Verifier* &verifier) const

inline *DepthEventPacket* ***UnPack** (const *flatbuffers::resolver_function_t* *_resolver = nullptr) const

inline void **UnPackTo** (*DepthEventPacket* *_o, const *flatbuffers::resolver_function_t* *_resolver = nullptr) const

Public Static Functions

```
static inline const flatbuffers::TypeTable *MiniReflectTypeTable()

static inline constexpr const char *GetFullyQualifiedName()

static inline void UnPackToFrom(DepthEventPacket *_o, const DepthEventPacketFlatbuffer *_fb, const
    flatbuffers::resolver_function_t *_resolver = nullptr)

static inline flatbuffers::Offset<DepthEventPacketFlatbuffer> Pack(flatbuffers::FlatBufferBuilder &_fbb, const
    DepthEventPacket *_o, const
    flatbuffers::rehasher_function_t *_rehasher =
    nullptr)
```

Public Static Attributes

```
static constexpr const char *identifier = "DEVT"
```

```
struct DepthFrame : public flatbuffers::NativeTable
```

Public Types

```
typedef DepthFrameFlatbuffer TableType
```

Public Functions

```
inline DepthFrame()

inline DepthFrame(int64_t _timestamp, int16_t _sizeX, int16_t _sizeY, uint16_t _minDepth, uint16_t
    _maxDepth, uint16_t _step, const std::vector<uint16_t> &_depth)
```

Public Members

```
int64_t timestamp

int16_t sizeX

int16_t sizeY

uint16_t minDepth

uint16_t maxDepth

uint16_t step

std::vector<uint16_t> depth
```

Public Static Functions

```
static inline constexpr const char *GetFullyQualifiedName ()
```

Friends

```
inline friend std::ostream &operator<< (std::ostream &os, const DepthFrame &frame)
```

```
struct DepthFrameBuilder
```

Public Functions

```
inline void add_timestamp (int64_t timestamp)

inline void add_sizeX (int16_t sizeX)

inline void add_sizeY (int16_t sizeY)

inline void add_minDepth (uint16_t minDepth)

inline void add_maxDepth (uint16_t maxDepth)

inline void add_step (uint16_t step)

inline void add_depth (flatbuffers::Offset<flatbuffers::Vector<uint16_t>> depth)

inline explicit DepthFrameBuilder (flatbuffers::FlatBufferBuilder &fbb)

DepthFrameBuilder &operator= (const DepthFrameBuilder &)

inline flatbuffers::Offset<DepthFrameFlatbuffer> Finish ()
```

Public Members

```
flatbuffers::FlatBufferBuilder &fbb_

flatbuffers::uoffset_t start_
```

```
struct DepthFrameFlatbuffer : private flatbuffers::Table
```

#include </builds/inivation/dv/dv-processing/include/dv-processing/data/depth_frame_base.hpp> A frame containing pixel depth values in millimeters.

Public Types

```
typedef DepthFrame NativeTableType
```

Public Functions

```
inline int64_t timestamp () const
    Central timestamp (μs), corresponds to exposure midpoint.

inline int16_t sizeX () const
    Start of Frame (SOF) timestamp.
```

```

inline int16_t sizeY() const
    Y axis length in pixels.

inline uint16_t minDepth() const
    Minimum valid depth value.

inline uint16_t maxDepth() const
    Maximum valid depth value.

inline uint16_t step() const
    Depth step value, minimal depth distance that can be measured by the sensor setup.

inline const flatbuffers::Vector<uint16_t> *depth() const
    Depth values, unsigned 16bit integers, millimeters from the camera frame, following the OpenNI standard.
    Depth value of 0 should be considered an invalid value.

inline bool verify(flatbuffers::Verifier &verifier) const

inline DepthFrame *UnPack(const flatbuffers::resolver_function_t *_resolver = nullptr) const

inline void UnPackTo(DepthFrame *_o, const flatbuffers::resolver_function_t *_resolver = nullptr) const

```

Public Static Functions

```

static inline const flatbuffers::TypeTable *MiniReflectTypeTable()

static inline constexpr const char *GetFullyQualifiedName()

static inline void UnPackToFrom(DepthFrame *_o, const DepthFrameFlatbuffer *_fb, const
    flatbuffers::resolver_function_t *_resolver = nullptr)

static inline flatbuffers::Offset<DepthFrameFlatbuffer> Pack(flatbuffers::FlatBufferBuilder &_fbb, const
    DepthFrame *_o, const
    flatbuffers::rehasher_function_t *_rehasher =
    nullptr)

```

Public Static Attributes

```

static constexpr const char *identifier = "DFRM"

```

struct **DeviceDescriptor**

#include </builds/inivation/dv/dv-processing/include/dv-processing/io/camera/usb_device.hpp> Structure to uniquely describe a USB device connected to the current host.

Public Members

```

std::string serialNumber

```

```

uint16_t vid

```

```

uint16_t pid

```

USBDeviceType **deviceType**

CameraModel **cameraModel**

uint8_t busNumber

uint8_t devAddress

uint8_t firmwareVersion

struct **DirectoryError**

Public Types

using **Info** = *std::filesystem::path*

Public Static Functions

static inline *std::string* **format** (const *Info* &info)

struct **DirectoryNotFound**

Public Types

using **Info** = *std::filesystem::path*

Public Static Functions

static inline *std::string* **format** (const *Info* &info)

class **DVS128** : public *dv::io::camera::USBDevice*, public *dv::io::camera::SyncCameraInputBase*

Public Types

enum class **Bias**

Values:

enumerator **CAS**

enumerator **INJGND**

enumerator **REQPD**

enumerator **PUX**

enumerator **DIFF_OFF**

enumerator **REQ**

enumerator **REFR**

enumerator **PUY**

enumerator **DIFF_ON**

enumerator **DIFF**

enumerator **FOLL**

enumerator **PR**

Public Functions

inline explicit **DVS128** (loggerCallbackType deviceLogger = {}, const LogLevel deviceLogLevel = LogLevel::LVL_WARNING)

Open the first *DVS128* camera that can be found. Throws if device cannot be opened.

Parameters

- **deviceLogger** – per-device logging callback, must be thread-safe
- **deviceLogLevel** – initial log-level

inline explicit **DVS128** (const *std::string_view* filterBySerialNumber, loggerCallbackType deviceLogger = {}, const LogLevel deviceLogLevel = LogLevel::LVL_WARNING)

Open the *DVS128* camera with the specified serial number. Throws if device cannot be opened.

Parameters

- **filterBySerialNumber** – serial number to search for
- **deviceLogger** – per-device logging callback, must be thread-safe
- **deviceLogLevel** – initial log-level

inline explicit **DVS128** (const DeviceDescriptor &deviceToOpen, loggerCallbackType deviceLogger = {}, const LogLevel deviceLogLevel = LogLevel::LVL_WARNING)

Open the *DVS128* camera corresponding to the specified descriptor. Throws if device cannot be opened.

Parameters

- **deviceToOpen** – device descriptor structure
- **deviceLogger** – per-device logging callback, must be thread-safe
- **deviceLogLevel** – initial log-level

inline **~DVS128** () override

inline virtual *std::string* **getCameraName**() const override

Get camera name, which is a combination of the camera model and the serial number.

Returns

String containing the camera model and serial number separated by an underscore character.

inline virtual *std::optional<cv::Size>* **getEventResolution**() const override

Get event stream resolution.

Returns

Event stream resolution, *std::nullopt* if event stream is unavailable.

inline virtual *std::optional<cv::Size>* **getFrameResolution**() const override

Retrieve frame stream resolution.

Returns

Frame stream resolution or *std::nullopt* if the frame stream is not available.

inline virtual *imu::ImuModel* **getImuModel**() const override

Return *IMU* model used on device.

Returns

IMU model in use.

inline virtual float **getPixelPitch**() const override

Return pixel pitch distance for the connected camera model. The value is returned in meters.

Returns

Pixel pitch distance in meters according to the connected device.

inline virtual bool **isMaster**() const override

Report if this camera is a clock synchronization master.

Returns

true if clock master, false otherwise.

inline void **setMaster**(const bool master)

Control clock master functionality of this camera. By default, this is enabled and allows the camera to receive triggers on its IN pin. If disabled, the camera will expect a clock synchronization signal on its IN pin and act as a secondary camera.

Parameters

master – true if clock synchronization master, false if secondary camera

inline virtual bool **isEventStreamAvailable**() const override

Check whether event stream is available.

Returns

True if event stream is available, false otherwise.

inline virtual bool **isFrameStreamAvailable**() const override

Check whether frame stream is available.

Returns

True if frame stream is available, false otherwise.

inline virtual bool **isImuStreamAvailable**() const override

Check whether *IMU* data is available.

Returns

True if *IMU* data stream is available, false otherwise.

inline virtual bool **isTriggerStreamAvailable** () const override

Check whether trigger data is available.

Returns

True if trigger data stream is available, false otherwise.

inline virtual bool **isRunning** () const override

Check whether any input data streams have terminated. For a live camera this should check if the device is still connected and functioning, while for a recording file this should check if any of the data streams have reached end-of-file (EOF). For a network input, this indicates the network stream is still connected.

Returns

True if data read on all streams is still possible, false otherwise.

inline virtual bool **isRunning** (const *std::string_view* streamName) const override

Check whether the input data stream with the specified name is still active.

Returns

True if data read on this stream is possible, false otherwise.

inline virtual bool **isRunningAny** () const override

Check whether any input data streams are still available. For a live camera this should check if the device is still connected and functioning and at least one data stream is active (different than *isRunning()*), while for a recording file this should check if any of the data streams have not yet reached end-of-file (EOF) and are still readable. For a network input, this indicates the network stream is still connected.

Returns

True if data read on at least one stream is still possible, false otherwise.

inline virtual *std::chrono::microseconds* **getTimeInterval** () const override

Get the time interval for data commit.

Returns

Time interval in microseconds.

inline virtual void **setTimeInterval** (const *std::chrono::microseconds* timeInterval) override

Set a new time interval value for data commit. Data is put in the queues for *getNextEventBatch()*, *readNext()*, ... at this interval's rate.

Parameters

timeInterval – New time interval value in microseconds.

inline virtual *std::chrono::microseconds* **getTimestampOffset** () const override

Get the timestamp offset.

Returns

Absolute timestamp offset value in microseconds.

inline uint32_t **getBias** (const *Bias* bias) const

Get current value of specified bias configuration.

Parameters

bias – name of bias

Returns

current bias value

inline void **setBias** (const *Bias* bias, const uint32_t value)

Set new value for specified bias configuration.

Parameters

- **bias** – name of bias
- **value** – new bias value

inline virtual bool **getFlipHorizontal** () const override

Status of horizontal events flip.

Returns

status of horizontal events flip.

inline virtual void **setFlipHorizontal** (const bool flipHorizontal) override

Flip events horizontally.

Parameters

flipHorizontalEvents – flip events horizontally.

inline virtual bool **getFlipVertical** () const override

Status of vertical events flip.

Returns

status of vertical events flip.

inline virtual void **setFlipVertical** (const bool flipVertical) override

Flip events vertically.

Parameters

flipVerticalEvents – flip events vertically.

inline virtual Flatten **getFlatten** () const override

Status of event polarity flattening.

Returns

status of event polarity flattening.

inline virtual void **setFlatten** (const Flatten flatten) override

Flatten events polarity.

Parameters

flattenEvents – flattening mode.

inline virtual cv::Rect **getCropArea** () const override

Get events Region of Interest (ROI).

Returns

get events Region of Interest (ROI).

inline virtual void **setCropArea** (const cv::Rect cropArea) override

Set events Region of Interest (ROI). Usually hardware accelerated.

Parameters

cropAreaEvents – region of interest (ROI) position and size.

Public Static Functions

static inline auto **findDevices** (const *std::string_view* filterBySerialNumber = {}, loggerCallbackType logger = {})

Find connected *DVS128* cameras.

Parameters

- **filterBySerialNumber** – only search for devices with this serial number

- **logger** – discovery logging callback, must be thread-safe

Returns

a descriptor structure describing a compatible device

Protected Functions

inline virtual void **sendTimestampReset** () override

Send a timestamp reset command to the device.

inline virtual void **setTimestampOffset** (const *std::chrono::microseconds* timestampOffset) override

Set a new timestamp offset value for the camera.

Parameters

timestampOffset – New timestamp offset value in microseconds.

Private Functions

inline void **shutdownCallback** ()

inline void **usbDataCallback** (const *std::span*<const uint8_t> data)

inline void **dataParserCallback** (*parser::ParsedData* data)

inline void **timeInitCallback** ()

inline void **setBiasInternal** (const *Bias* bias, const uint32_t value, const bool send)

Private Members

parser::DVS128::Parser **mParser**

mutable *std::mutex* **mConfigLock**

std::array<*std::array*<uint8_t, 3>, 12> **mBiases** = {}

Flatten **mFlatten** = {Flatten::NONE}

cv::Rect **mCropArea**

mutable *std::mutex* **mCallbackConfigLock**

std::atomic<bool> **mIsRunning** = {true}

std::atomic<bool> **mTimestampMaster** = {true}

Private Static Attributes

static constexpr uint16_t **PID_DVS128** = {0x8400}

```
static constexpr uint8_t FIRMWARE_REQUIRED_VERSION = {14}

static constexpr uint8_t DATA_ENDPOINT = {0x86}

static constexpr uint8_t VENDOR_REQUEST_START_TRANSFER = {0xB3}

static constexpr uint8_t VENDOR_REQUEST_STOP_TRANSFER = {0xB4}

static constexpr uint8_t VENDOR_REQUEST_SEND_BIASES = {0xB8}

static constexpr uint8_t VENDOR_REQUEST_RESET_TS = {0xBB}

static constexpr uint8_t VENDOR_REQUEST_RESET_ARRAY = {0xBD}

static constexpr uint8_t VENDOR_REQUEST_TS_MASTER = {0xBE}

static constexpr auto COMPATIBLE_CAMERA = [] (const uint16_t vid, const uint16_t
pid, [[maybe_unused]] const USBDeviceType deviceType) -> std::optional<CameraModel>
{if ((vid == VID_INIVATION) && (pid == PID_DVS128)) {return
CameraModel::DVS128;}return std::nullopt;}
```

```
class DVXplorer : public dv::io::camera::USBDevice, public dv::io::camera::SyncCameraInputBase
```

Public Types

```
enum class SubSample
```

Values:

```
enumerator EVERY_PIXEL
```

```
enumerator EVERY_SECOND
```

```
enumerator BAND_OF_TWO
```

```
enumerator EVERY_FOURTH
```

```
enumerator BAND_OF_FOUR
```

```
enumerator BIN_1010_0000
```

```
enumerator BIN_1100_0000
```

```
enumerator EVERY_EIGHTH
```

enum class **ReadoutFPS**

Values:

enumerator **CONSTANT_100**

enumerator **CONSTANT_200**

enumerator **CONSTANT_500**

enumerator **CONSTANT_1000**

enumerator **CONSTANT_LOSSY_2000**

enumerator **CONSTANT_LOSSY_5000**

enumerator **CONSTANT_LOSSY_10000**

enumerator **VARIABLE_2000**

enumerator **VARIABLE_5000**

enumerator **VARIABLE_10000**

enumerator **VARIABLE_15000**

Public Functions

inline explicit **DVXplorer** (loggerCallbackType deviceLogger = {}, const LogLevel deviceLogLevel = LogLevel::LVL_WARNING)

Open the first *DVXplorer* camera that can be found. Throws if device cannot be opened.

Parameters

- **deviceLogger** – per-device logging callback, must be thread-safe
- **deviceLogLevel** – initial log-level

inline explicit **DVXplorer** (const *std::string_view* filterBySerialNumber, loggerCallbackType deviceLogger = {}, const LogLevel deviceLogLevel = LogLevel::LVL_WARNING)

Open the *DVXplorer* camera with the specified serial number. Throws if device cannot be opened.

Parameters

- **filterBySerialNumber** – serial number to search for
- **deviceLogger** – per-device logging callback, must be thread-safe
- **deviceLogLevel** – initial log-level

```
inline explicit DVXplorer (const DeviceDescriptor &deviceToOpen, loggerCallbackType deviceLogger = {},
                           const LogLevel deviceLogLevel = LogLevel::LVL_WARNING)
```

Open the *DVXplorer* camera corresponding to the specified descriptor. Throws if device cannot be opened.

Parameters

- **deviceToOpen** – device descriptor structure
- **deviceLogger** – per-device logging callback, must be thread-safe
- **deviceLogLevel** – initial log-level

```
inline ~DVXplorer () override
```

```
inline virtual std::string getCameraName () const override
```

Get camera name, which is a combination of the camera model and the serial number.

Returns

String containing the camera model and serial number separated by an underscore character.

```
inline uint32_t getLogicVersion () const
```

Get camera FPGA logic version.

Returns

camera FPGA logic version

```
inline uint32_t getLogicPatchLevel () const
```

Get camera FPGA logic patch level.

Returns

camera FPGA logic patch level

```
inline virtual std::optional<cv::Size> getEventResolution () const override
```

Get event stream resolution.

Returns

Event stream resolution, `std::nullopt` if event stream is unavailable.

```
inline virtual std::optional<cv::Size> getFrameResolution () const override
```

Retrieve frame stream resolution.

Returns

Frame stream resolution or `std::nullopt` if the frame stream is not available.

```
inline virtual imu::ImuModel getImuModel () const override
```

Return *IMU* model used on device.

Returns

IMU model in use.

```
inline virtual float getPixelPitch () const override
```

Return pixel pitch distance for the connected camera model. The value is returned in meters.

Returns

Pixel pitch distance in meters according to the connected device.

```
inline virtual bool isMaster () const override
```

Report if this camera is a clock synchronization master.

Returns

true if clock master, false otherwise.

inline virtual bool **isEventStreamAvailable** () const override

Check whether event stream is available.

Returns

True if event stream is available, false otherwise.

inline virtual bool **isFrameStreamAvailable** () const override

Check whether frame stream is available.

Returns

True if frame stream is available, false otherwise.

inline virtual bool **isImuStreamAvailable** () const override

Check whether *IMU* data is available.

Returns

True if *IMU* data stream is available, false otherwise.

inline virtual bool **isTriggerStreamAvailable** () const override

Check whether trigger data is available.

Returns

True if trigger data stream is available, false otherwise.

inline virtual bool **isRunning** () const override

Check whether any input data streams have terminated. For a live camera this should check if the device is still connected and functioning, while for a recording file this should check if any of the data streams have reached end-of-file (EOF). For a network input, this indicates the network stream is still connected.

Returns

True if data read on all streams is still possible, false otherwise.

inline virtual bool **isRunning** (const *std::string_view* streamName) const override

Check whether the input data stream with the specified name is still active.

Returns

True if data read on this stream is possible, false otherwise.

inline virtual bool **isRunningAny** () const override

Check whether any input data streams are still available. For a live camera this should check if the device is still connected and functioning and at least one data stream is active (different than *isRunning()*), while for a recording file this should check if any of the data streams have not yet reached end-of-file (EOF) and are still readable. For a network input, this indicates the network stream is still connected.

Returns

True if data read on at least one stream is still possible, false otherwise.

inline virtual *std::chrono::microseconds* **getTimeInterval** () const override

Get the time interval for data commit.

Returns

Time interval in microseconds.

inline virtual void **setTimeInterval** (const *std::chrono::microseconds* timeInterval) override

Set a new time interval value for data commit. Data is put in the queues for *getNextEventBatch()*, *readNext()*, ... at this interval's rate.

Parameters

timeInterval – New time interval value in microseconds.

inline virtual *std::chrono::microseconds* **getTimestampOffset** () const override

Get the timestamp offset.

Returns

Absolute timestamp offset value in microseconds.

inline bool **isEventsRunning** () const

Report if the event output is running.

Returns

true if events active, false otherwise.

inline void **setEventsRunning** (const bool run)

Enable or disable the event output.

Parameters

run – whether to enable the event output or not.

inline virtual bool **getFlipHorizontal** () const override

Status of horizontal events flip.

Returns

status of horizontal events flip.

inline virtual void **setFlipHorizontal** (const bool flipHorizontal) override

Flip events horizontally.

Parameters

flipHorizontalEvents – flip events horizontally.

inline virtual bool **getFlipVertical** () const override

Status of vertical events flip.

Returns

status of vertical events flip.

inline virtual void **setFlipVertical** (const bool flipVertical) override

Flip events vertically.

Parameters

flipVerticalEvents – flip events vertically.

inline virtual Flatten **getFlatten** () const override

Status of event polarity flattening.

Returns

status of event polarity flattening.

inline virtual void **setFlatten** (const Flatten flatten) override

Flatten events polarity.

Parameters

flattenEvents – flattening mode.

inline *SubSample* **getSubSampleHorizontal** () const

Get horizontal subsampling mode.

Returns

current horizontal subsampling mode.

inline void **setSubSampleHorizontal** (const *SubSample* subSampleHorizontal)

Set horizontal subsampling mode. *DVXplorer* Lite only supports EVERY_PIXEL, EVERY_SECOND and EVERY_FOURTH.

Parameters

subSampleHorizontal – new horizontal subsampling mode.

inline *SubSample* **getSubSampleVertical** () const

Get vertical subsampling mode.

Returns

current vertical subsampling mode.

inline void **setSubSampleVertical** (const *SubSample* subSampleVertical)

Set vertical subsampling mode. *DVXplorer* Lite only supports EVERY_PIXEL, EVERY_SECOND and EVERY_FOURTH.

Parameters

subSampleVertical – new vertical subsampling mode.

inline bool **getDualBinning** () const

Report status of dual-binning feature. Always false for *DVXplorer* Lite.

Returns

true if enabled, false otherwise.

inline void **setDualBinning** (const bool dualBinning)

Enable dual-binning, maps 2x2 pixel blocks to one pixel address. Not available on *DVXplorer* Lite.

Parameters

dualBinning – true to enable, false to disable.

inline bool **getGlobalHold** () const

Report status of global hold feature.

Returns

true if enabled, false otherwise.

inline void **setGlobalHold** (const bool globalHold)

Enable or disable global hold feature. Default is enabled. For some applications like LED tracking, setting this to false can help.

Parameters

globalHold – true to enable, false to disable.

inline bool **getGlobalReset** () const

Report status of global reset feature.

Returns

true if enabled, false otherwise.

inline void **setGlobalReset** (const bool globalReset)

Enable or disable global reset feature.

Parameters

globalReset – true to enable, false to disable.

inline *ReadoutFPS* **getReadoutFPS** () const

Get currently set event-frame readout frequency.

Returns

event-frame readout frequency.

inline void **setReadoutFPS** (const *ReadoutFPS* readoutFps)

Set frequency of event-frame readouts on sensor. CONSTANT frequencies are guaranteed fixed frequencies with no data loss. CONSTANT_LOSSY are guaranteed fixed frequencies but will cut off a readout to respect the timing if too much data is present, resulting in data loss. VARIABLE are best-effort frequencies that will change if lots of data is present, slowing down, but without data loss.

Parameters

readoutFps – event-frame readout frequency.

inline uint8_t **getContrastThresholdOn** () const

Get the contrast threshold for ON polarity event generation.

Returns

ON contrast threshold, from 0 to 17.

inline void **setContrastThresholdOn** (const uint8_t contrastThresholdOn)

Set the contrast threshold for ON polarity event generation. Valid values from 0 to 17.

Parameters

contrastThresholdOn – ON contrast threshold, from 0 to 17.

inline uint8_t **getContrastThresholdOff** () const

Get the contrast threshold for OFF polarity event generation.

Returns

OFF contrast threshold, from 0 to 17.

inline void **setContrastThresholdOff** (const uint8_t contrastThresholdOff)

Set the contrast threshold for OFF polarity event generation. Valid values from 0 to 17.

Parameters

contrastThresholdOff – OFF contrast threshold, from 0 to 17.

inline virtual cv::Rect **getCropArea** () const override

Get events Region of Interest (ROI).

Returns

get events Region of Interest (ROI).

inline virtual void **setCropArea** (const cv::Rect cropArea) override

Set events Region of Interest (ROI). Usually hardware accelerated.

Parameters

cropAreaEvents – region of interest (ROI) position and size.

inline bool **isImuRunningAccelerometer** () const

Status of *IMU* accelerometer.

Returns

true if enabled, false otherwise.

inline void **setImuRunningAccelerometer** (const bool run)

Enable or disable *IMU* accelerometer.

Parameters

run – true to enable, false to disable.

inline bool **isImuRunningGyroscope** () const

Status of *IMU* gyroscope.

Returns

true if enabled, false otherwise.

inline void **setImuRunningGyroscope** (const bool run)

Enable or disable *IMU* gyroscope.

Parameters

run – true to enable, false to disable.

inline bool **isImuRunningTemperature** () const

Status of *IMU* temperature measurement.

Returns

true if enabled, false otherwise.

inline void **setImuRunningTemperature** (const bool run)

Enable or disable *IMU* temperature measurement.

Parameters

run – true to enable, false to disable.

inline bool **getIMUFlipX** () const

Status of *IMU* X axis flipping.

Returns

true if enabled, false otherwise.

inline void **setIMUFlipX** (const bool flipX)

Enable or disable *IMU* X axis flipping. Will negate (flip) all returned X axis values.

Parameters

flipX – true to enable, false to disable.

inline bool **getIMUFlipY** () const

Status of *IMU* Y axis flipping.

Returns

true if enabled, false otherwise.

inline void **setIMUFlipY** (const bool flipY)

Enable or disable *IMU* Y axis flipping. Will negate (flip) all returned Y axis values.

Parameters

flipY – true to enable, false to disable.

inline bool **getIMUFlipZ** () const

Status of *IMU* Z axis flipping.

Returns

true if enabled, false otherwise.

inline void **setIMUFlipZ** (const bool flipZ)

Enable or disable *IMU* Z axis flipping. Will negate (flip) all returned Z axis values.

Parameters

flipZ – true to enable, false to disable.

```
inline imu::BoschBMI160AccelDataRate getImuAccelDataRate () const
```

Get current *IMU* accelerometer data rate.

Returns

accelerometer data rate.

```
inline void setImuAccelDataRate (const imu::BoschBMI160AccelDataRate dataRate)
```

Set *IMU* accelerometer data rate.

Parameters

dataRate – accelerometer data rate.

```
inline imu::BoschBMI160AccelFilter getImuAccelFilter () const
```

Get current *IMU* accelerometer filter setting.

Returns

accelerometer filter setting.

```
inline void setImuAccelFilter (const imu::BoschBMI160AccelFilter filter)
```

Set *IMU* accelerometer filter setting.

Parameters

filter – accelerometer filter setting.

```
inline imu::BoschBMI160AccelRange getImuAccelRange () const
```

Get current *IMU* accelerometer range.

Returns

accelerometer range.

```
inline void setImuAccelRange (const imu::BoschBMI160AccelRange range)
```

Set *IMU* accelerometer range.

Parameters

range – accelerometer range.

```
inline imu::BoschBMI160GyroDataRate getImuGyroDataRate () const
```

Get current *IMU* gyroscope data rate.

Returns

gyroscope data rate.

```
inline void setImuGyroDataRate (const imu::BoschBMI160GyroDataRate dataRate)
```

Set *IMU* gyroscope data rate.

Parameters

dataRate – gyroscope data rate.

```
inline imu::BoschBMI160GyroFilter getImuGyroFilter () const
```

Get current *IMU* gyroscope filter setting.

Returns

gyroscope filter setting.

```
inline void setImuGyroFilter (const imu::BoschBMI160GyroFilter filter)
```

Set *IMU* gyroscope filter setting.

Parameters

filter – gyroscope filter setting.

inline *imu::BoschBMI160GyroRange* **getImuGyroRange** () const

Get current *IMU* gyroscope range.

Returns

gyroscope range.

inline void **setImuGyroRange** (const *imu::BoschBMI160GyroRange* range)

Set *IMU* gyroscope range.

Parameters

range – gyroscope range.

inline bool **isDetectorRunning** () const

Report status of external signal detector.

Returns

true if running, false otherwise.

inline void **setDetectorRunning** (const bool run)

Enable or disable external signal detector.

Parameters

run – true to enable, false to disable.

inline bool **getDetectorRisingEdges** () const

Report status of rising edge detection on the SIGNAL_IN line.

Returns

true if enabled, false otherwise.

inline void **setDetectorRisingEdges** (const bool detectRising)

Detect rising edges (low to high transitions) on the SIGNAL_IN line.

Parameters

detectRising – true to enable, false to disable.

inline bool **getDetectorFallingEdges** () const

Report status of falling edge detection on the SIGNAL_IN line.

Returns

true if enabled, false otherwise.

inline void **setDetectorFallingEdges** (const bool detectFalling)

Detect falling edges (high to low transitions) on the SIGNAL_IN line.

Parameters

detectFalling – true to enable, false to disable.

inline bool **isGeneratorRunning** () const

Report status of external signal generator.

Returns

true if running, false otherwise.

inline void **setGeneratorRunning** (const bool run)

Enable or disable external signal generator (PWM-like output).

Parameters

run – true to enable, false to disable.

inline `std::chrono::microseconds` `getGeneratorLowTime()` const

Get current PWM low time.

Returns

low time in microseconds.

inline void `setGeneratorLowTime` (const `std::chrono::microseconds` lowTimeUs)

Set PWM low time for external signal generator.

Parameters

`lowTimeUs` – low time in microseconds.

inline `std::chrono::microseconds` `getGeneratorHighTime()` const

Get current PWM high time.

Returns

high time in microseconds.

inline void `setGeneratorHighTime` (const `std::chrono::microseconds` highTimeUs)

Set PWM high time for external signal generator.

Parameters

`highTimeUs` – high time in microseconds.

inline bool `getGeneratorInjectTriggerOnRisingEdge()` const

Report status of trigger event injection feature for external signal generator rising edges.

Returns

true if enabled, false otherwise.

inline void `setGeneratorInjectTriggerOnRisingEdge` (const bool injectRising)

Inject a trigger event of type `EXTERNAL_GENERATOR_RISING_EDGE` into the event stream from the device, every time a rising edge is generated by the PWM-like output of the external signal generator.

Parameters

`injectRising` – true to inject trigger event, false to disable.

inline bool `getGeneratorInjectTriggerOnFallingEdge()` const

Report status of trigger event injection feature for external signal generator falling edges.

Returns

true if enabled, false otherwise.

inline void `setGeneratorInjectTriggerOnFallingEdge` (const bool injectFalling)

Inject a trigger event of type `EXTERNAL_GENERATOR_FALLING_EDGE` into the event stream from the device, every time a falling edge is generated by the PWM-like output of the external signal generator.

Parameters

`injectFalling` – true to inject trigger event, false to disable.

inline `std::chrono::microseconds` `getUSBEarlyPacketDelay()` const

Get value of USB early packet timeout.

Returns

timeout in microseconds.

inline void `setUSBEarlyPacketDelay` (const `std::chrono::microseconds` earlyPacketDelayUs)

Send data over USB early if this timeout is reached, instead of waiting on buffers being full. Timeout on device is in 125µs time-slices.

Parameters

`earlyPacketDelayUs` – timeout in microseconds.

Public Static Functions

static inline auto **findDevices** (const *std::string_view* filterBySerialNumber = {}, loggerCallbackType logger = {})

Find connected *DVXplorer* cameras.

Parameters

- **filterBySerialNumber** – only search for devices with this serial number
- **logger** – discovery logging callback, must be thread-safe

Returns

a descriptor structure describing a compatible device

Protected Functions

inline virtual void **sendTimestampReset** () override

Send a timestamp reset command to the device.

inline virtual void **setTimestampOffset** (const *std::chrono::microseconds* timestampOffset) override

Set a new timestamp offset value for the camera.

Parameters

timestampOffset – New timestamp offset value in microseconds.

inline *std::chrono::microseconds* **getGeneratorHighTimeInterval** () const

inline *std::chrono::microseconds* **getGeneratorLowTimeInterval** () const

inline void **setCropAreaInternal** (const cv::Rect cropArea)

Private Types

enum class **SensorModel**

Values:

enumerator **LITE_QVGA**

enumerator **DVX_VGA**

Private Functions

inline void **shutdownCallback** ()

inline void **usbDataCallback** (const *std::span*<const uint8_t> data)

inline void **dataParserCallback** (*parser::ParsedData* data)

inline void **timeInitCallback** ()

Private Members

uint32_t **mLogicVersion**

```
uint32_t mLogicPatch

cv::Size mResolution

SensorModel mSensorModel

imu::ImuModel mImuModel

float mLogicClockActual

float mUSBClockActual

std::unique_ptr<parser::DVXplorer::Parser> mParser

mutable std::mutex mConfigLock

ReadoutFPS mReadoutFPS

uint8_t mContrastThresholdOn

uint8_t mContrastThresholdOff

cv::Rect mCropArea

std::atomic<bool> mIsRunning = {true}

std::atomic<bool> mTimestampMaster = {true}
```

Private Static Attributes

```
static constexpr uint16_t MODULE_MULTIPLEXER = {0}
```

Module address: device-side Multiplexer configuration. The Multiplexer is responsible for mixing, timestamping and outputting (via USB) the various event types generated by the device. It is also responsible for timestamp generation and synchronization.

```
static constexpr uint16_t MODULE_DVS = {1}
```

Module address: device-side DVS configuration. The DVS state machine interacts with the DVS chip and gets the polarity events from it. It supports various configurable delays, as well as advanced filtering capabilities on the polarity events.

```
static constexpr uint16_t MODULE_IMU = {3}
```

Module address: device-side *IMU* (Inertial Measurement Unit) configuration. The *IMU* module connects to the external *IMU* chip and sends data on the device's movement in space. It can configure various options on the external chip, such as accelerometer range or gyroscope refresh rate.

```
static constexpr uint16_t MODULE_EXTERNAL_INPUT = {4}
```

Module address: device-side External Input (signal detector/generator) configuration. The External Input module is used to detect external signals on the external input jack and inject an event into the event stream when this happens. It can detect pulses of a specific length or rising and falling edges. On some systems, a signal generator module is also present, which can generate PWM-like pulsed signals with configurable timing.

```
static constexpr uint16_t MODULE_SYSINFO = {6}
```

Module address: device-side system information. The system information module provides various details on the device, such as currently installed logic revision or clock speeds. All its parameters are read-only. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct caer_info' documentation for more details on what information is available.

```
static constexpr uint16_t MODULE_USB = {9}
```

Module address: device-side USB output configuration. The USB output module forwards the data from the device and the FPGA/CPLD to the USB chip, usually a Cypress FX2 or FX3.

```
static constexpr uint16_t MUX_RUN = {0}
```

Parameter address for module MUX: run the Multiplexer state machine, which is responsible for mixing the various event types at the device level, timestamping them and outputting them via USB or other connectors.

```
static constexpr uint16_t MUX_TIMESTAMP_RUN = {1}
```

Parameter address for module MUX: run the Timestamp Generator inside the Multiplexer state machine, which will provide microsecond accurate timestamps to the events passing through.

```
static constexpr uint16_t MUX_TIMESTAMP_RESET = {2}
```

Parameter address for module MUX: reset the Timestamp Generator to zero. This also sends a reset pulse to all connected slave devices, resetting their timestamp too.

```
static constexpr uint16_t MUX_RUN_CHIP = {3}
```

Parameter address for module MUX: power up the chip's bias generator, enabling the chip to work.

```
static constexpr uint16_t MUX_DROP_EXTINPUT_ON_TRANSFER_STALL = {4}
```

Parameter address for module MUX: drop External Input events if the USB output FIFO is full, instead of having them pile up at the input FIFOs.

```
static constexpr uint16_t MUX_DROP_DVS_ON_TRANSFER_STALL = {5}
```

Parameter address for module MUX: drop DVS events if the USB output FIFO is full, instead of having them pile up at the input FIFOs.

```
static constexpr uint16_t MUX_STATISTICS_EXTINPUT_DROPPED = {81}
```

Parameter address for module MUX: read-only parameter, representing the number of dropped External Input events on the device due to full USB buffers. This is a 64bit value, and should always be read using the function: caerDeviceConfigGet64().

```
static constexpr uint16_t MUX_STATISTICS_DVS_DROPPED = {83}
```

Parameter address for module MUX: read-only parameter, representing the number of dropped DVS events

on the device due to full USB buffers. This is a 64bit value, and should always be read using the function: `caerDeviceConfigGet64()`.

`static constexpr uint16_t DVS_SIZE_COLUMNS = {0}`

Parameter address for module DVS: read-only parameter, contains the X axis resolution of the DVS events returned by the camera. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct caer_davis_info' documentation to get proper size information that already considers the rotation and orientation settings.

`static constexpr uint16_t DVS_SIZE_ROWS = {1}`

Parameter address for module DVS: read-only parameter, contains the Y axis resolution of the DVS events returned by the camera. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct caer_davis_info' documentation to get proper size information that already considers the rotation and orientation settings.

`static constexpr uint16_t DVS_ORIENTATION_INFO = {2}`

Parameter address for module DVS: read-only parameter, contains information on the orientation of the X/Y axes, whether they should be inverted or not on the host when parsing incoming events. Bit 2: `dvsInvertXY` Bit 1: reserved Bit 0: reserved This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct caer_davis_info' documentation to get proper size information that already considers the rotation and orientation settings.

`static constexpr uint16_t DVS_RUN = {3}`

Parameter address for module DVS: run the DVS state machine and read out polarity events from the chip.

`static constexpr uint16_t DVS_STATISTICS_COLUMN = {81}`

Parameter address for module DVS: read-only parameter, representing the number of column transactions completed successfully on the device. This is a 64bit value, and should always be read using the function: `caerDeviceConfigGet64()`.

`static constexpr uint16_t DVS_STATISTICS_GROUP = {83}`

Parameter address for module DVS: read-only parameter, representing the number of SGroup/MGroup transactions completed successfully on the device. This is a 64bit value, and should always be read using the function: `caerDeviceConfigGet64()`.

`static constexpr uint16_t DVS_STATISTICS_DROPPED_COLUMN = {85}`

Parameter address for module DVS: read-only parameter, representing the number of dropped column transactions on the device. This is a 64bit value, and should always be read using the function: `caerDeviceConfigGet64()`.

`static constexpr uint16_t DVS_STATISTICS_DROPPED_GROUP = {87}`

Parameter address for module DVS: read-only parameter, representing the number of dropped SGroup/MGroup transactions on the device. This is a 64bit value, and should always be read using the function: `caerDeviceConfigGet64()`.

`static constexpr uint16_t IMU_TYPE = {0}`

Parameter address for module *IMU*: read-only parameter, contains information on the type of *IMU* chip being used in this device: 0 - no *IMU* present 3 - Bosch BMI 160 This is reserved for internal use and should not be used by anything other than libcaer.

```
static constexpr uint16_t IMU_ORIENTATION_INFO = {1}
```

Parameter address for module *IMU*: read-only parameter, contains information on the orientation of the X/Y/Z axes, whether they should be flipped or not on the host when parsing incoming *IMU* data samples. Bit 2: imuFlipX Bit 1: imuFlipY Bit 0: imuFlipZ This is reserved for internal use and should not be used by anything other than libcaer. Generated *IMU* events are already properly flipped when returned to the user.

```
static constexpr uint16_t IMU_RUN_ACCELEROMETER = {2}
```

Parameter address for module *IMU*: enable the *IMU*'s accelerometer. This takes the *IMU* chip out of sleep.

```
static constexpr uint16_t IMU_RUN_GYROSCOPE = {3}
```

Parameter address for module *IMU*: enable the *IMU*'s gyroscope. This takes the *IMU* chip out of sleep.

```
static constexpr uint16_t IMU_RUN_TEMPERATURE = {4}
```

Parameter address for module *IMU*: enable the *IMU*'s temperature sensor. This takes the *IMU* chip out of sleep.

```
static constexpr uint16_t IMU_ACCEL_DATA_RATE = {5}
```

Parameter address for module *IMU*: 8 settings: 0 - 12.5 Hz 1 - 25 Hz 2 - 50 Hz 3 - 100 Hz 4 - 200 Hz 5 - 400 Hz 6 - 800 Hz 7 - 1600 Hz

```
static constexpr uint16_t IMU_ACCEL_FILTER = {6}
```

Parameter address for module *IMU*: 3 settings: 0 - OSR4 1 - OSR2 2 - Normal

```
static constexpr uint16_t IMU_ACCEL_RANGE = {7}
```

Parameter address for module *IMU*: 4 settings: 0 - +- 2g 1 - +- 4g 2 - +- 8g 3 - +- 16g

```
static constexpr uint16_t IMU_GYRO_DATA_RATE = {8}
```

Parameter address for module *IMU*: 8 settings: 0 - 25 Hz 1 - 50 Hz 2 - 100 Hz 3 - 200 Hz 4 - 400 Hz 5 - 800 Hz 6 - 1600 Hz 7 - 3200 Hz

```
static constexpr uint16_t IMU_GYRO_FILTER = {9}
```

Parameter address for module *IMU*: 3 settings: 0 - OSR4 1 - OSR2 2 - Normal

```
static constexpr uint16_t IMU_GYRO_RANGE = {10}
```

Parameter address for module *IMU*: 5 settings: 0 - +- 2000°/s 1 - +- 1000°/s 2 - +- 500°/s 3 - +- 250°/s 4 - +- 125°/s

```
static constexpr uint16_t EXTINPUT_RUN_DETECTOR = {0}
```

Parameter address for module EXTINPUT: enable the signal detector module. It generates events when it sees certain types of signals, such as edges or pulses of a defined length, on the SIGNAL pin of the INPUT synchronization connector. This can be useful to inject events into the event stream in response to external stimuli or controls, such as turning on a LED lamp.

```
static constexpr uint16_t EXTINPUT_DETECT_RISING_EDGES = {1}
```

Parameter address for module EXTINPUT: send a special EXTERNAL_INPUT_RISING_EDGE event when a rising edge is detected (transition from low voltage to high).

```
static constexpr uint16_t EXTINPUT_DETECT_FALLING_EDGES = {2}
```

Parameter address for module EXTINPUT: send a special `EXTERNAL_INPUT_FALLING_EDGE` event when a falling edge is detected (transition from high voltage to low).

```
static constexpr uint16_t EXTINPUT_RUN_GENERATOR = {11}
```

Parameter address for module EXTINPUT: enable the signal generator module. It generates a PWM-like signal based on configurable parameters and outputs it on the OUT JACK signal.

```
static constexpr uint16_t EXTINPUT_GENERATE_PULSE_INTERVAL = {13}
```

Parameter address for module EXTINPUT: the interval between the start of two consecutive pulses, expressed in cycles at LogicClock frequency (see ‘struct caer_davis_info’ for details on how to get the frequency). This must be bigger or equal to `EXTINPUT_GENERATE_PULSE_LENGTH`. To generate a signal with 50% duty cycle, this would have to be exactly double of `EXTINPUT_GENERATE_PULSE_LENGTH`.

```
static constexpr uint16_t EXTINPUT_GENERATE_PULSE_LENGTH = {14}
```

Parameter address for module EXTINPUT: the length a pulse stays active, expressed in cycles at LogicClock frequency (see ‘struct caer_davis_info’ for details on how to get the frequency). This must be smaller or equal to `EXTINPUT_GENERATE_PULSE_INTERVAL`. To generate a signal with 50% duty cycle, this would have to be exactly half of `EXTINPUT_GENERATE_PULSE_INTERVAL`.

```
static constexpr uint16_t EXTINPUT_GENERATE_INJECT_ON_RISING_EDGE = {15}
```

Parameter address for module EXTINPUT: enables event injection when a rising edge occurs in the generated signal; a special event `EXTERNAL_GENERATOR_RISING_EDGE` is emitted into the event stream.

```
static constexpr uint16_t EXTINPUT_GENERATE_INJECT_ON_FALLING_EDGE = {16}
```

Parameter address for module EXTINPUT: enables event injection when a falling edge occurs in the generated signal; a special event `EXTERNAL_GENERATOR_FALLING_EDGE` is emitted into the event stream.

```
static constexpr uint16_t SYSINFO_LOGIC_VERSION = {0}
```

Parameter address for module SYSINFO: read-only parameter, the version of the logic currently running on the device’s FPGA/CPLD. This is reserved for internal use and should not be used by anything other than libcaer. Please see the ‘struct caer_info’ documentation to get this information.

```
static constexpr uint16_t SYSINFO_CHIP_IDENTIFIER = {1}
```

Parameter address for module SYSINFO: read-only parameter, an integer used to identify the different types of sensor chips used on the device. This is reserved for internal use and should not be used by anything other than libcaer. Please see the ‘struct caer_info’ documentation to get this information.

```
static constexpr uint16_t SYSINFO_DEVICE_IS_MASTER = {2}
```

Parameter address for module SYSINFO: read-only parameter, whether the device is currently a timestamp master or slave when synchronizing multiple devices together. This is reserved for internal use and should not be used by anything other than libcaer. Please see the ‘struct caer_info’ documentation to get this information.

```
static constexpr uint16_t SYSINFO_LOGIC_CLOCK = {3}
```

Parameter address for module SYSINFO: read-only parameter, the frequency in MHz at which the main FPGA/CPLD logic is running. This is reserved for internal use and should not be used by anything other than libcaer.

```
static constexpr uint16_t SYSINFO_USB_CLOCK = {5}
```

Parameter address for module SYSINFO: read-only parameter, the frequency in MHz at which the FPGA/CPLD logic related to USB data transmission is running. This is reserved for internal use and should not be used by anything other than libcaer.

```
static constexpr uint16_t SYSINFO_CLOCK_DEVIATION = {6}
```

Parameter address for module SYSINFO: read-only parameter, the deviation factor for the clocks. Due to how FX3 generates the clocks, which are then used by FPGA/CPLD, they are not integers but have a fractional part. This is reserved for internal use and should not be used by anything other than libcaer.

```
static constexpr uint16_t SYSINFO_LOGIC_PATCH = {7}
```

Parameter address for module SYSINFO: read-only parameter, the patch version of the logic currently running on the device's FPGA/CPLD. This is reserved for internal use and should not be used by anything other than libcaer.

```
static constexpr uint16_t USB_RUN = {0}
```

Parameter address for module USB: enable the USB FIFO module, which transfers the data from the FPGA/CPLD to the USB chip, to be then sent to the host. Turning this off will suppress any USB data communication!

```
static constexpr uint16_t USB_EARLY_PACKET_DELAY = {1}
```

Parameter address for module USB: the time delay after which a packet of data is committed to USB, even if it is not full yet (short USB packet). The value is in 125µs time-slices, corresponding to how USB schedules its operations (a value of 4 for example would mean waiting at most 0.5ms until sending a short USB packet to the host).

```
static constexpr uint16_t MODULE_DEVICE = {5}
```

Module address: device-side chip configuration. This state machine is responsible for configuring the Samsung DVS chip.

```
static constexpr uint16_t REGISTER_BIAS_CURRENT_RANGE_SELECT_LOGSFONREST = {0x000B}
```

```
static constexpr uint16_t REGISTER_BIAS_CURRENT_RANGE_SELECT_LOGALOGD_MONITOR = {0x000C}
```

```
static constexpr uint16_t REGISTER_BIAS_OTP_TRIM = {0x000D}
```

```
static constexpr uint16_t REGISTER_BIAS_PINS_DBGP = {0x000F}
```

```
static constexpr uint16_t REGISTER_BIAS_PINS_DBGN = {0x0010}
```

```
static constexpr uint16_t REGISTER_BIAS_CURRENT_LEVEL_SFOFF = {0x0012}
```

```
static constexpr uint16_t REGISTER_BIAS_PINS_BUFN = {0x0013}
```

```
static constexpr uint16_t REGISTER_BIAS_PINS_BUFN = {0x0014}
```

```
static constexpr uint16_t REGISTER_BIAS_PINS_DOB = {0x0015}

static constexpr uint16_t REGISTER_BIAS_CURRENT_AMP = {0x0018}

static constexpr uint16_t REGISTER_BIAS_CURRENT_ON = {0x001C}

static constexpr uint16_t REGISTER_BIAS_CURRENT_OFF = {0x001E}

static constexpr uint16_t REGISTER_CONTROL_MODE = {0x3000}

static constexpr uint16_t REGISTER_CONTROL_INTERRUPT_SOURCE = {0x3004}

static constexpr uint16_t REGISTER_CONTROL_INTERRUPT_ENABLE_TIME = {0x3005}

static constexpr uint16_t REGISTER_CONTROL_INTERRUPT_ACKNOWLEDGE = {0x3007}

static constexpr uint16_t REGISTER_CONTROL_INTERRUPT_AUTO_MODE = {0x3008}

static constexpr uint16_t REGISTER_CONTROL_INTERRUPT_RELEASE_TIME = {0x3009}

static constexpr uint16_t REGISTER_CONTROL_PLL_P = {0x300D}

static constexpr uint16_t REGISTER_CONTROL_PLL_M = {0x300E}

static constexpr uint16_t REGISTER_CONTROL_PLL_S = {0x3010}

static constexpr uint16_t REGISTER_CONTROL_CLOCK_DIVIDER_SYS = {0x3011}

static constexpr uint16_t REGISTER_CONTROL_CLOCK_DIVIDER_PVI = {0x3012}

static constexpr uint16_t REGISTER_CONTROL_PARALLEL_OUT_CONTROL = {0x3019}

static constexpr uint16_t REGISTER_CONTROL_PARALLEL_OUT_ENABLE = {0x301E}

static constexpr uint16_t REGISTER_CONTROL_PACKET_FORMAT = {0x3067}

static constexpr uint16_t REGISTER_DIGITAL_ENABLE = {0x3200}

static constexpr uint16_t REGISTER_DIGITAL_RESTART = {0x3201}

static constexpr uint16_t REGISTER_DIGITAL_DUAL_BINNING = {0x3202}
```

```
static constexpr uint16_t REGISTER_DIGITAL_SUBSAMPLE_RATIO = {0x3204}

static constexpr uint16_t REGISTER_DIGITAL_AREA_BLOCK = {0x3205}

static constexpr uint16_t REGISTER_DIGITAL_TIMESTAMP_SUBUNIT = {0x3234}

static constexpr uint16_t REGISTER_DIGITAL_TIMESTAMP_REFUNIT = {0x3235}

static constexpr uint16_t REGISTER_DIGITAL_DTAG_REFERENCE = {0x323D}

static constexpr uint16_t REGISTER_DIGITAL_TIMESTAMP_RESET = {0x3238}

static constexpr uint16_t REGISTER_TIMING_FIRST_SELX_START = {0x323C}

static constexpr uint16_t REGISTER_TIMING_GH_COUNT = {0x3240}

static constexpr uint16_t REGISTER_TIMING_GH_COUNT_FINE = {0x3243}

static constexpr uint16_t REGISTER_TIMING_GRS_COUNT = {0x3244}

static constexpr uint16_t REGISTER_TIMING_GRS_COUNT_FINE = {0x3247}

static constexpr uint16_t REGISTER_DIGITAL_GLOBAL_RESET_READOUT = {0x3248}

static constexpr uint16_t REGISTER_TIMING_NEXT_GH_CNT = {0x324B}

static constexpr uint16_t REGISTER_TIMING_SELX_WIDTH = {0x324C}

static constexpr uint16_t REGISTER_TIMING_AY_START = {0x324E}

static constexpr uint16_t REGISTER_TIMING_AY_END = {0x324F}

static constexpr uint16_t REGISTER_TIMING_MAX_EVENT_NUM = {0x3251}

static constexpr uint16_t REGISTER_TIMING_R_START = {0x3253}

static constexpr uint16_t REGISTER_TIMING_R_END = {0x3254}

static constexpr uint16_t REGISTER_DIGITAL_MODE_CONTROL = {0x3255}

static constexpr uint16_t REGISTER_TIMING_GRS_END = {0x3256}
```

```
static constexpr uint16_t REGISTER_TIMING_GRS_END_FINE = {0x3259}

static constexpr uint16_t REGISTER_DIGITAL_FIXED_READ_TIME = {0x325C}

static constexpr uint16_t REGISTER_TIMING_READ_TIME_INTERVAL = {0x325D}

static constexpr uint16_t REGISTER_DIGITAL_EXTERNAL_TRIGGER = {0x3260}

static constexpr uint16_t REGISTER_TIMING_NEXT_SELX_START = {0x3261}

static constexpr uint16_t REGISTER_DIGITAL_BOOT_SEQUENCE = {0x3266}

static constexpr uint16_t REGISTER_CROPPER_BYPASS = {0x3300}

static constexpr uint16_t REGISTER_CROPPER_Y_START_GROUP = {0x3301}

static constexpr uint16_t REGISTER_CROPPER_Y_START_MASK = {0x3302}

static constexpr uint16_t REGISTER_CROPPER_Y_END_GROUP = {0x3303}

static constexpr uint16_t REGISTER_CROPPER_Y_END_MASK = {0x3304}

static constexpr uint16_t REGISTER_CROPPER_X_START_ADDRESS = {0x3305}

static constexpr uint16_t REGISTER_CROPPER_X_END_ADDRESS = {0x3307}

static constexpr uint16_t REGISTER_ACTIVITY_DECISION_BYPASS = {0x3500}

static constexpr uint16_t REGISTER_SPATIAL_HISTOGRAM_OFF = {0x3600}

static constexpr uint16_t DVS_CHIP_MODE_OFF = {0}

static constexpr uint16_t DVS_CHIP_MODE_MONITOR = {1}

static constexpr uint16_t DVS_CHIP_MODE_STREAM = {2}

static constexpr uint16_t DVS_CHIP_DTAG_CONTROL_STOP = {0}

static constexpr uint16_t DVS_CHIP_DTAG_CONTROL_START = {1}

static constexpr uint16_t DVS_CHIP_DTAG_CONTROL_RESTART = {2}
```

```

static constexpr uint16_t SYSTEM_CLOCK_FREQUENCY = {50}

static constexpr uint16_t PID_DVXPLOER = {0x8419}

static constexpr uint8_t FIRMWARE_REQUIRED_VERSION = {9}

static constexpr uint32_t LOGIC_REQUIRED_VERSION = {18}

static constexpr uint32_t LOGIC_MINIMUM_PATCH = {4}

static constexpr uint32_t USB_EARLY_PACKET_DELAY_MAX = {(1 << 20) - 1}

static constexpr uint32_t EXT_INPUT_TIME_MAX = {(1 << 20) - 1}

static constexpr auto COMPATIBLE_CAMERA = [] (const uint16_t vid, const uint16_t pid, const
USBDeviceType deviceType) -> std::optional<CameraModel> {if ((vid ==
VID_INIVATION) && (pid == PID_DVXPLOER) && (deviceType ==
USBDeviceType::FX3_RED)) {return CameraModel::DVXPLOER;}return std::nullopt;}

```

Friends

```

inline friend std::ostream &operator<< (std::ostream &os, const SubSample &var)

inline friend std::ostream &operator<< (std::ostream &os, const ReadoutFPS &var)

```

```
class DVXplorerM : public dv::io::camera::USBDeviceNextGen, public dv::io::camera::CameraInputBase
```

Public Functions

```

inline explicit DVXplorerM (loggerCallbackType deviceLogger = {}, const LogLevel deviceLogLevel =
LogLevel::LVL_WARNING)

```

Open the first *DVXplorer* Mini/Micro camera that can be found. Throws if device cannot be opened.

Parameters

- **deviceLogger** – per-device logging callback, must be thread-safe
- **deviceLogLevel** – initial log-level

```

inline explicit DVXplorerM (const std::string_view filterBySerialNumber, loggerCallbackType deviceLogger =
{}, const LogLevel deviceLogLevel = LogLevel::LVL_WARNING)

```

Open the *DVXplorer* Mini/Micro camera with the specified serial number. Throws if device cannot be opened.

Parameters

- **filterBySerialNumber** – serial number to search for
- **deviceLogger** – per-device logging callback, must be thread-safe
- **deviceLogLevel** – initial log-level

```
inline explicit DVXplorerM (const DeviceDescriptor &deviceToOpen, loggerCallbackType deviceLogger = {},  
                           const LogLevel deviceLogLevel = LogLevel::LVL_WARNING)
```

Open the *DVXplorer* Mini/Micro camera corresponding to the specified descriptor. Throws if device cannot be opened.

Parameters

- **deviceToOpen** – device descriptor structure
- **deviceLogger** – per-device logging callback, must be thread-safe
- **deviceLogLevel** – initial log-level

```
inline ~DVXplorerM () override
```

```
inline void configLoadFromFlash ()
```

Load configuration from on-device flash memory.

```
inline void configStoreToFlash ()
```

Store current configuration to on-device flash memory.

```
inline virtual std::string getCameraName () const override
```

Get camera name, which is a combination of the camera model and the serial number.

Returns

String containing the camera model and serial number separated by an underscore character.

```
inline virtual std::optional<cv::Size> getEventResolution () const override
```

Get event stream resolution.

Returns

Event stream resolution, `std::nullopt` if event stream is unavailable.

```
inline virtual std::optional<cv::Size> getFrameResolution () const override
```

Retrieve frame stream resolution.

Returns

Frame stream resolution or `std::nullopt` if the frame stream is not available.

```
inline virtual imu::ImuModel getImuModel () const override
```

Return *IMU* model used on device.

Returns

IMU model in use.

```
inline virtual float getPixelPitch () const override
```

Return pixel pitch distance for the connected camera model. The value is returned in meters.

Returns

Pixel pitch distance in meters according to the connected device.

```
inline virtual bool isEventStreamAvailable () const override
```

Check whether event stream is available.

Returns

True if event stream is available, false otherwise.

```
inline virtual bool isFrameStreamAvailable () const override
```

Check whether frame stream is available.

Returns

True if frame stream is available, false otherwise.

inline virtual bool **isImuStreamAvailable** () const override

Check whether *IMU* data is available.

Returns

True if *IMU* data stream is available, false otherwise.

inline virtual bool **isTriggerStreamAvailable** () const override

Check whether trigger data is available.

Returns

True if trigger data stream is available, false otherwise.

inline virtual bool **isRunning** () const override

Check whether any input data streams have terminated. For a live camera this should check if the device is still connected and functioning, while for a recording file this should check if any of the data streams have reached end-of-file (EOF). For a network input, this indicates the network stream is still connected.

Returns

True if data read on all streams is still possible, false otherwise.

inline virtual bool **isRunning** (const *std::string_view* streamName) const override

Check whether the input data stream with the specified name is still active.

Returns

True if data read on this stream is possible, false otherwise.

inline virtual bool **isRunningAny** () const override

Check whether any input data streams are still available. For a live camera this should check if the device is still connected and functioning and at least one data stream is active (different than *isRunning()*), while for a recording file this should check if any of the data streams have not yet reached end-of-file (EOF) and are still readable. For a network input, this indicates the network stream is still connected.

Returns

True if data read on at least one stream is still possible, false otherwise.

inline virtual *std::chrono::microseconds* **getTimeInterval** () const override

Get the time interval for data commit.

Returns

Time interval in microseconds.

inline virtual void **setTimeInterval** (const *std::chrono::microseconds* timeInterval) override

Set a new time interval value for data commit. Data is put in the queues for *getNextEventBatch()*, *readNext()*, ... at this interval's rate.

Parameters

timeInterval – New time interval value in microseconds.

inline virtual *std::chrono::microseconds* **getTimestampOffset** () const override

Get the timestamp offset.

Returns

Absolute timestamp offset value in microseconds.

inline virtual bool **getFlipHorizontal** () const override

Status of horizontal events flip.

Returns

status of horizontal events flip.

inline virtual void **setFlipHorizontal** (const bool flipHorizontal) override

Flip events horizontally.

Parameters

flipHorizontalEvents – flip events horizontally.

inline virtual bool **getFlipVertical** () const override

Status of vertical events flip.

Returns

status of vertical events flip.

inline virtual void **setFlipVertical** (const bool flipVertical) override

Flip events vertically.

Parameters

flipVerticalEvents – flip events vertically.

inline virtual Flatten **getFlatten** () const override

Status of event polarity flattening.

Returns

status of event polarity flattening.

inline virtual void **setFlatten** (const Flatten flatten) override

Flatten events polarity.

Parameters

flattenEvents – flattening mode.

inline *DVXplorer::SubSample* **getSubSampleHorizontal** () const

Get horizontal subsampling mode. VGA sensor: all values. HD sensor: only EVERY_PIXEL, EVERY_SECOND, EVERY_FOURTH, EVERY_EIGHTH.

Returns

current horizontal subsampling mode.

inline void **setSubSampleHorizontal** (const *DVXplorer::SubSample* subSampleHorizontal)

Set horizontal subsampling mode. VGA sensor: all values. HD sensor: only EVERY_PIXEL, EVERY_SECOND, EVERY_FOURTH, EVERY_EIGHTH.

Parameters

subSampleHorizontal – new horizontal subsampling mode.

inline *DVXplorer::SubSample* **getSubSampleVertical** () const

Get vertical subsampling mode. VGA sensor: all values. HD sensor: only EVERY_PIXEL, EVERY_SECOND, EVERY_FOURTH, EVERY_EIGHTH.

Returns

current vertical subsampling mode.

inline void **setSubSampleVertical** (const *DVXplorer::SubSample* subSampleVertical)

Set vertical subsampling mode. VGA sensor: all values. HD sensor: only EVERY_PIXEL, EVERY_SECOND, EVERY_FOURTH, EVERY_EIGHTH.

Parameters

subSampleVertical – new vertical subsampling mode.

inline bool **getGlobalHold** () const

Report status of global hold feature.

Returns

true if enabled, false otherwise.

inline void **setGlobalHold** (const bool globalHold)

Enable or disable global hold feature. Default is enabled. For some applications like LED tracking, setting this to false can help.

Parameters

globalHold – true to enable, false to disable.

inline bool **getGlobalReset** () const

Report status of global reset feature.

Returns

true if enabled, false otherwise.

inline void **setGlobalReset** (const bool globalReset)

Enable or disable global reset feature.

Parameters

globalReset – true to enable, false to disable.

inline uint8_t **getGlobalResetSkip** () const

Get global reset skip value.

Returns

only perform global reset every N+1 readout frames.

inline void **setGlobalResetSkip** (const uint8_t globalResetSkip)

Control global reset skip feature. Only perform global reset, if enabled, every N+1 readout frames. Setting to zero disables this feature.

Parameters

globalResetSkip – only perform global reset every N+1 readout frames.

inline bool **getMIPITimeoutEnable** () const

Report status of MIPI timeout feature.

Returns

true if enabled, false otherwise.

inline void **setMIPITimeoutEnable** (const bool mipiTimeoutEnable)

Enable MIPI timeout, will ensure data is sent from sensor to host at least every N microseconds, even during periods of low activity. See [setMIPITimeoutValue\(\)](#) to set the timeout value.

Parameters

mipiTimeoutEnable – true to enable, false to disable.

inline [std::chrono::microseconds](#) **getMIPITimeoutValue** () const

Get MIPI timeout in microseconds.

Returns

timeout in microseconds.

inline void **setMIPITimeoutValue** (const [std::chrono::microseconds](#) mipiTimeoutValueUs)

Set MIPI timeout in microseconds. See [setMIPITimeoutEnable\(\)](#) to enable this functionality.

Parameters

mipiTimeoutValueUs – timeout in microseconds.

inline *DVXplorer::ReadoutFPS* **getReadoutFPS** () const

Get currently set event-frame readout frequency. Not available on HD sensor.

Returns

event-frame readout frequency.

inline void **setReadoutFPS** (const *DVXplorer::ReadoutFPS* readoutFps)

Set frequency of event-frame readouts on sensor. Not available on HD sensor. CONSTANT frequencies are guaranteed fixed frequencies with no data loss. CONSTANT_LOSSY are guaranteed fixed frequencies but will cut off a readout to respect the timing if too much data is present, resulting in data loss. VARIABLE are best-effort frequencies that will change if lots of data is present, slowing down, but without data loss.

Parameters

readoutFps – event-frame readout frequency.

inline uint8_t **getContrastThresholdOn** () const

Get the contrast threshold for ON polarity event generation. VGA sensor: values from 0 to 17. HD sensor: values from 0 to 127.

Returns

ON contrast threshold.

inline void **setContrastThresholdOn** (const uint8_t contrastThresholdOn)

Set the contrast threshold for ON polarity event generation. VGA sensor: values from 0 to 17. HD sensor: values from 0 to 127.

Parameters

contrastThresholdOn – ON contrast threshold.

inline uint8_t **getContrastThresholdOff** () const

Get the contrast threshold for OFF polarity event generation. VGA sensor: values from 0 to 17. HD sensor: values from 0 to 127.

Returns

OFF contrast threshold.

inline void **setContrastThresholdOff** (const uint8_t contrastThresholdOff)

Set the contrast threshold for OFF polarity event generation. VGA sensor: values from 0 to 17. HD sensor: values from 0 to 127.

Parameters

contrastThresholdOff – OFF contrast threshold.

inline virtual cv::Rect **getCropArea** () const override

Get events Region of Interest (ROI).

Returns

get events Region of Interest (ROI).

inline virtual void **setCropArea** (const cv::Rect cropArea) override

Set events Region of Interest (ROI). Usually hardware accelerated.

Parameters

cropAreaEvents – region of interest (ROI) position and size.

inline bool **getAreaBlock** (const uint32_t blockX, const uint32_t blockY) const

Report if the pixel block at the given X,Y address is filtered out or not.

Parameters

- **blockX** – horizontal address of pixel block. VGA sensor: from 0 to 19, pixel block size 32x32. HD sensor: from 0 to 23, pixel block size 40x40.
- **blockY** – vertical address of pixel block. VGA sensor: from 0 to 14, pixel block size 32x32. HD sensor: from 0 to 17, pixel block size 40x40.

Returns

true if filtered out, false if normal operation.

inline void **setAreaBlock** (const uint32_t blockX, const uint32_t blockY, const bool block)

Enable pixel block filtering at the given X,Y address.

Parameters

- **blockX** – horizontal address of pixel block. VGA sensor: from 0 to 19, pixel block size 32x32. HD sensor: from 0 to 23, pixel block size 40x40.
- **blockY** – vertical address of pixel block. VGA sensor: from 0 to 14, pixel block size 32x32. HD sensor: from 0 to 17, pixel block size 40x40.
- **block** – true to filter out, false for normal operation.

inline bool **isImuRunningAccelerometer** () const

Status of *IMU* accelerometer.

Returns

true if enabled, false otherwise.

inline void **setImuRunningAccelerometer** (const bool run)

Enable or disable *IMU* accelerometer.

Parameters

run – true to enable, false to disable.

inline bool **isImuRunningGyroscope** () const

Status of *IMU* gyroscope.

Returns

true if enabled, false otherwise.

inline void **setImuRunningGyroscope** (const bool run)

Enable or disable *IMU* gyroscope.

Parameters

run – true to enable, false to disable.

inline bool **isImuRunningTemperature** () const

Status of *IMU* temperature measurement.

Returns

true if enabled, false otherwise.

inline void **setImuRunningTemperature** (const bool run)

Enable or disable *IMU* temperature measurement.

Parameters

run – true to enable, false to disable.

inline *imu::BoschBMI160AccelDataRate* **getImuAccelDataRate** () const

Get current *IMU* accelerometer data rate.

Returns

accelerometer data rate.

inline void **setImuAccelDataRate** (const *imu::BoschBMI160AccelDataRate* dataRate)

Set *IMU* accelerometer data rate.

Parameters

dataRate – accelerometer data rate.

inline *imu::BoschBMI160AccelFilter* **getImuAccelFilter** () const

Get current *IMU* accelerometer filter setting.

Returns

accelerometer filter setting.

inline void **setImuAccelFilter** (const *imu::BoschBMI160AccelFilter* filter)

Set *IMU* accelerometer filter setting.

Parameters

filter – accelerometer filter setting.

inline *imu::BoschBMI160AccelRange* **getImuAccelRange** () const

Get current *IMU* accelerometer range.

Returns

accelerometer range.

inline void **setImuAccelRange** (const *imu::BoschBMI160AccelRange* range)

Set *IMU* accelerometer range.

Parameters

range – accelerometer range.

inline *imu::BoschBMI160GyroDataRate* **getImuGyroDataRate** () const

Get current *IMU* gyroscope data rate.

Returns

gyroscope data rate.

inline void **setImuGyroDataRate** (const *imu::BoschBMI160GyroDataRate* dataRate)

Set *IMU* gyroscope data rate.

Parameters

dataRate – gyroscope data rate.

inline *imu::BoschBMI160GyroFilter* **getImuGyroFilter** () const

Get current *IMU* gyroscope filter setting.

Returns

gyroscope filter setting.

inline void **setImuGyroFilter** (const *imu::BoschBMI160GyroFilter* filter)

Set *IMU* gyroscope filter setting.

Parameters

filter – gyroscope filter setting.

```
inline imu::BoschBMI160GyroRange getImuGyroRange () const
```

Get current *IMU* gyroscope range.

Returns

gyroscope range.

```
inline void setImuGyroRange (const imu::BoschBMI160GyroRange range)
```

Set *IMU* gyroscope range.

Parameters

range – gyroscope range.

Public Static Functions

```
static inline auto findDevices (const std::string_view filterBySerialNumber = {}, loggerCallbackType logger = {})
```

Find connected *DVXplorer* Mini/Micro cameras.

Parameters

- **filterBySerialNumber** – only search for devices with this serial number
- **logger** – discovery logging callback, must be thread-safe

Returns

a descriptor structure describing a compatible device

Private Types

```
enum class SensorModel
```

Values:

```
enumerator S5K231Y
```

```
enumerator S5KRC1S
```

Private Functions

```
inline void shutdownCallback ()
```

```
inline void usbDataCallback (const std::span<const uint8_t> data)
```

```
inline void usbDebugCallback (const std::span<const uint8_t> data) const
```

```
inline void dataParserCallback (parser::ParsedData data)
```

```
inline void timeInitCallback ()
```

```
inline char getDVSOrientationX () const
```

```
inline char getDVSOrientationY () const
```

Private Members

cv::Size **mResolution**

SensorModel **mSensorModel**

imu::ImuModel **mImuModel**

std::unique_ptr<parser::ParserBase> **mParser**

mutable *std::mutex* **mConfigLock**

std::atomic<bool> **mIsRunning** = {true}

Private Static Attributes

static constexpr uint8_t **MODULE_DVS** = {1}

static constexpr uint8_t **DVS_RESOLUTION_X** = {0}

static constexpr uint8_t **DVS_RESOLUTION_Y** = {1}

static constexpr uint8_t **DVS_ORIENTATION** = {2}

static constexpr uint8_t **DVS_RUN** = {3}

static constexpr uint8_t **DVS_FLATTEN** = {4}

static constexpr uint8_t **DVS_SUBSAMPLE_HORIZONTAL** = {5}

static constexpr uint8_t **DVS_SUBSAMPLE_VERTICAL** = {6}

static constexpr uint8_t **DVS_GLOBAL_HOLD** = {8}

static constexpr uint8_t **DVS_GLOBAL_RESET** = {9}

static constexpr uint8_t **DVS_GLOBAL_RESET_SKIP** = {10}

static constexpr uint8_t **DVS_MIPI_TIMEOUT_ENABLE** = {11}

static constexpr uint8_t **DVS_MIPI_TIMEOUT_VALUE** = {12}

static constexpr uint8_t **DVS_EFPS_S5K231Y** = {13}

```
static constexpr uint8_t DVS_CONTRAST_THRESHOLD_ON = {14}

static constexpr uint8_t DVS_CONTRAST_THRESHOLD_OFF = {15}

static constexpr uint8_t DVS_AREA_BLOCKING_X_BLOCK = {16}

static constexpr uint8_t DVS_AREA_BLOCKING_Y_BLOCK = {17}

static constexpr uint8_t DVS_AREA_BLOCKING_BLOCK = {18}

static constexpr uint8_t DVS_CROP_X = {19}

static constexpr uint8_t DVS_CROP_Y = {20}

static constexpr uint8_t DVS_CROP_WIDTH = {21}

static constexpr uint8_t DVS_CROP_HEIGHT = {22}

static constexpr uint8_t DVS_CROP_APPLY = {23}

static constexpr uint8_t DVS_FLIP_HORIZONTAL = {24}

static constexpr uint8_t DVS_FLIP_VERTICAL = {25}

static constexpr uint8_t MODULE_IMU = {3}

static constexpr uint8_t IMU_MODEL = {0}

static constexpr uint8_t IMU_ORIENTATION = {1}

static constexpr uint8_t IMU_RUN_ACCELEROMETER = {2}

static constexpr uint8_t IMU_RUN_GYROSCOPE = {3}

static constexpr uint8_t IMU_RUN_TEMPERATURE = {4}

static constexpr uint8_t IMU_ACCEL_DATA_RATE = {5}

static constexpr uint8_t IMU_ACCEL_FILTER = {6}

static constexpr uint8_t IMU_ACCEL_RANGE = {7}
```

```
static constexpr uint8_t IMU_GYRO_DATA_RATE = {8}

static constexpr uint8_t IMU_GYRO_FILTER = {9}

static constexpr uint8_t IMU_GYRO_RANGE = {10}

static constexpr uint8_t IMU_FOC_RUN = {11}

static constexpr uint8_t IMU_FOC_ACCEL_X = {12}

static constexpr uint8_t IMU_FOC_ACCEL_Y = {13}

static constexpr uint8_t IMU_FOC_ACCEL_Z = {14}

static constexpr uint8_t IMU_FOC_GYRO_X = {15}

static constexpr uint8_t IMU_FOC_GYRO_Y = {16}

static constexpr uint8_t IMU_FOC_GYRO_Z = {17}

static constexpr uint8_t IMU_FOC_APPLY = {18}

static constexpr uint8_t VR_CONFIG_STORE = {0xC7}

static constexpr uint8_t VR_CONFIG_RELOAD = {0xC8}

static constexpr uint16_t PID_DVXPLOER = {0x8419}

static constexpr uint8_t FIRMWARE_REQUIRED_VERSION = {10}

static constexpr auto COMPATIBLE_CAMERA = [] (const uint16_t vid, const uint16_t pid, const
USBDeviceType deviceType) -> std::optional<CameraModel> {if ((vid ==
VID_INIVATION) && (pid == PID_DVXPLOER) && (deviceType ==
USBDeviceType::CX3_MIPI)) {return CameraModel::DVXPLOER_M;}return std::nullopt;}
```

struct **eColumn**

Public Functions

inline explicit **eColumn** (const uint32_t event)

Public Members

`int16_t columnAddress`

`int16_t timestampSubUnit`

`bool startOfFrame`

struct **eColumn**

Public Functions

inline explicit **eColumn** (const uint32_t event)

Public Members

`int16_t columnAddress`

`uint8_t frameNumber`

`bool startOfFrame`

`bool mirrorMode`

struct **eDataLost**

class **EdgeMapAccumulator** : public *dv::AccumulatorBase*

#include </builds/inivation/dv/dv-processing/include/dv-processing/core/frame/edge_map_accumulator.hpp>
dv::EdgeMapAccumulator accumulates events in a histogram representation with configurable contribution, but it is more efficient compared to generic accumulator since it uses 8-bit unsigned integers as internal memory type.

The *EdgeMapAccumulator* behaves the same as a generic *dv::Accumulator* with STEP decay function, neutral and minimum value of 0.0, maximum value of 1.0 and configurable event contribution. The difference is that it doesn't use floating point numbers for the potential surface representation. The output data type of this accumulator is single channel 8-bit unsigned integer (CV_8UC1). Accumulation is performed using integer operations as well. Due to performance, no check on the event coordinates inside image plane is performed, unless compiled specifically in DEBUG mode. Events out of the image plane bounds will result in undefined behaviour, or program termination in DEBUG mode.

Public Functions

inline explicit **EdgeMapAccumulator** (const cv::Size &resolution, const float contribution_ = 0.25f, const bool ignorePolarity_ = true, const float neutralPotential = 0.f, const float decay_ = *EdgeMapAccumulator::DECAY_FULL*)

Create a pixel accumulator with known image dimensions and event contribution.

Parameters

- **resolution** – Dimensions of the expected event sensor

- **contribution_** – Contribution coefficient for a single event. The contribution value is multiplied by the maximum possible pixel value (255) to get the increment value. E.g. contribution value of 0.1 will increment a pixel value at a single event coordinates by 26.
- **ignorePolarity_** – Set ignore polarity option. All events are considered positive if enabled.
- **neutralPotential** – Neutral potential value. Neutral value is the default pixel value when decay is disabled and the value that pixels decay into when decay is enabled. The range for neutral potential value is [0.0; 1.0], where 1.0 stands for maximum possible potential - 255 in 8-bit pixel representation.
- **decay_** – Decay coefficient value. This value defines how fast pixel values decay to neutral value. The bigger the value the faster the pixel value will reach neutral value. Decay is applied before each frame generation. The range for decay value is [0.0; 1.0], where 0.0 will not apply any decay and 1.0 will apply maximum decay value resetting a pixel to neutral potential at each generation (default behavior).

inline float **getEventContribution** () const

Get the contribution coefficient for a single event. The contribution value is multiplied by the maximum possible pixel value (255) to get the increment value. E.g. contribution value of 0.1 will increment a pixel value at a single event coordinates by 26.

Returns

Contribution coefficient

inline void **setEventContribution** (const float contribution_)

Set new contribution coefficient.

Parameters

contribution_ – Contribution coefficient for a single event. The contribution value is multiplied by the maximum possible pixel value (255) to get the increment value. E.g. contribution value of 0.1 will increment a pixel value at a single event coordinates by 26.

inline virtual void **accumulate** (const *EventStore* &packet) override

Perform accumulation on given events.

Parameters

packet – Event store containing event to be accumulated.

inline virtual *dv::Frame* **generateFrame** () override

Generates the accumulation frame (potential surface) at the time of the last consumed event. The function writes the output image into the given `outFrame` argument. The output frame will contain data with type CV_8UC1.

The function resets any events accumulated up to this function call.

Returns

accumulated frame

inline void **reset** ()

Clear the buffered events.

inline *EdgeMapAccumulator* &**operator<<** (const *EventStore* &store)

Accumulates the event store into the accumulator.

Parameters

store – The event store to be accumulated.

Returns

A reference to this *EdgeMapAccumulator*.

inline bool **isIgnorePolarity** () const

Check whether ignore polarity option is set to true.

Returns

True if the accumulator assumes all events as positive, false otherwise.

inline void **setIgnorePolarity** (const bool ignorePolarity_)

Set ignore polarity option. All events are considered positive if enabled.

Parameters

ignorePolarity_ – True to enable ignore polarity option.

inline float **getNeutralPotential** () const

Get the neutral potential value for the accumulator. The range for potential value is [0.0; 1.0], where 1.0 stands for maximum possible potential - 255 in 8-bit pixel representation.

Returns

Neutral potential value in range [0.0; 1.0]

inline void **setNeutralPotential** (const float neutralPotential)

Set the neutral potential value. The value should be in range 0.0 to 1.0, other values will be clamped to this range.

Parameters

neutralPotential – Neutral potential value in range [0.0; 1.0].

inline float **getDecay** () const

Get current decay value.

Returns

Decay value.

inline void **setDecay** (const float decay_)

Set the decay value. Decay value is clamped to range of [0.0; 1.0].

Parameters

decay_ – Decay value. Negative value disabled the decay.

Public Static Attributes

static constexpr float **DECAY_NONE** = 0.0f

Decay coefficient value to disable any decay - zero decay.

static constexpr float **DECAY_FULL** = 1.0f

Maximum decay coefficient value which causes reset of pixels into neutral potential at each frame generation.

Protected Types

enum class **DecayMode**

Values:

enumerator **NONE**

enumerator **FULL**

enumerator **DECAY**

Protected Attributes

dv::EventStore **buffer**

Buffer to keep the latest events

uint8_t **maxByteValue** = 255

Max unsigned byte value

float **contribution** = 0.25f

Default contribution

uint8_t **drawIncrement** = (static_cast<uint8_t>(static_cast<float>(maxByteValue) * contribution))

Increment value for a single event

std::vector<uint8_t> **incrementLUT**

A look-up table for increment values at each possible pixel value.

bool **ignorePolarity** = true

float **neutralValue** = 0.f

uint8_t **neutralByteValue** = 0

float **decay** = 1.0

std::vector<uint8_t> **decayLUT**

cv::Mat **imageBuffer**

DecayMode **decayMode** = *DecayMode::FULL*

Friends

inline friend *std::ostream* &**operator<<** (*std::ostream* &os, const *DecayMode* &var)

struct **eFrameEnd**

Public Functions

inline explicit **eFrameEnd** (const uint32_t event)

Public Members

uint8_t **frameNumber**

struct **EigenEvents**

#include </builds/inivation/dv/dv-processing/include/dv-processing/core/core.hpp> A structure that contains events represented in eigen matrices. Useful for mathematical operations using the Eigen library..

Public Functions

inline explicit **EigenEvents** (const size_t size)

Public Members

Eigen::Matrix<int64_t, Eigen::Dynamic, 1> **timestamps**

Eigen::Matrix<int16_t, Eigen::Dynamic, 2> **coordinates**

Eigen::Matrix<uint8_t, Eigen::Dynamic, 1> **polarities**

struct **EmptyException**

Subclassed by *dv::exceptions::info::BadAlloc*, *dv::exceptions::info::IOError*, *dv::exceptions::info::LengthError*, *dv::exceptions::info::NullPointer*, *dv::exceptions::info::OutOfRange*, *dv::exceptions::info::RuntimeError*

Public Types

using **Info** = void

struct **EndOfFile**

Public Types

using **Info** = *std::filesystem::path*

Public Static Functions

static inline *std::string* **format** (const *Info* &info)

struct **Epanechnikov**

Public Static Functions

static inline float **getSearchRadius** (const float bandwidth)

static inline float **apply** (const float squaredDistance, const float bandwidth)

struct **ErrorInfo**

Public Members

std::string mName

std::string mTypeIdentifier

struct **ErrorInfo**

Public Members

std::string mName

std::string mTypeIdentifier

struct **eSMGroup**

Public Functions

inline explicit **eSMGroup** (const uint32_t event)

Public Members

int16_t group1Address

int16_t group2Address

uint8_t group1Events

uint8_t group2Events

bool group1Polarity

bool group2Polarity

struct **eSMGroup**

Public Functions

inline explicit **eSMGroup** (const uint32_t event)

Public Members

int16_t group1Address

int16_t group2Offset

uint8_t group1Events

uint8_t group2Events

bool group1Polarity

bool group2Polarity

struct **eTimestampReference**

Public Functions

inline explicit **eTimestampReference** (const uint32_t event)

Public Members

int32_t timestampReference

struct **eTimestampReference**

Public Functions

inline explicit **eTimestampReference** (const uint32_t event)

Public Members

int32_t timestampReference

struct **eTimestampSubUnit**

Public Functions

inline explicit **eTimestampSubUnit** (const uint32_t event)

Public Members

int32_t timestampSubUnit

class **EventBlobDetector**

#include </builds/inivation/dv/dv-processing/include/dv-processing/features/event_blob_detector.hpp> Event-based blob detector performing detection on accumulated event images.

Public Functions

```
inline explicit EventBlobDetector (const cv::Size &resolution, const int pyramidLevel = 0,  
                                std::function<void(cv::Mat&> preprocess = {},  
                                cv::Ptr<cv::SimpleBlobDetector> blobDetector = defaultBlobDetector())
```

Constructor for blob detector.

The detection steps are as following: 1) Compute accumulated image from events 2) Apply ROI to the accumulated event image 3) Down sample image (if pyramidLevel >= 1) 4) Apply preprocess function (if exists) 5) Detect blobs 6) Rescale blobs to original resolution (if pyramidLevel >= 1) 7) If ROI has an offset from (0,0) of initial image plane, add offset back to bring blobs location in the original image space coordinate system 8) Remove blobs where mask value is 0.

Parameters

- **resolution** – original image plane resolution
- **pyramidLevel** – integer defining number of down samples applied to the accumulated image. E.g. if pyramidLevel = 3 -> we down sample the image by a factor of 2 for N=3 times. this means that an image of size (100, 100) is down sampled to (25, 25) before performing the blob detection. Note that blob location is always returned in the original resolution size.
- **preprocess** – function to be applied to the accumulated image before performing the detection step. The function modifies the input image passed as argument to the function in place. Internally, the api check that resolution and type of the image are kept.
- **blobDetector** – blob detector instance performing the detection step

```
inline std::vector<dv::TimedKeyPoint> detect (const dv::EventStore &events, const cv::Rect &roi = cv::Rect(),  
                                             const cv::Mat &mask = cv::Mat())
```

Detection step.

Parameters

- **events** – data used to create the accumulated image over which blob detection will be applied
- **roi** – region in which blobs will be searched
- **mask** – disable any blob detections on coordinates with zero pixel value on the mask.

Returns

blobs found from blob detector

Public Static Functions

```
static inline cv::Ptr<cv::SimpleBlobDetector> defaultBlobDetector ()
```

Create a reasonable default blob detector.

The method creates an instance of cv::SimpleBlobDetector with following parameter values:

- filterByArea = true
- minArea = 10 : minimum area of blobs to be detected - reasonable value to safely detect blobs and not noise in the accumulated image
- maxArea = 10000
- filterByCircularity = false
- filterByConvexity = false

- `filterByInertia = false`

Returns

blob detector used by default to detect interesting blobs

Private Members

`cv::Ptr<cv::SimpleBlobDetector> mBlobDetector`

Blob detector instance performing the detection step

`int32_t mPyramidLevel`

Number of `pyrDown` applied to the accumulated image

`std::function<void(cv::Mat&)> mPreprocessFcn`

Preprocessing function to be applied before the detection step

`dv::EdgeMapAccumulator mAccumulator`

Accumulator generating the image used for blob detection

`template<dv::concepts::EventToFrameConverter<dv::EventStore> AccumulatorType = dv::EdgeMapAccumulator>`

`class EventCombinedLKTracker : public dv::features::ImageFeatureLKTracker`

`#include </builds/inivation/dv/dv-processing/include/dv-processing/features/event_combined_lk_tracker.hpp>` Implements an event combined Lucas-Kanade tracker. The algorithm detects and tracks features on a regular frame image, but to improve tracking quality, it accumulates intermediate frames from events, performs tracking on those frames and uses the output to predict the track locations on the regular frame.

Template Parameters

`AccumulatorType` – *Accumulator* class to be used for frame generation.

Public Types

using `SharedPtr` = `std::shared_ptr<EventCombinedLKTracker>`

using `UniquePtr` = `std::unique_ptr<EventCombinedLKTracker>`

Public Functions

inline void `accept` (const `dv::EventStore` &store)

Add an event batch. Added events should contain at least some events that were registered further in the future of the next image.

Parameters

`store` – Batch of events.

inline const `std::vector<std::vector<cv::Point2f>> &getEventTrackPoints` () const

Get the intermediate tracking points on the event frames.

Returns

A vector of tracked points on the intermediate frames.

```
inline const std::vector<dv::features::ImagePyramid> &getAccumulatedFrames () const
```

Get a vector containing the intermediate accumulated frames.

Returns

A vector containing the intermediate accumulated frames.

```
inline dv::Duration getStoreTimeLimit () const
```

Get the event storage time limit.

Returns

Duration of the event storage in microseconds.

```
inline void setStoreTimeLimit (const dv::Duration storeTimeLimit)
```

Set the event buffer storage duration limit.

Parameters

storeTimeLimit – Storage duration limit in microseconds.

```
inline size_t getNumberOfEvents () const
```

Get the number of latest events that are going to be accumulated for each frame.

Returns

Number of accumulated events.

```
inline void setNumberOfEvents (const size_t numberOfEvents)
```

Set the number of latest events that are going to be accumulated for each frame.

Parameters

_numberOfEvents – Number of accumulated events.

```
inline int getNumIntermediateFrames () const
```

Get the number of intermediate frames that are going to be generated.

Returns

Number of intermediate frames between the frames.

```
inline void setNumIntermediateFrames (const int numIntermediateFrames)
```

Set the number of intermediate frames that are going to be generated.

Parameters

numIntermediateFrames – Number of intermediate frames between the frames.

```
inline void setAccumulator (std::unique_ptr<AccumulatorType> accumulator)
```

Set an accumulator instance to be used for frame generation. If a `nullptr` is passed, the function will instantiate an accumulator with no parameters (defaults).

Parameters

accumulator – An accumulator instance, can be `nullptr` to instantiate a default accumulator.

```
inline virtual void accept (const dv::measurements::Depth &timedDepth) override
```

Add scene depth, a median depth value of tracked landmarks usually works well enough.

Parameters

timedDepth – Depth measurement value (pair of timestamp and measured depth)

```
inline virtual void accept (const kinematics::Transformationf &transform) override
```

Add camera transformation, usually in the world coordinate frame (`TWC`). Although the class only extract the motion difference, so any other reference frame should also work as long as reference frames are not mixed up.

Parameters

transform – Camera pose represented by a transformation.

inline virtual void **accept** (const *dv::Frame* &image) override

Add an input image for the tracker. Image pyramid will be built from the given image.

Parameters

image – Acquired image.

inline double **getMinRateForIntermediateTracking** () const

Get the minimum event rate that is required to perform intermediate tracking.

Returns

Minimum event rate per second value.

inline void **setMinRateForIntermediateTracking** (const double minRateForIntermediateTracking)

Set a minimum event rate per second value that is used to perform intermediate. If the event rate between last and current frame is lower than this, tracker assumes very little motion and does not perform intermediate tracking.

Parameters

minRateForIntermediateTracking – Event rate (number of incoming events per second) required to perform intermediate tracking on accumulated frames.

inline virtual void **setConstantDepth** (const float depth) override

Set constant depth value that is assumed if no depth measurement is passed using `accept (dv::measurements::Depth)`. By default the constant depth is assumed to be 3.0 meters, which is just a reasonable guess.

This value is propagated into the accumulator if it supports constant depth setting.

Parameters

depth – Distance to the scene (depth).

Throws

`InvalidArgument` – Exception is thrown if a negative depth value is passed.

Public Static Functions

```
static inline EventCombinedLKTracker::UniquePtr RegularTracker (const cv::Size &resolution, const Config
&config = Config(),
std::unique_ptr<AccumulatorType>
accumulator = nullptr,
ImagePyrFeatureDetector::UniquePtr
detector = nullptr,
RedetectionStrategy::UniquePtr
redetection = nullptr)
```

Create a tracker instance that performs tracking of features on both - event accumulated and regular images. Tracking is performed by detecting and tracking features on a regular image. It also uses events to generate intermediate accumulated frames between the regular frames, track the features on them and use the intermediate tracking results as feature position priors for the image frame.

Parameters

- **resolution** – Sensor resolution
- **config** – Lucas-Kanade tracker configuration
- **accumulator** – The accumulator instance to be used for intermediate frame accumulation. Uses *dv::EdgeMapAccumulator* with default parameters if `nullptr` is passed.

- **detector** – Feature (corner) detector to be used. Uses `cv::Fast` with a threshold of 10 by default.
- **redetection** – Feature redetection strategy. By default, redetects features when feature count is below 0.5 of maximum value.

Returns

The tracker instance

```
static inline EventCombinedLKTracker::UniquePtr MotionAwareTracker (const camera-  
era::CameraGeometry::SharedPtr &camera, const Config &config =  
Config(),  
std::unique_ptr<AccumulatorType> accumulator = nullptr, kinemat-  
ics::PixelMotionPredictor::UniquePtr motionPredictor = nullptr, Im-  
agePyrFeatureDetector::UniquePtr detector = nullptr,  
RedetectionStrategy::UniquePtr redetection = nullptr)
```

Create a tracker instance that performs tracking of features on both - event accumulated and regular images. Tracking is performed by detecting and tracking features on a regular image. It also uses events to generate intermediate accumulated frames between the regular frames, track the features on them and use the intermediate tracking results as feature position priors for the image frame. The implementation also uses camera motion and scene depth to motion compensate events, so the intermediate accumulated frames are sharp and the Lucas-Kanade tracker works more accurately. This requires camera sensor to be calibrated.

Parameters

- **camera** – Camera geometry class instance, containing the intrinsic calibration of the camera sensor.
- **config** – Lucas-Kanade tracker configuration
- **accumulator** – The accumulator instance to be used for intermediate frame accumulation. Uses `dv::EdgeMapAccumulator` with default parameters if `nullptr` is passed.
- **motionPredictor** – Motion predictor class, by default it uses pixel reprojection `dv::kinematics::PixelMotionPredictor` without distortion model.
- **detector** – Feature (corner) detector to be used. Uses `cv::Fast` with a threshold of 10 by default.
- **redetection** – Feature redetection strategy. By default, redetects features when feature count is below 0.5 of maximum value.

Returns

The tracker instance

Protected Functions

```
inline std::vector<cv::Point2f> trackIntermediateEvents ()
```

Run the intermediate tracking on accumulated events. The `lastFrameResults` are modified if any of the intermediate tracks are lost. The predicted coordinates are returned which must match the indices of the keypoints in `lastFrameResults` `keypoint` list.

Returns

Predicted feature track locations that correspond to modified `lastFrameResults->keypoints` vector.

inline virtual Result::SharedPtr **track** () override

Perform the tracking.

Returns

Tracking result.

inline **EventCombinedLKTracker** (const cv::Size &resolution, const *ImageFeatureLKTracker::Config* &config)

Initialize the event combined Lucas-Kanade tracker and custom tracker parameters. It is going to use *EdgeMapAccumulator* with 15000 events and 0.25 event contribution. It will accumulate 3 intermediate frames from events to predict the track positions on regular frame.

Parameters

- **resolution** – Image resolution.
- **config** – Image tracker configuration.

inline **EventCombinedLKTracker** (const *camera::CameraGeometry::SharedPtr* &camera, const *ImageFeatureLKTracker::Config* &config)

Initialize the event combined Lucas-Kanade tracker and custom tracker parameters. It is going to use *EdgeMapAccumulator* with 15000 events and 0.25 event contribution. It will accumulate 3 intermediate frames from events to predict the track positions on regular frame.

Parameters

- **camera** – Camera geometry.
- **config** – Image tracker configuration.

Protected Attributes

std::unique_ptr<AccumulatorType> **mAccumulator** = nullptr

dv::Duration **mStoreTimeLimit** = *dv::Duration*(5000000)

size_t **mNumberOfEvents** = 20000

double **mMinRateForIntermediateTracking** = 0

int **mNumIntermediateFrames** = 3

dv::EventStore **mEventBuffer**

std::vector<dv::features::ImagePyramid> **mAccumulatedFrames**

std::vector<std::vector<cv::Point2f>> **mEventTrackPoints**

template<*concepts::EventToFrameConverter<dv::EventStore>* **AccumulatorType** = *dv::EdgeMapAccumulator*>

class **EventFeatureLKTracker** : public *dv::features::ImageFeatureLKTracker*

#include </builds/inivation/dv/dv-processing/include/dv-processing/features/event_feature_lk_tracker.hpp>
Event-based Lucas-Kanade tracker, the tracking is achieved by accumulating frames and running the classic LK frame based tracker on them.

Since the batch of events might contain information for more than a single tracking iteration configurable by the framerate parameter, the tracking function should be executed on a loop until it returns a null-pointer, signifying end of available data processing:

```
tracker.addEventInput(eventStore);  
while (auto result = tracker.runTracking()) {  
    // process the tracking result  
}
```

Template Parameters

AccumulatorType – *Accumulator* class to be used for frame generation.

Public Types

using **SharedPtr** = *std::shared_ptr*<*EventFeatureLKTracker*>

using **UniquePtr** = *std::unique_ptr*<*EventFeatureLKTracker*>

Public Functions

inline const cv::Mat &**getAccumulatedFrame**() const

Get the latest accumulated frame.

Returns

An accumulated frame.

inline int **getFramerate**() const

Get configured framerate.

Returns

Current accumulation and tracking framerate.

inline void **setFramerate**(int framerate)

Set the accumulation and tracking framerate.

Parameters

framerate_ – New accumulation and tracking framerate.

inline void **accept**(const *dv::EventStore* &store)

Add the input events. Since the batch of events might contain information for more than a single tracking iteration configurable by the framerate parameter, the tracking function should be executed on a loop until it returns a null-pointer, signifying end of available data processing:

```
tracker.addEventInput(eventStore);  
while (auto result = tracker.runTracking()) {  
    // process the tracking result  
}
```

Parameters

store – Event batch.

inline *dv::Duration* **getStoreTimeLimit**() const

Get the event storage time limit.

Returns

Duration of the event storage in microseconds.

inline void **setStoreTimeLimit** (const *dv::Duration* storeTimeLimit)

Set the event buffer storage duration limit.

Parameters

storeTimeLimit – Storage duration limit in microseconds.

inline size_t **getNumberOfEvents** () const

Get the number of latest events that are going to be accumulated for each frame. The default number of event is a third of of total pixels in the sensor.

Returns

Number of event to be accumulated.

inline void **setNumberOfEvents** (size_t numberOfEvents)

Set the number of latest events that are going to be accumulated for each frame. The default number of event is a third of of total pixels in the sensor.

Parameters

numberOfEvents – Number of accumulated events.

inline void **setAccumulator** (*std::unique_ptr<AccumulatorType>* accumulator)

Set an accumulator instance to be used for frame generation. If a `nullptr` is passed, the function will instantiate an accumulator with no parameters (defaults).

Parameters

accumulator – An accumulator instance, can be `nullptr` to instantiate a default accumulator.

inline virtual void **accept** (const *dv::measurements::Depth* &timedDepth) override

Add scene depth, a median depth value of tracked landmarks usually works well enough.

Parameters

timedDepth – Depth measurement value (pair of timestamp and measured depth)

inline virtual void **accept** (const *kinematics::Transformationf* &transform) override

Add camera transformation, usually in the world coordinate frame (T_{WC}). Although the class only extract the motion difference, so any other reference frame should also work as long as reference frames are not mixed up.

Parameters

transform – Camera pose represented by a transformation.

inline virtual void **setConstantDepth** (const float depth) override

Set constant depth value that is assumed if no depth measurement is passed using `accept (dv::measurements::Depth)`. By default the constant depth is assumed to be 3.0 meters, which is just a reasonable guess.

This value is used for predicting feature track positions when no depth measurements are passed in and also is propagated into the accumulator if it supports constant depth setting.

Parameters

depth – Distance to the scene (depth).

Throws

InvalidArgument – Exception is thrown if a negative depth value is passed.

Public Static Functions

```
static inline EventFeatureLKTracker::UniquePtr RegularTracker (const cv::Size &resolution, const Config
&config = Config(),
std::unique_ptr<AccumulatorType>
accumulator = nullptr,
ImagePyrFeatureDetector::UniquePtr
detector = nullptr,
RedetectionStrategy::UniquePtr redetection =
nullptr)
```

Create a tracker instance that performs tracking of features on event accumulated frames. Features are detected and tracked on event accumulated frames.

Parameters

- **resolution** – Sensor resolution
- **config** – Lucas-Kanade tracker configuration
- **accumulator** – The accumulator instance to be used for intermediate frame accumulation. Uses *dv::EdgeMapAccumulator* with default parameters if *nullptr* is passed.
- **detector** – Feature (corner) detector to be used. Uses *cv::Fast* with a threshold of 10 by default.
- **redetection** – Feature redetection strategy. By default, redetects features when feature count is below 0.5 of maximum value.

Returns

The tracker instance

```
static inline EventFeatureLKTracker::UniquePtr MotionAwareTracker (const
camera::CameraGeometry::SharedPtr
&camera, const Config &config =
Config(),
std::unique_ptr<AccumulatorType>
accumulator = nullptr, kinemat-
ics::PixelMotionPredictor::UniquePtr
motionPredictor = nullptr,
ImagePyrFeatureDetector::UniquePtr
detector = nullptr,
RedetectionStrategy::UniquePtr
redetection = nullptr)
```

Create a tracker instance that performs tracking of features on event accumulated frames. Features are detected and tracked on event accumulated frames. Additionally, camera motion and scene depth are used to generate motion compensated frames, which are way sharper than usual accumulated frames. This requires camera sensor to be calibrated.

Parameters

- **camera** – Camera geometry class instance, containing the intrinsic calibration of the camera sensor.
- **config** – Lucas-Kanade tracker configuration
- **accumulator** – The accumulator instance to be used for intermediate frame accumulation. Uses *dv::EdgeMapAccumulator* with default parameters if *nullptr* is passed.
- **motionPredictor** – Motion predictor class, by default it uses pixel reprojection *dv::kinematics::PixelMotionPredictor* without distortion model.

- **detector** – Feature (corner) detector to be used. Uses `cv::Fast` with a threshold of 10 by default.
- **redetection** – Feature redetection strategy. By default, redetects features when feature count is below 0.5 of maximum value.

Returns

The tracker instance

Protected Functions

inline virtual Result::SharedPtr **track** () override

Perform the tracking

Returns

Tracking result.

inline explicit **EventFeatureLKTracker** (const cv::Size &dimensions, const Config &config)

Initialize the event-frame tracker with default configuration: all the defaults of *ImageFeatureLKTracker* and a *EdgeMapAccumulator* executing at 50 FPS with and event count equal to third of the camera resolution and event contribution of 0.25.

Parameters

- **imageDimensions** – Image resolution.
- **config** – Lukas-Kanade tracker configuration.

inline explicit **EventFeatureLKTracker** (const *dv::camera::CameraGeometry::SharedPtr* &camera, const Config &config)

Initialize the event-frame tracker with default configuration: all the defaults of *ImageFeatureLKTracker* and a *EdgeMapAccumulator* executing at 50 FPS with and event count equal to third of the camera resolution and event contribution of 0.25.

Parameters

- **camera** – Camera geometry.
- **config** – Lukas-Kanade tracker configuration.

Protected Attributes

std::unique_ptr<AccumulatorType> **mAccumulator** = nullptr

int **mFramerate** = 50

int64_t **mPeriod** = 1000000 / *mFramerate*

int64_t **mLastRunTimestamp** = 0

dv::Duration **mStoreTimeLimit** = *dv::Duration*(5000000)

size_t **mNumberOfEvents**

The default number of event is a third of of total pixels in the sensor.

dv::EventStore **mEventBuffer**

cv::Mat **mAccumulatedFrame**

Private Functions

inline virtual void **accept** (const *dv::Frame* &image)

Add an input image for the tracker. Image pyramid will be built from the given image.

Parameters

image – Acquired image.

inline virtual void **accept** (const *dv::measurements::Depth* &timedDepth)

Add scene depth, a median depth value of tracked landmarks usually works well enough.

Parameters

timedDepth – Depth measurement value (pair of timestamp and measured depth)

inline virtual void **accept** (const *dv::kinematics::Transformationf* &transform)

Add camera transformation, usually in the world coordinate frame (T_{WC}). Although the class only extract the motion difference, so any other reference frame should also work as long as reference frames are not mixed up.

Parameters

transform – Camera pose represented by a transformation

template<*dv::concepts::EventStorage* **EventStoreClass** = *dv::EventStore*>

class **EventFilterBase**

#include </builds/inivation/dv/dv-processing/include/dv-processing/core/filters.hpp> A base class for noise filter implementations. Handles data input and output, derived classes only have to implement a retain function that tests whether event should be retained or discarded.

Subclassed by *dv::EventFilterChain< EventStoreClass >*, *dv::EventMaskFilter< EventStoreClass >*, *dv::EventPolarityFilter< EventStoreClass >*, *dv::EventRegionFilter< EventStoreClass >*, *dv::noise::BackgroundActivityNoiseFilter< EventStoreClass >*, *dv::noise::BaseFrequencyFilter< EventStoreClass >*, *dv::noise::FastDecayNoiseFilter< EventStoreClass >*, *dv::noise::KNoiseFilter< EventStoreClass >*

Public Functions

virtual **~EventFilterBase** () = default

inline void **accept** (const *EventStoreClass* &store)

Accepts incoming events.

Parameters

store – Event packet.

virtual bool **retain** (const typename *EventStoreClass::value_type* &event) noexcept = 0

A function to be implemented by derived class which tests whether given event should be retained or discarded.

Parameters

event – An event to be checked.

Returns

Return true if the event is to be retained or false to discard the event.

```
inline EventStoreClass generateEvents ()
```

Apply the filter algorithm and return only the filtered events from the ones that were accepted as input.

Returns

```
inline size_t getNumberOfIncomingEvents () const
```

Get number of total events that were accepted by the noise filter.

Returns

Total number of incoming events to this filter instance.

```
inline size_t getNumberOfOutgoingEvents () const
```

Total number of outgoing events from this filter instance.

Returns

Total number of outgoing events from this filter instance.

```
inline float getReductionFactor () const
```

Get the reduction factor of this filter. It's a fraction representation of events that were discard by this filter compared to the amount of incoming events.

Returns

Reduction factor value.

```
inline EventStoreClass &operator>> (EventStoreClass &out)
```

Retrieve filtered events using output stream operator.

Parameters

out – Filtered events.

Returns

Protected Attributes

```
EventStoreClass mBuffer = {}
```

```
int64_t mHighestProcessedTime = {-1}
```

```
size_t mNumberIncomingEvents = {0}
```

```
size_t mNumberOutgoingEvents = {0}
```

```
template<dv::concepts::EventStorage EventStoreClass = dv::EventStore>
```

```
class EventFilterChain : public dv::EventFilterBase<dv::EventStore>
```

#include </builds/inivation/dv/dv-processing/include/dv-processing/core/filters.hpp> Event filter based on multiple event filter applied sequentially. Internally stores any added filters and runs them one after another.

Template Parameters

EventStoreClass – Type of event store

Public Functions

```
inline void addFilter (std::shared_ptr<dv::EventFilterBase<EventStoreClass>> filter)
```

Add a filter to the chain of filtering.

Parameters**filter** –inline virtual bool **retain** (const typename *EventStoreClass*::value_type &event) noexcept override

Test whether event is of configured polarity.

Parameters**event** – Event to be checked.**Returns**

True if event has the expected polarity, false otherwise.

inline *EventFilterChain* &**operator<<** (const *EventStoreClass* &events)

Accept events using the input stream operator.

Parameters**events** – Input events.**Returns****Protected Attributes***std::vector<std::shared_ptr<dv::EventFilterBase<EventStoreClass>>>* **filters**template<*dv::concepts::EventStorage* **EventStoreClass** = *dv::EventStore*>class **EventMaskFilter** : public *dv::EventFilterBase<dv::EventStore>***Public Functions**inline explicit **EventMaskFilter** (const cv::Mat &mask)

Create an event masking filter. Discards any events that happen on coordinates where mask has a zero value and retains all events with coordinates where mask has a non-zero value.

Parameters**mask** – The mask to be applied (requires CV_8UC1 type).**Throws***InvalidArgument* – Exception thrown if the mask is of incorrect type.inline virtual bool **retain** (const typename *EventStoreClass*::value_type &event) noexcept override

A function to be implemented by derived class which tests whether given event should be retained or discarded.

Parameters**event** – An event to be checked.**Returns**

Return true if the event is to be retained or false to discard the event.

inline const cv::Mat &**getMask** () const

Get the mask that is currently applied.

Returnsinline void **setMask** (const cv::Mat &mask)

Set a new mask to this filter.

Parameters**mask** – The mask to be applied (requires CV_8UC1 type).

```
inline EventMaskFilter &operator<< (const EventStoreClass &events)
```

Accept events using the input stream operator.

Parameters

events – Input events.

Returns

Protected Attributes

```
cv::Mat mMask
```

```
struct EventPacket : public flatbuffers::NativeTable
```

Public Types

```
typedef EventPacketFlatbuffer TableType
```

Public Functions

```
inline EventPacket ()
```

```
inline EventPacket (const std::vector<Event> &_elements)
```

Public Members

```
std::vector<Event> elements
```

Public Static Functions

```
static inline constexpr const char *GetFullyQualifiedName ()
```

Friends

```
inline friend std::ostream &operator<< (std::ostream &os, const EventPacket &packet)
```

```
struct EventPacketBuilder
```

Public Functions

```
inline void add_elements (flatbuffers::Offset<flatbuffers::Vector<const Event*>> elements)
```

```
inline explicit EventPacketBuilder (flatbuffers::FlatBufferBuilder &_fbb)
```

```
EventPacketBuilder &operator= (const EventPacketBuilder &)
```

```
inline flatbuffers::Offset<EventPacketFlatbuffer> Finish ()
```

Public Members

flatbuffers::FlatBufferBuilder &**fbb_**

flatbuffers::uoffset_t **start_**

struct **EventPacketFlatbuffer** : private *flatbuffers::Table*

Public Types

typedef *EventPacket* **NativeTableType**

Public Functions

inline const *flatbuffers::Vector*<const Event*> ***elements** () const

inline bool **Verify** (*flatbuffers::Verifier* &verifier) const

inline *EventPacket* ***UnPack** (const *flatbuffers::resolver_function_t* *_resolver = nullptr) const

inline void **UnPackTo** (*EventPacket* *_o, const *flatbuffers::resolver_function_t* *_resolver = nullptr) const

Public Static Functions

static inline const *flatbuffers::TypeTable* ***MiniReflectTypeTable** ()

static inline constexpr const char ***GetFullyQualifiedName** ()

static inline void **UnPackToFrom** (*EventPacket* *_o, const *EventPacketFlatbuffer* *_fb, const
flatbuffers::resolver_function_t *_resolver = nullptr)

static inline *flatbuffers::Offset*<*EventPacketFlatbuffer*> **Pack** (*flatbuffers::FlatBufferBuilder* &**fbb**, const
EventPacket *_o, const
flatbuffers::rehasher_function_t *_rehasher =
nullptr)

Public Static Attributes

static constexpr const char ***identifier** = "EVTS"

template<*dv::concepts::EventStorage* **EventStoreClass** = *dv::EventStore*>

class **EventPolarityFilter** : public *dv::EventFilterBase*<*dv::EventStore*>

#include </builds/inivation/dv/dv-processing/include/dv-processing/core/filters.hpp> Event filter based on polarity.

Template Parameters

EventStoreClass – Type of event store

Public Functions

inline explicit **EventPolarityFilter** (const bool polarity)

Construct an event filter which filters out only events of given polarity.

Parameters

polarity – Extract events only of matching polarity.

inline bool **getPolarity** () const

Get the currently extracted polarity setting.

Returns

inline void **setPolarity** (const bool polarity)

Set a new polarity to extract (keep after filtering).

Parameters

polarity – Extract events only of matching polarity.

inline virtual bool **retain** (const typename *EventStoreClass*::value_type &event) noexcept override

Test whether event is of configured polarity.

Parameters

event – Event to be checked.

Returns

True if event has the expected polarity, false otherwise.

inline *EventPolarityFilter* &**operator<<** (const *EventStoreClass* &events)

Accept events using the input stream operator.

Parameters

events – Input events.

Returns

Protected Attributes

bool **mPolarity**

template<*dv::concepts::EventStorage* **EventStoreClass** = *dv::EventStore*>

class **EventRegionFilter** : public *dv::EventFilterBase*<*dv::EventStore*>

#include </builds/inivation/dv/dv-processing/include/dv-processing/core/filters.hpp> Event filter that filters events based on a given ROI.

Template Parameters

EventStoreClass – Type of event store

Public Functions

inline explicit **EventRegionFilter** (const cv::Rect region)

Filter event based on an ROI.

Parameters

region – Region of interest, events outside of this region will be discarded.

```
inline cv::Rect getRegion () const
```

Get the currently applied region of interest.

Returns

```
inline void setRegion (const cv::Rect region)
```

Set a new region of interest for events.

Parameters

region – Region of interest, events outside of this region will be discarded.

```
inline virtual bool retain (const typename EventStoreClass::value_type &event) noexcept override
```

Test whether event belongs to an ROI.

Parameters

event – Event to be checked.

Returns

True if event belongs to ROI, false otherwise.

```
inline EventRegionFilter &operator<< (const EventStoreClass &events)
```

Accept events using the input stream operator.

Parameters

events – Input events.

Returns

Protected Attributes

```
cv::Rect mRegion
```

```
template<dv::concepts::AddressableEvent EventType>
```

```
class EventTimeComparator
```

```
#include </builds/inivation/dv/dv-processing/include/dv-processing/core/core.hpp> INTERNAL USE ONLY
```

Compares an events timestamp to that of a timestamp.

Public Functions

```
inline bool operator () (const EventType &evt, const int64_t time) const
```

```
inline bool operator () (const int64_t time, const EventType &evt) const
```

```
class EventVisualizer
```

```
#include </builds/inivation/dv/dv-processing/include/dv-processing/visualization/event_visualizer.hpp> EventVisualizer class implements simple color-coded representation of events. It applies certain colors where positive or negative polarity events are registered.
```

Public Functions

```
inline explicit EventVisualizer (const cv::Size &resolution, const cv::Scalar &backgroundColor = colors::white, const cv::Scalar &positiveColor = colors::iniBlue, const cv::Scalar &negativeColor = colors::darkGray)
```

Initialize event visualizer.

Parameters

- **resolution** – Resolution of incoming events.
- **backgroundColor** – Background color.
- **positiveColor** – Color applied to positive polarity events.
- **negativeColor** – Color applied to negative polarity events.

inline cv::Mat **generateImage** (const *dv::EventStore* &events) const

Generate a preview image from an event store.

Parameters

events – Input events.

Returns

Colored preview image of given events.

inline void **generateImage** (const *dv::EventStore* &events, cv::Mat &background) const

Generate a preview image from an event store.

Parameters

- **events** – Input events.
- **background** – Image to draw the events on. The pixels type has to be 3-channel 8-bit unsigned integer (BGR).

inline cv::Scalar **getBackgroundColor** () const

Get currently configured background color.

Returns

Background color.

inline void **setBackgroundColor** (const cv::Scalar &backgroundColor_)

Set new background color.

Parameters

backgroundColor_ – New background color.

inline cv::Scalar **getPositiveColor** () const

Get currently configured positive polarity color.

Returns

Positive polarity color.

inline void **setPositiveColor** (const cv::Scalar &positiveColor_)

Set new positive polarity color.

Parameters

positiveColor_ – New positive polarity color.

inline cv::Scalar **getNegativeColor** () const

Get negative polarity color.

Returns

Negative polarity color.

inline void **setNegativeColor** (const cv::Scalar &negativeColor_)

Set new negative polarity color.

Parameters

negativeColor_ – New negative polarity color.

Private Members

const cv::Size **resolution**

cv::Vec3b **backgroundColor**

cv::Vec3b **positiveColor**

cv::Vec3b **negativeColor**

class **Exception** : public *std::exception*

Public Functions

inline explicit **Exception** (const *std::source_location* &location = *std::source_location::current*(), const boost::stacktrace::stacktrace &stacktrace = boost::stacktrace::stacktrace(), const *std::string_view* type = boost::core::demangle(typeid(*Exception*).name()))

inline explicit **Exception** (const *std::string_view* whatInfo, const *std::source_location* &location = *std::source_location::current*(), const boost::stacktrace::stacktrace &stacktrace = boost::stacktrace::stacktrace(), const *std::string_view* type = boost::core::demangle(typeid(*Exception*).name()))

~Exception () override = default

Exception (const *Exception* &other) = default

Exception (*Exception* &&other) = default

inline *Exception* **operator<<** (const *std::string_view* info)

inline const char ***what** () const noexcept override

inline const char ***shortWhat** () const noexcept

Protected Attributes

std::string **mInfo**

std::string **mShortInfo**

Private Functions

inline void **createInfo** (const *std::string_view* whatInfo, const *std::string_view* file, const *std::string_view* function, const uint32_t line, const *std::string_view* stacktrace, const *std::string_view* type)

template<typename **EXCEPTION_TYPE**, typename **BASE_TYPE** = *Exception*>

class **Exception_** : public *dv::exceptions::Exception*

Public Types

```
using Info = typename EXCEPTION_TYPE::Info
```

Public Functions

```
template<internal::HasExtraExceptionInfo T = EXCEPTION_TYPE>
inline Exception_ (const std::string_view whatInfo, const typename T::Info &errorInfo, const
    std::source_location &location = std::source_location::current(), const
    boost::stacktrace::stacktrace &stacktrace = boost::stacktrace::stacktrace(), const
    std::string_view type = boost::core::demangle(typeid(EXCEPTION_TYPE).name()))
```

```
template<internal::HasExtraExceptionInfo T = EXCEPTION_TYPE>
inline Exception_ (const typename T::Info &errorInfo, const std::source_location &location =
    std::source_location::current(), const boost::stacktrace::stacktrace &stacktrace =
    boost::stacktrace::stacktrace(), const std::string_view type =
    boost::core::demangle(typeid(EXCEPTION_TYPE).name()))
```

```
inline Exception_ (const std::string_view whatInfo, const std::source_location &location =
    std::source_location::current(), const boost::stacktrace::stacktrace &stacktrace =
    boost::stacktrace::stacktrace(), const std::string_view type =
    boost::core::demangle(typeid(EXCEPTION_TYPE).name()))
```

```
inline Exception_ (const std::source_location &location = std::source_location::current(), const
    boost::stacktrace::stacktrace &stacktrace = boost::stacktrace::stacktrace(), const
    std::string_view type = boost::core::demangle(typeid(EXCEPTION_TYPE).name()))
```

```
~Exception_() override = default
```

```
Exception_ (const Exception_ &other) = default
```

```
Exception_ (Exception_ &&other) = default
```

```
template<internal::HasExtraExceptionInfo T = EXCEPTION_TYPE>
inline Exception_ operator<< (const typename T::Info &errorInfo)
```

```
inline Exception_ operator<< (const std::string_view whatInfo)
```

```
template<dv::concepts::EventStorage EventStoreClass = dv::EventStore>
```

```
class FastDecayNoiseFilter : public dv::EventFilterBase<dv::EventStore>
```

Public Functions

```
inline explicit FastDecayNoiseFilter (const cv::Size &resolution, const dv::Duration halfLife =
    dv::Duration(10 '000), const uint32_t subdivisionFactor = 4,
    const float noiseThreshold = 6.f)
```

Create a fast decay noise filter. This filter uses a concept that performs a fast decay on a low resolution representation of the image and checks whether corresponding neighbourhood of the event has recent activity.

Parameters

- **resolution** – Sensor resolution.
- **halfLife** – Half-life is the amount of time it takes for the internal event counter to halve. Decreasing this will increase the strength of the noise filter (cause it to reject more events).

- **subdivisionFactor** – Subdivision factor, this is used calculate a low resolution image dimensions used for the fast decay operations.
- **noiseThreshold** – Noise threshold value, amount of filtered events can be increased by decreasing this value.

inline virtual bool **retain** (const typename *EventStoreClass*::value_type &event) noexcept override

Test whether to retain this event.

Parameters

event – Event to be checked.

Returns

True to retain an event, false to discard it.

inline *FastDecayNoiseFilter* &**operator<<** (const *EventStoreClass* &events)

Accept events using the input stream operator.

Parameters

events – Input events.

Returns

inline float **getNoiseThreshold** () const

Get the currently configured noise threshold.

Returns

Noise threshold value.

inline void **setNoiseThreshold** (const float noiseThreshold)

Set a new noise threshold value.

Parameters

noiseThreshold – Noise threshold value.

inline *dv::Duration* **getHalfLife** () const

Get the current configured half-life value.

Half-life is the amount of time it takes for the internal event counter to halve. Decreasing this will increase the strength of the noise filter (cause it to reject more events).

Returns

Currently configured event counter half life value.

inline void **setHalfLife** (const *dv::Duration* halfLife)

Set a new counter half-life value.

Half-life is the amount of time it takes for the internal event counter to halve. Decreasing this will increase the strength of the noise filter (cause it to reject more events).

Parameters

halfLife – New event counter half life value.

Protected Attributes

uint32_t **mSubdivisionFactor** = 4

cv::Mat **mDecayLUT**

dv::TimeSurface **mTimeSurface**

float **mNoiseThreshold** = 6.f

float **mHalfLifeMicros** = 10'000.f

class **FeatureCountRedetection** : public *dv::features::RedetectionStrategy*

#include </builds/inivation/dv/dv-processing/include/dv-processing/features/redetection_strategy.hpp> Redetection strategy based on number of features.

Public Functions

inline explicit **FeatureCountRedetection** (float minimumProportionOfTracks)

Redetection strategy based on number of features.

Parameters

minimumProportionOfTracks – Feature count coefficient, redetection is performed when feature count goes lower than the given proportion of maximum tracks, redetection will be executed.

inline virtual bool **decideRedetection** (const *TrackerBase* &tracker) override

Check whether to perform redetection.

Parameters

tracker – Current state of the tracker.

Returns

True to perform redetection of features, false to continue.

Protected Attributes

float **mMinimumProportionOfTracks** = 0.5f

template<class **InputType**, *dv::concepts::FeatureDetectorAlgorithm*<*InputType*> **Algorithm**>

class **FeatureDetector**

#include </builds/inivation/dv/dv-processing/include/dv-processing/features/feature_detector.hpp> A base class to implement feature detectors on different input types, specifically either images, time surfaces, or event stores. The implementing class should override the `detect` function and output a vector of unordered features with a quality score. The API will handle margin calculations and post processing of the features.

➡ See also

dv::concepts::FeatureDetectorAlgorithm

Template Parameters

- **InputType** – The type of input that is needed for the detector.
- **Algorithm** – The underlying detection algorithm, can be an `OpenCV::Feature2D` algorithm or a custom implementation, as long as it satisfies

Public Types

enum class **FeaturePostProcessing**

Feature post processing step performed after the the features were detected. Currently available types of post processing:

- **None**: Do not perform any postprocessing, all keypoints from detection will be returned.
- **TopN**: Retrieve the top number of highest scoring features.
- **AdaptiveNMS**: Apply the AdaptiveNMS algorithm to retrieve equally spaced keypoints in pixel space dimensions. More information on the AdaptiveNMS here: original code: <https://github.com/BAILLOOL/ANMS-Codes> paper: https://www.researchgate.net/publication/323388062_Efficient_adaptive_non-maximal_suppression_algorithms_for_homogeneous_spatial_keypoint_distribution

Values:

enumerator **NONE**

enumerator **TOP_N**

enumerator **ADAPTIVE_NMS**

using **ThisType** = *FeatureDetector*<*InputType*, *Algorithm*>

using **SharedPtr** = *std::shared_ptr*<*ThisType*>

using **UniquePtr** = *std::unique_ptr*<*ThisType*>

using **AlgorithmPtr** = typename *std::conditional_t*<*std::is_base_of*<*cv::Feature2D*, *Algorithm*>, *cv::Ptr*<*Algorithm*>, *std::shared_ptr*<*Algorithm*>>

Public Functions

inline **FeatureDetector** (const *cv::Size* &*_imageDimensions*, const *AlgorithmPtr* &*_detector*, const *FeaturePostProcessing* *_postProcessing*, const float *_margin* = 0.f)

Create a feature detector.

See also

FeatureDetectorBase::FeaturePostProcessing

Parameters

- **_imageDimensions** – Image dimensions.
- **_postProcessing** – Post processing step - subsampling of events,
- **_margin** – Margin coefficient, it will be multiplied by the width and height of the image to calculate an adaptive border alongside the edges of image, where features should not be detected.

inline explicit **FeatureDetector** (const cv::Size &_imageDimensions, const *AlgorithmPtr* &_detector)

Create a feature detector. This constructor defaults post-processing step to AdaptiveNMS and margin coefficient value of 0.02.

Parameters

_imageDimensions – Image dimensions.

virtual **~FeatureDetector** () = default

Destructor

inline *std::vector*<*dv::TimedKeyPoint*> **runDetection** (const *InputType* &input, const size_t numPoints = *FIND_ALL*, const cv::Mat &mask = cv::Mat())

Public detection call. Calls the overloaded `detect` function, applies margin and post processing.

Parameters

- **input** – The input to the detector
- **numPoints** – Number of keypoints to be detected
- **mask** – Detection mask, detection will be performed where mask value is non-zero.

Returns

A list of keypoints with timestamp.

inline void **runRedetection** (*std::vector*<*dv::TimedKeyPoint*> &prior, const *InputType* &input, const size_t numPoints = *FIND_ALL*, const cv::Mat &mask = cv::Mat())

Redetect new features and add them to already detected features. This function performs detection within masked region (if mask is non-empty), runs postprocessing and appends the additional features to the prior keypoint list.

Parameters

- **prior** – A list of existing features.
- **input** – The input to the detector (events, images, etc.).
- **numPoints** – Number of total features after detection.
- **mask** – Detection mask.

inline *FeaturePostProcessing* **getPostProcessing** () const

Get the type of post-processing.

See also

FeatureDetectorBase::FeaturePostProcessing

Returns

Type of post-processing.

inline void **setPostProcessing** (*FeaturePostProcessing* _postProcessing)

Set the type of post-processing.

See also

FeatureDetectorBase::FeaturePostProcessing

Parameters

_postProcessing – Type of post-processing.

inline float **getMargin** () const

Get currently applied margin coefficient. Margin coefficient is multiplied by the width and height of the image to calculate an adaptive border alongside the edges of image, where features should not be detected.

Returns

The margin coefficient.

inline void **setMargin** (float _margin)

Set the margin coefficient. Margin coefficient is multiplied by the width and height of the image to calculate an adaptive border alongside the edges of image, where features should not be detected.

Parameters

_margin – The margin coefficient

inline bool **isWithinROI** (const cv::Point2f &point) const

Check whether a point belongs to the ROI without the margins.

Parameters

point – Point to be checked

Returns

True if point belongs to the valid ROI, false otherwise.

inline const cv::Size &**getImageDimensions** () const

Get configured image dimensions.

Returns

Image dimensions.

Public Static Attributes

static constexpr size_t **FIND_ALL** = {std::numeric_limits<size_t>::max()}

Private Functions

inline std::vector<dv::TimedKeyPoint> **detect** (const InputType &input, const cv::Rect &roi, const cv::Mat &mask)

The detection function to be implemented for feature detection. It should return a list of keypoints with a quality score, but it should *not* be ordered in any way. The sorting will be performed by the `runDetection` function as a postprocessing step.

Parameters

- **input** – Input for the detector.
- **roi** – Region of interest where detection should be performed, the region is estimated using the margin configuration value.
- **mask** – Detection mask, can be empty. If non empty, the detection should be performed where mask value is non-zero.

Returns

A list of keypoint features with timestamp.

inline cv::Rect **getMarginROI** () const

Calculate the region of interest with the margin coefficient. Margin is a coefficient of width / height, which should be used to ignore pixels near borders of the image.

Returns

Region of interest for detection of features.

Private Members

FeaturePostProcessing **postProcessing**

float **margin**

cv::Size **imageDimensions**

cv::Rect **roiBuffered**

AlgorithmPtr **detector**

Container of the feature detector

int **classIdCounter** = 0

Class id counter, each new feature will be assigned on incremented class id.

KeyPointResampler **resampler**

Friends

inline friend *std::ostream* &**operator<<** (*std::ostream* &os, const *FeaturePostProcessing* &var)

class **FeatureTracks**

#include </builds/inivation/dv/dv-processing/include/dv-processing/features/feature_tracks.hpp> A class to store a time limited amount of feature tracks. Sorts and stores the data in separate queues for each track id. Provides *visualize* function to generate visualization images of the tracks.

Public Functions

inline void **accept** (const *dv::TimedKeyPoint* &keypoint)

Add a keypoint measurement into the feature track.

Parameters

keypoint – Single keypoint measurement.

inline void **accept** (const *dv::TimedKeyPointPacket* &keypoints)

Add a set of keypoint measurements into the feature track.

Parameters

keypoints – Vector of keypoint measurements.

inline void **accept** (const cv::KeyPoint &keypoint)

Add OpenCV type keypoint. It is missing a timestamp, so current system clock time will be used for the timestamp.

Parameters

keypoint – KeyPoint measurement.

inline void **accept** (const *TrackerBase::Result::ConstPtr* &trackingResult)

Add keypoint tracking result from a tracker.

Parameters

trackingResult – Tracking results.

inline *Duration* **getHistoryDuration** () const

Retrieve the history duration.

Returns

Currently applied track history time limit.

inline void **setHistoryDuration** (const *dv::Duration* historyDuration)

Set new history duration limit to buffer. If the new limit is shorter than the previously set, the tracks will be reduced to the new limit right away.

Parameters

historyDuration – New time limit for the track history buffer.

inline *std::optional<std::shared_ptr<const std::deque<dv::TimedKeyPoint>>>* **getTrack** (const int32_t trackId)
const

Retrieve a track of given track id.

Parameters

trackId – Track id to retrieve.

Returns

A pointer to feature track history, *std::nullopt* if unavailable.

inline *std::vector<int32_t>* **getTrackIds** () const

Return all track ids that are available in the buffer.

Returns

A vector containing track ids store in the history buffer.

inline *dv::TimedKeyPointPacket* **getLatestTrackKeypoints** ()

Return last keypoint from all tracks in the history.

Returns

inline void **eachTrack** (const *std::function<void(const int32_t, const std::shared_ptr<const std::deque<dv::TimedKeyPoint>>&>* &callback) const

Run a callback function to each of the stored tracks.

Parameters

callback – Callback function that is going to be called for each of the tracks, tracks are passed into the callback function as arguments.

inline cv::Mat **visualize** (const cv::Mat &background) const

Draws tracks on the input image, by default uses neon color palette from the *dv::visualization::colors* namespace for each of the tracks.

Parameters

background – Background image to be used for tracks.

Throws

InvalidArgument – An *InvalidArgument* exception is thrown if an empty image is passed as background.

Returns

Input image with drawn colored feature tracks.

```
inline bool isEmpty () const
```

Checks whether the feature track history buffer is empty.

Returns

True if there are no feature keypoints in the buffer.

```
inline void clear ()
```

Deletes any data stored in feature track buffer and resets visualization image.

```
inline const std::optional<dv::Duration> &getTrackTimeout () const
```

Get the track timeout value.

 See also

setTrackTimeout

Returns

Current track timeout value.

```
inline void setTrackTimeout (const std::optional<dv::Duration> &trackTimeout)
```

Set the track timeout value, pass `std::nullopt` to disable the this feature at all. Track latest timestamp is going to be compared to highest received timestamp in accept method, if the value is exceeded the track is going to be removed. This is useful to remove lost tracks without waiting for the track history to remove it, consider setting it to 2x of tracking rate, so tracks will remove if the track is not updated for two consecutive frames.

By default the feature is disabled, so lost tracks are kept until it's removed by the history time limit.

Parameters

trackTimeout – Track timeout value or `std::nullopt` to disable the feature.

```
inline int64_t getHighestTime ()
```

Return latest time from all existing tracks.

Private Functions

```
inline void addKeypoint (const dv::TimedKeyPoint &keypoint)
```

Add a keypoint measurement

Parameters

keypoint – Keypoint measurement

```
inline void maintainBufferDuration ()
```

Check the whole buffer for out-of-limit data, remove any tracks that do not contain any measurements.

Private Members

```
std::map<int32_t, std::shared_ptr<std::deque<dv::TimedKeyPoint>>>> mHistory
```

```
dv::Duration mHistoryDuration = dv::Duration(500'000)
```

```
std::optional<dv::Duration> mTrackTimeout = std::nullopt
```

```
int64_t mHighestTime = -1
```

```
struct FileDataDefinition : public flatbuffers::NativeTable
```

Public Types

```
typedef FileDataDefinitionFlatbuffer TableType
```

Public Functions

```
inline FileDataDefinition ()
```

```
inline FileDataDefinition (int64_t _ByteOffset, const PacketHeader &_PacketInfo, int64_t _NumElements,  
                           int64_t _TimestampStart, int64_t _TimestampEnd)
```

Public Members

```
int64_t ByteOffset
```

```
PacketHeader PacketInfo
```

```
int64_t NumElements
```

```
int64_t TimestampStart
```

```
int64_t TimestampEnd
```

Public Static Functions

```
static inline constexpr const char *GetFullyQualifiedName ()
```

```
struct FileDataDefinitionBuilder
```

Public Functions

```
inline void add_ByteOffset (int64_t ByteOffset)
```

```
inline void add_PacketInfo (const PacketHeader *PacketInfo)
```

```
inline void add_NumElements (int64_t NumElements)
```

```
inline void add_TimestampStart (int64_t TimestampStart)
```

```
inline void add_TimestampEnd (int64_t TimestampEnd)
```

```
inline explicit FileDataDefinitionBuilder (flatbuffers::FlatBufferBuilder &_fbb)
```

```
FileDataDefinitionBuilder &operator= (const FileDataDefinitionBuilder&)
```

```
inline flatbuffers::Offset<FileDataDefinitionFlatbuffer> Finish ()
```

Public Members

```
flatbuffers::FlatBufferBuilder &fbb_
```

```
flatbuffers::uoffset_t start_
```

```
struct FileDataDefinitionFlatbuffer : private flatbuffers::Table
```

Public Types

```
typedef FileDataDefinition NativeTableType
```

Public Functions

```
inline int64_t ByteOffset () const
```

```
inline const PacketHeader *PacketInfo () const
```

```
inline int64_t NumElements () const
```

```
inline int64_t TimestampStart () const
```

```
inline int64_t TimestampEnd () const
```

```
inline bool Verify (flatbuffers::Verifier &verifier) const
```

```
inline FileDataDefinition *UnPack (const flatbuffers::resolver_function_t *_resolver = nullptr) const
```

```
inline void UnPackTo (FileDataDefinition *_o, const flatbuffers::resolver_function_t *_resolver = nullptr) const
```

Public Static Functions

```
static inline const flatbuffers::TypeTable *MiniReflectTypeTable ()
```

```
static inline constexpr const char *GetFullyQualifiedName ()
```

```
static inline void UnPackToFrom (FileDataDefinition *_o, const FileDataDefinitionFlatbuffer *_fb, const  

flatbuffers::resolver_function_t *_resolver = nullptr)
```

```
static inline flatbuffers::Offset<FileDataDefinitionFlatbuffer> Pack (flatbuffers::FlatBufferBuilder &_fbb, const  

FileDataDefinition *_o, const  

flatbuffers::rehasher_function_t *_rehasher =  

nullptr)
```

```
struct FileDataTable : public flatbuffers::NativeTable
```

Public Types

```
typedef FileDataTableFlatbuffer TableType
```

Public Functions

```
inline FileDataTable ()
```

```
inline FileDataTable (const std::vector<FileDataDefinition> &_Table)
```

Public Members

```
std::vector<FileDataDefinition> Table
```

Public Static Functions

```
static inline constexpr const char *GetFullyQualifiedName ()
```

```
struct FileDataTableBuilder
```

Public Functions

```
inline void add_Table (flat-  
                          buffers::Offset<flatbuffers::Vector<flatbuffers::Offset<FileDataDefinitionFlatbuffer>>>  
                          Table)
```

```
inline explicit FileDataTableBuilder (flatbuffers::FlatBufferBuilder &_fbb)
```

```
FileDataTableBuilder &operator= (const FileDataTableBuilder &)
```

```
inline flatbuffers::Offset<FileDataTableFlatbuffer> Finish ()
```

Public Members

```
flatbuffers::FlatBufferBuilder &fbb_
```

```
flatbuffers::uoffset_t start_
```

```
struct FileDataTableFlatbuffer : private flatbuffers::Table
```

Public Types

```
typedef FileDataTable NativeTableType
```

Public Functions

```
inline const flatbuffers::Vector<flatbuffers::Offset<FileDataDefinitionFlatbuffer>> *Table () const
```

```
inline bool Verify (flatbuffers::Verifier &verifier) const
```

```
inline FileDataTable *UnPack (const flatbuffers::resolver_function_t *_resolver = nullptr) const
inline void UnPackTo (FileDataTable *_o, const flatbuffers::resolver_function_t *_resolver = nullptr) const
```

Public Static Functions

```
static inline const flatbuffers::TypeTable *MiniReflectTypeTable ()
static inline constexpr const char *GetFullyQualifiedName ()
static inline void UnPackToFrom (FileDataTable *_o, const FileDataTableFlatbuffer *_fb, const
                                flatbuffers::resolver_function_t *_resolver = nullptr)
static inline flatbuffers::Offset<FileDataTableFlatbuffer> Pack (flatbuffers::FlatBufferBuilder &fbb, const
                                                                FileDataTable *_o, const
                                                                flatbuffers::rehasher_function_t *_rehasher =
                                                                nullptr)
```

Public Static Attributes

```
static constexpr const char *identifier = "FTAB"
```

```
struct FileError
```

Public Types

```
using Info = std::filesystem::path
```

Public Static Functions

```
static inline std::string format (const Info &info)
```

```
struct FileInfo
```

Public Members

```
uint64_t mFileSize
dv::CompressionType mCompression
int64_t mDataTablePosition
int64_t mDataTableSize
dv::FileDataTable mDataTable
int64_t mTimeLowest
```

```
int64_t mTimeHighest
```

```
int64_t mTimeDifference
```

```
int64_t mTimeShift
```

```
std::vector<dv::io::Stream> mStreams
```

```
std::unordered_map<int32_t, dv::FileDataTable> mPerStreamDataTables
```

```
struct FileNotFound
```

Public Types

```
using Info = std::filesystem::path
```

Public Static Functions

```
static inline std::string format (const Info &info)
```

```
struct FileOpenError
```

Public Types

```
using Info = std::filesystem::path
```

Public Static Functions

```
static inline std::string format (const Info &info)
```

```
struct FileReadError
```

Public Types

```
using Info = std::filesystem::path
```

Public Static Functions

```
static inline std::string format (const Info &info)
```

```
struct FileWriteError
```

Public Types

```
using Info = std::filesystem::path
```

Public Static Functions

static inline *std::string* **format** (const *Info* &info)

template<>

struct **formatter**<cv::Mat> : public *fmt::formatter*<*std::string_view*>

Public Functions

inline auto **format** (const cv::Mat &var, *fmt::format_context* &ctx) const -> *fmt::format_context::iterator*

template<>

struct **formatter**<cv::Point> : public *fmt::ostream_formatter*

template<>

struct **formatter**<cv::Rect> : public *fmt::ostream_formatter*

template<>

struct **formatter**<cv::Size> : public *fmt::ostream_formatter*

template<>

struct **formatter**<*dv::Accumulator::Decay*> : public *fmt::ostream_formatter*

template<>

struct **formatter**<*dv::BoundingBoxPacket*> : public *fmt::ostream_formatter*

template<>

struct **formatter**<*dv::camera::CameraGeometry::FunctionImplementation*> : public *fmt::ostream_formatter*

template<>

struct **formatter**<*dv::camera::DistortionModel*> : public *fmt::ostream_formatter*

template<>

struct **formatter**<*dv::camera::StereoGeometry::CameraPosition*> : public *fmt::ostream_formatter*

template<>

struct **formatter**<*dv::camera::StereoGeometry::FunctionImplementation*> : public *fmt::ostream_formatter*

template<>

struct **formatter**<*dv::DepthEventPacket*> : public *fmt::ostream_formatter*

template<>

struct **formatter**<*dv::DepthFrame*> : public *fmt::ostream_formatter*

template<>

struct **formatter**<*dv::EdgeMapAccumulator::DecayMode*> : public *fmt::ostream_formatter*

template<>

```
struct formatter<dv::Event> : public fmt::ostream_formatter
template<>
struct formatter<dv::EventColor> : public fmt::ostream_formatter
template<>
struct formatter<dv::EventPacket> : public fmt::ostream_formatter
template<>
struct formatter<dv::EventStore> : public fmt::ostream_formatter
template<>
struct formatter<dv::Frame> : public fmt::ostream_formatter
template<>
struct formatter<dv::IMUPacket> : public fmt::ostream_formatter
template<>
struct formatter<dv::io::camera::CameraInputBase::Flatten> : public fmt::ostream_formatter
template<>
struct formatter<dv::io::camera::CameraModel> : public fmt::ostream_formatter
template<>
struct formatter<dv::io::camera::DVXplorer::ReadoutFPS> : public fmt::ostream_formatter
template<>
struct formatter<dv::io::camera::DVXplorer::SubSample> : public fmt::ostream_formatter
template<>
struct formatter<dv::io::camera::imu::ImuModel> : public fmt::ostream_formatter
template<>
struct formatter<dv::io::camera::parser::DAVIS::ColorMode> : public fmt::ostream_formatter
template<>
struct formatter<dv::io::camera::parser::DAVIS::SensorModel> : public fmt::ostream_formatter
template<>
struct formatter<dv::io::camera::USBDevice::LogLevel> : public fmt::ostream_formatter
template<>
struct formatter<dv::io::camera::USBDeviceType> : public fmt::ostream_formatter
template<>
class formatter<dv::io::support::VariantValueOwning>
```

Public Functions

```
inline constexpr auto parse (const format_parse_context &ctx)
```

```
template<typename FormatContext>
```

```
inline auto format (const dv::io::support::VariantValueOwning &obj, FormatContext &ctx) const
```

Private Members

```
std::array<char, FORMATTER_MAX_LEN> mFmtForward = {}
```

Private Static Attributes

```
static constexpr size_t FORMATTER_MAX_LEN = {32}
```

```
template<>
```

```
struct formatter<dv::LandmarksPacket> : public fmt::ostream_formatter
```

```
template<>
```

```
struct formatter<dv::noise::FrequencyFilterType> : public fmt::ostream_formatter
```

```
template<>
```

```
struct formatter<dv::PixelArrangement> : public fmt::ostream_formatter
```

```
template<>
```

```
struct formatter<dv::Pose> : public fmt::ostream_formatter
```

```
template<>
```

```
struct formatter<dv::TimedKeyPointPacket> : public fmt::ostream_formatter
```

```
template<>
```

```
struct formatter<dv::TimeSlicingApproach> : public fmt::ostream_formatter
```

```
template<>
```

```
struct formatter<dv::TriggerPacket> : public fmt::ostream_formatter
```

```
template<>
```

```
struct formatter<dv::visualization::PoseVisualizer::GridPlane> : public fmt::ostream_formatter
```

```
template<>
```

```
struct formatter<dv::visualization::PoseVisualizer::ViewMode> : public fmt::ostream_formatter
```

```
struct Frame : public flatbuffers::NativeTable
```

Public Types

typedef *FrameFlatbuffer* **TableType**

Public Functions

inline **Frame** ()

inline **Frame** (int64_t _timestamp, int64_t _timestampStartOfFrame, int64_t _timestampEndOfFrame, int64_t _timestampStartOfExposure, int64_t _timestampEndOfExposure, *FrameFormat* _format, int16_t _sizeX, int16_t _sizeY, int16_t _positionX, int16_t _positionY, const *std::vector*<uint8_t> &_pixels)

inline **Frame** (int64_t _timestamp, int64_t _exposure, int16_t _positionX, int16_t _positionY, const cv::Mat &_image, *dv::FrameSource* _source)

inline **Frame** (int64_t _timestamp, const cv::Mat &_image)

Public Members

int64_t **timestamp**

int16_t **positionX**

int16_t **positionY**

cv::Mat **image**

dv::Duration **exposure**

FrameSource **source**

Public Static Functions

static inline constexpr const char ***GetFullyQualifiedName** ()

Friends

inline friend *std::ostream* &**operator**<< (*std::ostream* &os, const *Frame* &frame)

struct **FrameBuilder**

Public Functions

inline void **add_timestamp** (int64_t timestamp)

inline void **add_timestampStartOfFrame** (int64_t timestampStartOfFrame)

inline void **add_timestampEndOfFrame** (int64_t timestampEndOfFrame)

```

inline void add_timestampStartOfExposure (int64_t timestampStartOfExposure)

inline void add_timestampEndOfExposure (int64_t timestampEndOfExposure)

inline void add_format (FrameFormat format)

inline void add_sizeX (int16_t sizeX)

inline void add_sizeY (int16_t sizeY)

inline void add_positionX (int16_t positionX)

inline void add_positionY (int16_t positionY)

inline void add_pixels (flatbuffers::Offset<flatbuffers::Vector<uint8_t>> pixels)

inline void add_exposure (int64_t exposure)

inline void add_source (FrameSource source)

inline explicit FrameBuilder (flatbuffers::FlatBufferBuilder &_fbb)

FrameBuilder &operator= (const FrameBuilder&)

inline flatbuffers::Offset<FrameFlatbuffer> Finish ()

```

Public Members

```

flatbuffers::FlatBufferBuilder &fbb_

flatbuffers::uoffset_t start_

```

```
struct FrameFlatbuffer : private flatbuffers::Table
```

Public Types

```
typedef Frame NativeTableType
```

Public Functions

```

inline int64_t timestamp () const
    Central timestamp (μs), corresponds to exposure midpoint.

inline int64_t timestampStartOfFrame () const
    Start of Frame (SOF) timestamp.

inline int64_t timestampEndOfFrame () const
    End of Frame (EOF) timestamp.

inline int64_t timestampStartOfExposure () const
    Start of Exposure (SOE) timestamp.

inline int64_t timestampEndOfExposure () const
    End of Exposure (EOE) timestamp.

```

```
inline FrameFormat format () const
    Pixel format (grayscale, RGB, ...).

inline int16_t sizeX () const
    X axis length in pixels.

inline int16_t sizeY () const
    Y axis length in pixels.

inline int16_t positionX () const
    X axis position (upper left offset) in pixels.

inline int16_t positionY () const
    Y axis position (upper left offset) in pixels.

inline const flatbuffers::Vector<uint8_t> *pixels () const
    Pixel values, 8bit depth.

inline int64_t exposure () const
    Exposure duration.

inline FrameSource source () const
    Source of the image data, whether it's from sensor or from some form of event accumulation.

inline bool Verify (flatbuffers::Verifier &verifier) const

inline Frame *UnPack (const flatbuffers::resolver_function_t *_resolver = nullptr) const

inline void UnPackTo (Frame *_o, const flatbuffers::resolver_function_t *_resolver = nullptr) const
```

Public Static Functions

```
static inline const flatbuffers::TypeTable *MiniReflectTypeTable ()

static inline constexpr const char *GetFullyQualifiedName ()

static inline void UnPackToFrom (Frame *_o, const FrameFlatbuffer *_fb, const flatbuffers::resolver_function_t
    *_resolver = nullptr)

static inline flatbuffers::Offset<FrameFlatbuffer> Pack (flatbuffers::FlatBufferBuilder &_fbb, const Frame *_o,
    const flatbuffers::rehasher_function_t *_rehasher =
    nullptr)
```

Public Static Attributes

```
static constexpr const char *identifier = "FRME"
```

```
struct Gaussian
```

Public Static Functions

```
static inline float getSearchRadius (const float bandwidth)

static inline float apply (const float squaredDistance, const float bandwidth)
```

```
template<dv::concepts::EventStorage EventStoreClass = dv::EventStore>
```

class **HighPassFilter** : public *dv::noise::BaseFrequencyFilter*<*dv::EventStore*>

#include </builds/inivation/dv/dv-processing/include/dv-processing/noise/frequency_filters.hpp> A high-pass event frequency filter. Discards events at a pixel location with a frequency below a given cutoff frequency.

Template Parameters

EventStoreClass – Type of event store.

Public Functions

inline explicit **HighPassFilter** (const cv::Size &resolution, const float cutOffFrequency)

A high-pass event frequency filter. Discards events at a pixel location with a frequency below a given cutoff frequency.

Parameters

- **resolution** – Sensor resolution.
- **cutOffFrequency** – Filter cutoff frequency. All events with a frequency below this given cutoff are discarded.

inline *HighPassFilter* &operator<< (const *EventStoreClass* &events)

Accept events using the input stream operator.

Parameters

events – Input events.

Returns

inline float **getCutOffFrequency** () const

Get the cutoff frequency for the frequency filter.

Returns

Currently configured cutoff frequency.

inline void **setCutOffFrequency** (const float frequency)

Set a new cutoff frequency for the frequency filter.

Parameters

frequency – New cutoff frequency value.

class **ImageFeatureLKTracker** : public *dv::features::TrackerBase*

#include </builds/inivation/dv/dv-processing/include/dv-processing/features/image_feature_lk_tracker.hpp> A feature based sparse Lucas-Kanade feature tracker based on image pyramids.

Subclassed by *dv::features::EventCombinedLKTracker*<*AccumulatorType*>, *dv::features::EventFeatureLKTracker*<*AccumulatorType*>

Public Types

using **Config** = *LucasKanadeConfig*

using **SharedPtr** = *std::shared_ptr*<*ImageFeatureLKTracker*>

using **UniquePtr** = *std::unique_ptr*<*ImageFeatureLKTracker*>

Public Functions

inline virtual void **accept** (const *dv::Frame* &image)

Add an input image for the tracker. Image pyramid will be built from the given image.

Parameters

image – Acquired image.

inline void **setRedetectionStrategy** (*RedetectionStrategy::UniquePtr* redetectionStrategy)

Set a new redetection strategy.

Parameters

redetectionStrategy – Redetection strategy instance.

inline void **setDetector** (*ImagePyrFeatureDetector::UniquePtr* detector)

Set a new feature (corner) detector. If a `nullptr` is passed, the function will instantiate a feature detector with no parameters (defaults).

Parameters

detector – Feature detector instance.

inline void **setMotionPredictor** (*kinematics::PixelMotionPredictor::UniquePtr* predictor)

Set new pixel motion predictor instance. If a `nullptr` is passed, the function will instantiate a pixel motion predictor with no parameters (defaults).

Warning: motion prediction requires camera calibration to be set, otherwise the function will not instantiate the motion predictor.

Parameters

predictor – Pixel motion predictor instance.

inline virtual void **accept** (const *dv::measurements::Depth* &timedDepth)

Add scene depth, a median depth value of tracked landmarks usually works well enough.

Parameters

timedDepth – Depth measurement value (pair of timestamp and measured depth)

inline virtual void **accept** (const *dv::kinematics::Transformationf* &transform)

Add camera transformation, usually in the world coordinate frame (`T_WC`). Although the class only extract the motion difference, so any other reference frame should also work as long as reference frames are not mixed up.

Parameters

transform – Camera pose represented by a transformation

inline bool **isLookbackRejectionEnabled** () const

Check whether lookback is enabled.

Returns

True if lookback rejection is enabled.

inline void **setLookbackRejection** (const bool lookbackRejection)

Enable or disable lookback rejection based on Forward-Backward error. Lookback rejection applies Lucas-Kanade tracking backwards after running the usual tracking and rejects any tracks that fails to successfully track back to same approximate location by measuring Euclidean distance. Euclidean distance threshold for rejection can be set using `setRejectionDistanceThreshold` method.

This is a real-time implementation of the method proposed by Zdenek et al. 2010, that only performs forward-backward error measurement within a single pair of latest and previous frame: http://kahlan.eecs.surrey.ac.uk/featurespace/tld/Publications/2010_icpr.pdf

 See also[*setRejectionDistanceThreshold*](#)**Parameters**

lookbackRejection – Pass true to enable lookback rejection based on Forward-Backward error.

inline float **getRejectionDistanceThreshold**() const

Get the current rejection distance threshold for the lookback rejection feature.

 See also[*setRejectionDistanceThreshold*](#)**Returns**

Rejection distance value which represents the Euclidean distance in pixel space between backward tracked feature pose and initial feature position before performing forward tracking.

inline void **setRejectionDistanceThreshold**(const float rejectionDistanceThreshold)

Set the threshold for lookback rejection feature. This value is a maximum Euclidean distance value that is considered successful when performing backwards tracking check after forward tracking. If the backward tracked feature location is further away from initial position than this given value, the tracker will reject the track as a failed track. See method `setLookbackRejection` documentation for further explanation of the approach.

 See also[*setLookbackRejection*](#)**Parameters**

rejectionDistanceThreshold – Rejection distance threshold value.

inline float **getConstantDepth**() const

Get currently assumed constant depth value. It is used if no depth measurements are provided.

 See also[*setConstantDepth*](#)**Returns**

Currently used aistance to the scene (depth).

inline virtual void **setConstantDepth**(const float depth)

Set constant depth value that is assumed if no depth measurement is passed using `accept(dv::measurements::Depth)`. By default the constant depth is assumed to be 3.0 meters, which is just a reasonable guess.

This value is used for predicting feature track positions when no depth measurements are passed in.

Parameters

depth – Distance to the scene (depth).

Throws

InvalidArgument – Exception is thrown if a negative depth value is passed.

Public Static Functions

```
static inline ImageFeatureLKTracker::UniquePtr RegularTracker (const cv::Size &resolution, const Config
&_config = Config(),
ImagePyrFeatureDetector::UniquePtr
detector = nullptr,
RedetectionStrategy::UniquePtr redetection
= nullptr)
```

```
static inline ImageFeatureLKTracker::UniquePtr MotionAwareTracker (const
camera::CameraGeometry::SharedPtr
&camera, const Config &config =
Config(), kinemat-
ics::PixelMotionPredictor::UniquePtr
motionPredictor = nullptr,
ImagePyrFeatureDetector::UniquePtr
detector = nullptr,
RedetectionStrategy::UniquePtr
redetection = nullptr)
```

Protected Functions

```
inline std::vector<cv::Point2f> predictNextPoints (const int64_t previousTime, const
std::vector<cv::Point2f> &previousPoints, const int64_t
nextTime)
```

```
inline virtual Result::SharedPtr track () override
```

Perform the LK tracking.

Returns

Result of the tracking.

```
inline ImageFeatureLKTracker (const cv::Size &resolution, const Config &config)
```

Construct a tracker with default detector parameters, but configurable tracker parameters.

Parameters

- **resolution** – Image resolution.
- **_config** – Lucas-Kanade tracker parameters.

```
inline ImageFeatureLKTracker (const camera::CameraGeometry::SharedPtr &cameraGeometry, const Config
&config)
```

Construct a tracker with default detector parameters, but configurable tracker parameters.

Parameters

- **resolution** – Image resolution.
- **_config** – Lucas-Kanade tracker parameters.

Protected Attributes

```

Config mConfig = {}

RedetectionStrategy::UniquePtr mRedetectionStrategy = nullptr

ImagePyrFeatureDetector::UniquePtr mDetector = nullptr

cv::Ptr<cv::SparsePyrLKOpticalFlow> mTracker

ImagePyramid::UniquePtr mPreviousFrame = nullptr

ImagePyramid::UniquePtr mCurrentFrame = nullptr

kinematics::PixelMotionPredictor::UniquePtr mPredictor = nullptr

std::unique_ptr<kinematics::LinearTransformerf> mTransformer = nullptr

std::map<int64_t, float> mDepthHistory

camera::CameraGeometry::SharedPtr mCamera = nullptr

cv::Size mResolution

bool mLookbackRejection = false

float mRejectionDistanceThreshold = 10.f

const int64_t depthHistoryDuration = 5000000

float constantDepth = 3.f

```

class **ImagePyramid**

#include </builds/inivation/dv/dv-processing/include/dv-processing/features/image_pyramid.hpp> Class that holds image pyramid layers with an according timestamp.

Public Types

```

typedef std::shared_ptr<ImagePyramid> SharedPtr

typedef std::unique_ptr<ImagePyramid> UniquePtr

```

Public Functions

inline **ImagePyramid** (int64_t timestamp_, const cv::Mat &image, const cv::Size &winSize, int maxPyrLevel)

Construct the image pyramid.

Parameters

- **timestamp_** – Image timestamp.
- **image** – Image values.
- **winSize** – Window size for the search.
- **maxPyrLevel** – Maximum pyramid layer id (zero-based).

inline **ImagePyramid** (const *dv::Frame* &frame, const cv::Size &winSize, int maxPyrLevel)

Construct the image pyramid.

Parameters

- **frame** – *dv::Frame* containing an image and timestamp.
- **winSize** – Window size for the search.
- **maxPyrLevel** – Maximum pyramid layer id (zero-based).

inline **ImagePyramid** (int64_t timestamp_, const cv::Mat &image)

Create a single layer image representation (no pyramid is going to be built).

Parameters

- **timestamp_** – Image timestamp.
- **image** – Image values.

Public Members

int64_t **timestamp**

Timestamp of the image pyramid.

std::vector<cv::Mat> **pyramid**

Pyramid layers of the image.

struct **IMU** : public *flatbuffers::NativeTable*

Public Types

typedef *IMUFlatbuffer* **TableType**

Public Functions

inline **IMU** ()

inline **IMU** (int64_t _timestamp, float _temperature, float _accelerometerX, float _accelerometerY, float _accelerometerZ, float _gyroscopeX, float _gyroscopeY, float _gyroscopeZ, float _magnetometerX, float _magnetometerY, float _magnetometerZ)

```
inline Eigen::Vector3f getAccelerations () const
```

Get measured acceleration in m/s^2.

Returns

Measured acceleration.

```
inline Eigen::Vector3f getAngularVelocities () const
```

Get measured angular velocities in rad/s.

Returns

Measured angular velocities.

Public Members

int64_t **timestamp**

float **temperature**

float **accelerometerX**

float **accelerometerY**

float **accelerometerZ**

float **gyroscopeX**

float **gyroscopeY**

float **gyroscopeZ**

float **magnetometerX**

float **magnetometerY**

float **magnetometerZ**

Public Static Functions

```
static inline constexpr const char *GetFullyQualifiedName ()
```

```
struct IMUBuilder
```

Public Functions

```
inline void add_timestamp (int64_t timestamp)
```

```
inline void add_temperature (float temperature)
```

```
inline void add_accelerometerX (float accelerometerX)

inline void add_accelerometerY (float accelerometerY)

inline void add_accelerometerZ (float accelerometerZ)

inline void add_gyroscopeX (float gyroscopeX)

inline void add_gyroscopeY (float gyroscopeY)

inline void add_gyroscopeZ (float gyroscopeZ)

inline void add_magnetometerX (float magnetometerX)

inline void add_magnetometerY (float magnetometerY)

inline void add_magnetometerZ (float magnetometerZ)

inline explicit IMUBuilder (flatbuffers::FlatBufferBuilder &_fbb)

IMUBuilder &operator= (const IMUBuilder&)

inline flatbuffers::Offset<IMUFlatbuffer> Finish ()
```

Public Members

flatbuffers::FlatBufferBuilder &**fbb_**

flatbuffers::uoffset_t **start_**

struct **IMUCalibration**

Public Functions

IMUCalibration () = default

```
inline explicit IMUCalibration (const std::string_view name_, const float omegaMax_, const float accMax_,
                                const cv::Point3f &omegaOffsetAvg_, const cv::Point3f &accOffsetAvg_,
                                const float omegaOffsetVar_, const float accOffsetVar_, const float
                                omegaNoiseDensity_, const float accNoiseDensity_, const float
                                omegaNoiseRandomWalk_, const float accNoiseRandomWalk_, const int64_t
                                timeOffsetMicros_, const dv::kinematics::Transformationf
                                &transformationToC0_, const std::optional<Metadata> &metadata_)
```

```
inline explicit IMUCalibration (const boost::property_tree::ptree &tree)
```

```
inline boost::property_tree::ptree toPropertyTree () const
```

```
inline bool operator== (const IMUCalibration &rhs) const
```

Public Members

std::string **name**

Sensor name (e.g. "IMU_DVXplorer_DXA02137")

float **omegaMax** = -1.f

Maximum (saturation) angular velocity of the gyroscope [rad/s].

float **accMax** = -1.f

Maximum (saturation) acceleration of the accelerometer [m/s²].

cv::Point3f **omegaOffsetAvg**

Average offset (bias) of the angular velocity [rad/s].

cv::Point3f **accOffsetAvg**

Average offset (bias) of the acceleration [m/s²].

float **omegaOffsetVar** = -1.f

Variance of the offset of the angular velocity [rad/s].

float **accOffsetVar** = -1.f

Variance of the offset of the acceleration [m/s²].

float **omegaNoiseDensity** = -1.f

Noise density of the gyroscope [rad/s/sqrt(Hz)].

float **accNoiseDensity** = -1.f

Noise density of the accelerometer [m/s²/sqrt(Hz)].

float **omegaNoiseRandomWalk** = -1.f

Noise random walk of the gyroscope [rad/s/sqrt(Hz)].

float **accNoiseRandomWalk** = -1.f

Noise random walk of the accelerometer [m/s²/sqrt(Hz)].

int64_t **timeOffsetMicros** = -1

Offset between the camera and *IMU* timestamps in microseconds ($t_{\text{correct}} = t_{\text{imu}} - \text{offset}$)

dv::kinematics::Transformationf **transformationToC0**

Transformation converting points in *IMU* frame to C0 frame $p_{\text{C0}} = T * p_{\text{IMU}}$.

std::optional<Metadata> **metadata**

Metadata.

Friends

```
inline friend std::ostream &operator<< (std::ostream &os, const IMUCalibration &calibration)
```

```
struct IMUFlatbuffer : private flatbuffers::Table
```

Public Types

```
typedef IMU NativeTableType
```

Public Functions

```
inline int64_t timestamp() const
```

Timestamp (μs).

```
inline float temperature() const
```

Temperature, measured in °C.

```
inline float accelerometerX() const
```

Acceleration in the X axis, measured in g (9.81m/s²).

```
inline float accelerometerY() const
```

Acceleration in the Y axis, measured in g (9.81m/s²).

```
inline float accelerometerZ() const
```

Acceleration in the Z axis, measured in g (9.81m/s²).

```
inline float gyroscopeX() const
```

Rotation in the X axis, measured in °/s.

```
inline float gyroscopeY() const
```

Rotation in the Y axis, measured in °/s.

```
inline float gyroscopeZ() const
```

Rotation in the Z axis, measured in °/s.

```
inline float magnetometerX() const
```

Magnetometer X axis, measured in μT (magnetic flux density).

```
inline float magnetometerY() const
```

Magnetometer Y axis, measured in μT (magnetic flux density).

```
inline float magnetometerZ() const
```

Magnetometer Z axis, measured in μT (magnetic flux density).

```
inline bool Verify(flatbuffers::Verifier &verifier) const
```

```
inline IMU *UnPack(const flatbuffers::resolver_function_t *_resolver = nullptr) const
```

```
inline void UnPackTo(IMU *_o, const flatbuffers::resolver_function_t *_resolver = nullptr) const
```

Public Static Functions

```
static inline const flatbuffers::TypeTable *MiniReflectTypeTable()
```

```
static inline constexpr const char *GetFullyQualifiedName()
```

```
static inline void UnPackToFrom (IMU *_o, const IMUFlatbuffer *_fb, const flatbuffers::resolver_function_t
*_resolver = nullptr)
```

```
static inline flatbuffers::Offset<IMUFlatbuffer> Pack (flatbuffers::FlatBufferBuilder &_fbb, const IMU *_o, const
flatbuffers::rehasher_function_t *_rehasher = nullptr)
```

```
struct IMUPacket : public flatbuffers::NativeTable
```

Public Types

```
typedef IMUPacketFlatbuffer TableType
```

Public Functions

```
inline IMUPacket ()
```

```
inline IMUPacket (const std::vector<IMU> &_elements)
```

Public Members

```
std::vector<IMU> elements
```

Public Static Functions

```
static inline constexpr const char *GetFullyQualifiedName ()
```

Friends

```
inline friend std::ostream &operator<< (std::ostream &os, const IMUPacket &packet)
```

```
struct IMUPacketBuilder
```

Public Functions

```
inline void add_elements (flatbuffers::Offset<flatbuffers::Vector<flatbuffers::Offset<IMUFlatbuffer>>>
elements)
```

```
inline explicit IMUPacketBuilder (flatbuffers::FlatBufferBuilder &_fbb)
```

```
IMUPacketBuilder &operator= (const IMUPacketBuilder &)
```

```
inline flatbuffers::Offset<IMUPacketFlatbuffer> Finish ()
```

Public Members

```
flatbuffers::FlatBufferBuilder &fbb_
```

```
flatbuffers::uoffset_t start_
```

```
struct IMUPacketFlatbuffer : private flatbuffers::Table
```

Public Types

```
typedef IMUPacket NativeTableType
```

Public Functions

```
inline const flatbuffers::Vector<flatbuffers::Offset<IMUFlatbuffer>> *elements () const
```

```
inline bool Verify (flatbuffers::Verifier &verifier) const
```

```
inline IMUPacket *UnPack (const flatbuffers::resolver_function_t *_resolver = nullptr) const
```

```
inline void UnPackTo (IMUPacket *_o, const flatbuffers::resolver_function_t *_resolver = nullptr) const
```

Public Static Functions

```
static inline const flatbuffers::TypeTable *MiniReflectTypeTable ()
```

```
static inline constexpr const char *GetFullyQualifiedName ()
```

```
static inline void UnPackToFrom (IMUPacket *_o, const IMUPacketFlatbuffer *_fb, const  
                                flatbuffers::resolver_function_t *_resolver = nullptr)
```

```
static inline flatbuffers::Offset<IMUPacketFlatbuffer> Pack (flatbuffers::FlatBufferBuilder &_fbb, const  
                                                         IMUPacket *_o, const  
                                                         flatbuffers::rehasher_function_t *_rehasher =  
                                                         nullptr)
```

Public Static Attributes

```
static constexpr const char *identifier = "IMUS"
```

```
struct Info
```

Public Members

```
bool imageCompensated = false
```

```
bool depthAvailable = false
```

```
bool transformsAvailable = false
```

```
int64_t depthTime = -1LL
```

```
int64_t generationTime = -1LL
```

```
size_t inputEventCount = 0ULL
```

```
size_t accumulatedEventCount = 0ULL
```

class **InputBase**

#include </builds/inivation/dv/dv-processing/include/dv-processing/io/input_base.hpp> Camera input base class to abstract live camera and recorded files with a common interface.

Subclassed by *dv::io::MonoCameraRecording*, *dv::io::NetworkReader*, *dv::io::camera::CameraInputBase*

Public Functions

virtual **~InputBase** () = default

virtual *std::optional<dv::EventStore>* **getNextEventBatch** () = 0

Parse and retrieve next event batch.

Returns

Event batch or *std::nullopt* if no events were received since last read.

virtual *std::optional<dv::Frame>* **getNextFrame** () = 0

Parse and retrieve next frame.

Returns

Frame or *std::nullopt* if no frames were received since last read.

virtual *std::optional<std::vector<dv::IMU>>* **getNextImuBatch** () = 0

Parse and retrieve next *IMU* data batch.

Returns

IMU data batch or *std::nullopt* if no *IMU* data was received since last read.

virtual *std::optional<std::vector<dv::Trigger>>* **getNextTriggerBatch** () = 0

Parse and retrieve next trigger data batch.

Returns

Trigger data batch or *std::nullopt* if no triggers were received since last read.

virtual *std::optional<cv::Size>* **getEventResolution** () const = 0

Get event stream resolution.

Returns

Event stream resolution, *std::nullopt* if event stream is unavailable.

virtual *std::optional<cv::Size>* **getFrameResolution** () const = 0

Retrieve frame stream resolution.

Returns

Frame stream resolution or *std::nullopt* if the frame stream is not available.

virtual bool **isEventStreamAvailable** () const = 0

Check whether event stream is available.

Returns

True if event stream is available, false otherwise.

virtual bool **isFrameStreamAvailable** () const = 0

Check whether frame stream is available.

Returns

True if frame stream is available, false otherwise.

virtual bool **isImuStreamAvailable** () const = 0

Check whether *IMU* data is available.

Returns

True if *IMU* data stream is available, false otherwise.

virtual bool **isTriggerStreamAvailable** () const = 0

Check whether trigger data is available.

Returns

True if trigger data stream is available, false otherwise.

virtual bool **isStreamAvailable** (std::string_view streamName) const = 0

Check whether a stream with given name is available.

Returns

True if data stream is available, false otherwise.

virtual std::string **getCameraName** () const = 0

Get camera name, which is a combination of the camera model and the serial number.

Returns

String containing the camera model and serial number separated by an underscore character.

virtual bool **isRunning** () const = 0

Check whether any input data streams have terminated. For a live camera this should check if the device is still connected and functioning, while for a recording file this should check if any of the data streams have reached end-of-file (EOF). For a network input, this indicates the network stream is still connected.

Returns

True if data read on all streams is still possible, false otherwise.

virtual bool **isRunning** (std::string_view streamName) const = 0

Check whether the input data stream with the specified name is still active.

Returns

True if data read on this stream is possible, false otherwise.

virtual bool **isRunningAny** () const = 0

Check whether any input data streams are still available. For a live camera this should check if the device is still connected and functioning and at least one data stream is active (different than *isRunning()*), while for a recording file this should check if any of the data streams have not yet reached end-of-file (EOF) and are still readable. For a network input, this indicates the network stream is still connected.

Returns

True if data read on at least one stream is still possible, false otherwise.

struct **InputError**

Public Types

using **Info** = *ErrorInfo*

Public Static Functions

```
static inline std::string format (const Info &info)
```

```
template<class TYPE>
```

```
struct InvalidArgument
```

Public Types

```
using Info = TYPE
```

Public Static Functions

```
static inline std::string format (const Info &info)
```

```
class IODataBuffer
```

Public Functions

```
IODataBuffer () = default
```

```
inline dv::PacketHeader *getHeader ()
```

```
inline const dv::PacketHeader *getHeader () const
```

```
inline flatbuffers::FlatBufferBuilder *getBuilder ()
```

```
inline std::vector<std::byte> *getBuffer ()
```

```
inline const std::byte *getData () const
```

```
inline size_t getDataSize () const
```

```
inline void switchToBuffer ()
```

Private Members

```
dv::PacketHeader mHeader
```

```
std::vector<std::byte> mBuffer
```

```
flatbuffers::FlatBufferBuilder mBuilder = {INITIAL_SIZE}
```

```
bool mIsFlatBuffer = {true}
```

Private Static Attributes

```
static constexpr size_t INITIAL_SIZE = {64 * 1024}
```

```
struct IOError : public dv::exceptions::info::EmptyException
```

```
struct IOHeader : public flatbuffers::NativeTable
```

Public Types

```
typedef IOHeaderFlatbuffer TableType
```

Public Functions

```
inline IOHeader ()
```

```
inline IOHeader (CompressionType _compression, int64_t _dataTablePosition, const std::string &_infoNode)
```

Public Members

```
CompressionType compression
```

```
int64_t dataTablePosition
```

```
std::string infoNode
```

Public Static Functions

```
static inline constexpr const char *GetFullyQualifiedName ()
```

```
struct IOHeaderBuilder
```

Public Functions

```
inline void add_compression (CompressionType compression)
```

```
inline void add_dataTablePosition (int64_t dataTablePosition)
```

```
inline void add_infoNode (flatbuffers::Offset<flatbuffers::String> infoNode)
```

```
inline explicit IOHeaderBuilder (flatbuffers::FlatBufferBuilder &_fbb)
```

```
IOHeaderBuilder &operator= (const IOHeaderBuilder &)
```

```
inline flatbuffers::Offset<IOHeaderFlatbuffer> Finish ()
```

Public Members

```
flatbuffers::FlatBufferBuilder &fbb_
```

```
flatbuffers::uoffset_t start_
```

```
struct IOHeaderFlatbuffer : private flatbuffers::Table
```

Public Types

```
typedef IOHeader NativeTableType
```

Public Functions

```

inline CompressionType compression() const

inline int64_t dataTablePosition() const

inline const flatbuffers::String *infoNode() const

inline bool verify(flatbuffers::Verifier &verifier) const

inline IOHeader *UnPack(const flatbuffers::resolver_function_t *_resolver = nullptr) const

inline void UnPackTo(IOHeader *_o, const flatbuffers::resolver_function_t *_resolver = nullptr) const

```

Public Static Functions

```

static inline const flatbuffers::TypeTable *MiniReflectTypeTable()

static inline constexpr const char *GetFullyQualifiedName()

static inline void UnPackToFrom(IOHeader *_o, const IOHeaderFlatbuffer *_fb, const
                                flatbuffers::resolver_function_t *_resolver = nullptr)

static inline flatbuffers::Offset<IOHeaderFlatbuffer> Pack(flatbuffers::FlatBufferBuilder &_fbb, const IOHeader
                                                         *_o, const flatbuffers::rehasher_function_t *_rehasher
                                                         = nullptr)

```

Public Static Attributes

```

static constexpr const char *identifier = "IOHE"

```

```

class IOStatistics

```

Public Functions

```

IOStatistics() = default

virtual ~IOStatistics() = default

IOStatistics(const IOStatistics &other) = delete

IOStatistics &operator=(const IOStatistics &other) = delete

IOStatistics(IOStatistics &&other) noexcept = default

IOStatistics &operator=(IOStatistics &&other) = default

virtual void publish() = 0

inline void addBytes(const uint64_t bytes)

inline void update(const uint64_t addedDataSize, const uint64_t addedPacketsNumber, const uint64_t
                  addedPacketsElements, const uint64_t addedPacketsSize)

```

Protected Attributes

```
uint64_t mPacketsNumber = {0}
```

```
uint64_t mPacketsElements = {0}
```

```
uint64_t mPacketsSize = {0}
```

```
uint64_t mDataSize = {0}
```

```
template<typename T>
```

```
struct is_eigen_impl : public std::false_type
```

```
template<typename T, int... Is>
```

```
struct is_eigen_impl<Eigen::Matrix<T, Is...>> : public std::true_type
```

```
class KDTreeEventStoreAdaptor
```

```
#include </builds/inivation/dv/dv-processing/include/dv-processing/containers/kd_tree/event_store_adaptor.hpp>
```

Wrapper class around nanoflann::KDTree for dv::EventStore data, which provides efficient approximate nearest neighbour search as well as radius search.

Public Functions

```
inline KDTreeEventStoreAdaptor (const dv::EventStore &data, const uint32_t maxLeaves = 32768)
```

Constructor

See also

MeanShift::Matrix

Template Parameters

T – The matrix type. Must be of exact same type as MeanShift::Matrix, to avoid copy construction of a temporary variable and thereby creating dangling references.

Parameters

- **data** – The Matrix containing the data. The data is neither copied nor otherwise managed, ownership remains with the user of this class.
- **maxLeaves** – the maximum number of leaves for the KDTree. A smaller number typically increases the time used for construction of the tree, but may decrease the time used for searching it. A higher number typically does the opposite.

```
KDTreeEventStoreAdaptor () = delete
```

```
KDTreeEventStoreAdaptor (const KDTreeEventStoreAdaptor &other) = delete
```

```
KDTreeEventStoreAdaptor (KDTreeEventStoreAdaptor &&other) = delete
```

```
KDTreeEventStoreAdaptor &operator= (const KDTreeEventStoreAdaptor &other) = delete
```

KDTreeEventStoreAdaptor &operator= (*KDTreeEventStoreAdaptor* &&other) = delete

~*KDTreeEventStoreAdaptor* () = default

```
template<class T> inline or dv::concepts::Key-
Point< T > auto knnSearch (const T &centrePoint, const size_t numClosest) const
```

Searches for the k nearest neighbours surrounding centrePoint.

Parameters

- **centrePoint** – The point for which the nearest neighbours are to be searched
- **numClosest** – The number of neighbours to be searched (i.e. the parameter “k”)

Returns

The number of actually found neighbours

```
inline std::vector<std::pair<const dv::Event*, int32_t>> knnSearch (const int32_t x, const int32_t y, const
                                                                    size_t numClosest) const
```

Searches for the k nearest neighbours surrounding centrePoint.

Parameters

- **x** – The x-coordinate of the centre point for which the nearest neighbours are to be searched
- **y** – The y-coordinate of the centre point for which the nearest neighbours are to be searched
- **numClosest** – The number of neighbours to be searched (i.e. the parameter “k”)

Returns

The number of actually found neighbours

```
template<class T> inline or dv::concepts::KeyPoint< T > auto ra-
diusSearch (const T &centrePoint, const int16_t &radius, const float eps=0.0f,
const bool sorted=false) const
```

Searches for all neighbours surrounding centrePoint that are within a certain radius.

Parameters

- **centrePoint** – The point for which the nearest neighbours are to be searched
- **radius** – The radius
- **eps** – The search accuracy
- **sorted** – True if the neighbours should be sorted with respect to their distance to centrePoint (comes with a significant performance impact)

Returns

The number of actually found neighbours

```
inline std::vector<std::pair<const dv::Event*, int32_t>> radiusSearch (const int32_t x, const int32_t y, const
                                                                    int16_t &radius, const float eps = 0.0f,
                                                                    const bool sorted = false) const
```

Searches for all neighbours surrounding centrePoint that are within a certain radius.

Parameters

- **x** – The x-coordinate of the centre point for which the nearest neighbours are to be searched
- **y** – The y-coordinate of the centre point for which the nearest neighbours are to be searched
- **radius** – The radius

- **eps** – The search accuracy
- **sorted** – True if the neighbours should be sorted with respect to their distance to centrePoint (comes with a significant performance impact)

Returns

The number of actually found neighbours

```
inline dv::EventStore::iterator begin () const noexcept
```

Returns an iterator to the begin of the EventStore

Returns

an iterator to the begin of the EventStore

```
inline dv::EventStore::iterator end () const noexcept
```

Returns an iterator to the end of the EventStore

Returns

an iterator to the end of the EventStore

```
inline const KDTreeEventStoreAdaptor &derived () const
```

Returns the reference to the this object. Required by the nanoflann adaptors

Returns

the reference to “this”

```
inline KDTreeEventStoreAdaptor &derived ()
```

Returns the reference to the this object. Required by the nanoflann adaptors

Returns

the reference to “this”

```
inline uint32_t kdtree_get_point_count () const
```

Returns the point count of the event store. Required by the nanoflann adaptors

Returns

the reference to “this”

```
inline int16_t kdtree_get_pt (const dv::Event *event, const size_t dim) const
```

Returns the dim'th dimension of an event. Required by the nanoflann adaptors

Returns

the reference to “this”

```
template<class BBOX>
```

```
inline bool kdtree_get_bbox (BBOX&) const
```

Bounding box computation required by the nanoflann adaptors As the documentation allows for it not being implemented and we don't need it, it was left empty.

Returns

false

Private Types

```
using Index =
```

```
nanoflann::KDTreeSingleIndexNonContiguousIteratorAdaptor<nanoflann::metric_L2_Simple::traits<int32_t,  
KDTreeEventStoreAdaptor, const dv::Event*>::distance_t, KDTreeEventStoreAdaptor, 2, const dv::Event*>
```

Private Members

const *dv::EventStore* &**mData**

std::unique_ptr<Index> **mIndex**

template<typename **TYPE**, int32_t **ROWS** = Eigen::Dynamic, int32_t **COLUMNS** = Eigen::Dynamic, int32_t **SAMPLE_ORDER** = Eigen::ColMajor>

class **KDTreeMatrixAdaptor**

#include </builds/inivation/dv/dv-processing/include/dv-processing/containers/kd_tree/eigen_matrix_adaptor.hpp>
Wrapper class around nanoflann::KDTree for data contained in Eigen matrices, which provides efficient approximate nearest neighbour search as well as radius search.

↪ See also

Eigen::Dynamic

↪ See also

Eigen::Dynamic

↪ See also

Eigen::StorageOptions

Template Parameters

- **TYPE** – the underlying data type
- **ROWS** – the number of rows in the data matrix. May be Eigen::Dynamic or >= 0.
- **COLUMNS** – the number of columns in the data matrix. May be Eigen::Dynamic or >= 0.
- **SAMPLE_ORDER** – the order in which samples are entered in the matrix.

Public Types

using **Matrix** = Eigen::Matrix<*TYPE*, *ROWS*, *COLUMNS*, *STORAGE_ORDER*>

using **Vector** = Eigen::Matrix<*TYPE*, *SAMPLE_ORDER* == Eigen::ColMajor ? *ROWS* : 1, *SAMPLE_ORDER* == Eigen::ColMajor ? 1 : *COLUMNS*, *STORAGE_ORDER*>

Public Functions

template<typename **T**, *std::enable_if_t<std::is_same_v<T, Matrix>*, bool> = false>
inline explicit **KDTreeMatrixAdaptor** (const *T* &data, const uint32_t maxLeaves = 32768)

Constructor

 See also

MeanShift::Matrix

Template Parameters

T – The matrix type. Must be of exact same type as MeanShift::Matrix, to avoid copy construction of a temporary variable and thereby creating dangling references.

Parameters

- **data** – The Matrix containing the data. The data is neither copied nor otherwise managed, ownership remains with the user of this class.
- **maxLeaves** – the maximum number of leaves for the KDTree. A smaller number typically increases the time used for construction of the tree, but may decrease the time used for searching it. A higher number typically does the opposite.

KDTreeMatrixAdaptor () = delete

KDTreeMatrixAdaptor (const *ThisType* &other) = delete

KDTreeMatrixAdaptor (*ThisType* &&other) = delete

KDTreeMatrixAdaptor &operator= (const *ThisType* &other) = delete

KDTreeMatrixAdaptor &operator= (*ThisType* &&other) = delete

~**KDTreeMatrixAdaptor** () = default

inline auto **knnSearch** (const *Vector* ¢rePoint, const size_t numClosest) const

Searches for the k nearest neighbours surrounding centrePoint.

Parameters

- **centrePoint** – The point for which the nearest neighbours are to be searched
- **numClosest** – The number of neighbours to be searched (i.e. the parameter “k”)

Returns

A pair containing the indices of the neighbours in the underlying matrix as well as the distances to centrePoint

inline auto **radiusSearch** (const *Vector* ¢rePoint, const *TYPE* &radius, const float eps = 0.0f, const bool sorted = false) const

Searches for all neighbours surrounding centrePoint that are within a certain radius.

Parameters

- **centrePoint** – The point for which the nearest neighbours are to be searched
- **radius** – The radius
- **eps** – The search accuracy
- **sorted** – True if the neighbours should be sorted with respect to their distance to centrePoint (comes with a significant performance impact)

Returns

A vector of pairs containing the indices of the neighbours in the underlying matrix as well as the distances to centrePoint

```
inline auto getSample (const uint32_t index) const
```

Returns a sample at a given index

Parameters

index – the index of the sample in mData

Returns

the sample

Private Types

```
using ThisType = KDTreeMatrixAdaptor<TYPE, ROWS, COLUMNS, SAMPLE_ORDER>
```

```
using Tree = nanoflann::KDTreeEigenMatrixAdaptor<Matrix, SAMPLE_ORDER == Eigen::ColMajor ? ROWS : COLUMNS, nanoflann::metric_L2_Simple, SAMPLE_ORDER == Eigen::RowMajor>
```

Private Members

```
const uint32_t mNumSamples
```

```
const uint32_t mNumDimensions
```

```
std::unique_ptr<Tree> mTree
```

Private Static Attributes

```
static constexpr int32_t DIMS = SAMPLE_ORDER == Eigen::ColMajor ? ROWS : COLUMNS
```

```
static constexpr int32_t NOT_SAMPLE_ORDER = (SAMPLE_ORDER == Eigen::ColMajor ? Eigen::RowMajor : Eigen::ColMajor)
```

```
static constexpr int32_t STORAGE_ORDER = DIMS == 1 ? NOT_SAMPLE_ORDER : SAMPLE_ORDER
```

```
class KeyPointResampler
```

#include </builds/inivation/dv/dv-processing/include/dv-processing/features/keypoint_resampler.hpp> Create a feature resampler, which resamples given keypoints with homogenous distribution in pixel space.

Implementation was inspired by: <https://github.com/BAILLOOL/ANMS-Codes>

Public Functions

```
inline explicit KeyPointResampler (const cv::Size &resolution)
```

Initialize resampler with given resolution.

Parameters

resolution – Image resolution

```
template<class KeyPointVectorType> inline or dv::concepts::Coordinate2DMutableIterable< KeyPointVectorType > KeyPointVectorType resample (const KeyPointVectorType &keyPoints, const size_t numRetPoints)
```

Perform resampling on given keypoints.

 See also

setTolerance)

Parameters

- **keyPoints** – Prior keypoints.
- **numRetPoints** – Number of expected keypoints, the exact number of output keypoints can vary to configured tolerance value (

Returns

Resampled keypoints

inline float **getTolerance** () const

Get currently set tolerance for output keypoint count.

Returns

Tolerance value

inline void **setTolerance** (const float tolerance)

Set a new output size tolerance value.

The algorithm search for an optimal distance between keypoints so the resulting vector would contain the expected amount of keypoints. This search is performed with a given tolerance, by default - 0.1 (so by default the final resampled amount of events will be within +/-10% of requested amount).

Parameters

tolerance – Output keypoint amount tolerance value.

Protected Types

typedef *std*::pair<*dv*::Point2f, size_t> **RangeValue**

Protected Attributes

float **mPreviousSolution** = -1.f

float **mRows**

float **mCols**

float **mTolerance** = 0.1f

struct **KMemCell**

Memory cell used per row/column to store latest event information. Note that, differently from the original implementation, the number of bits per memory cell is not exactly 64 (unnecessary bookkeeping bit removed).

Public Members

int64_t mTimestamp

int16_t mOtherAddress

bool mPolarity

template<dv::concepts::EventStorage EventStoreClass = dv::EventStore>

class **KNoiseFilter** : public dv::EventFilterBase<dv::EventStore>

#include </builds/inivation/dv/dv-processing/include/dv-processing/noise/k_noise_filter.hpp> Memory efficient, spatiotemporal event filter as proposed by Khodamoradi's algorithm; "O(N)-Space Spatiotemporal Filter for Reducing Noise in Neuromorphic Vision Sensors".

Template Parameters

EventStoreClass – Type of event store to filter.

Public Functions

inline explicit **KNoiseFilter** (const cv::Size &resolution, const dv::Duration timeDelta = dv::Duration(2'000))

Construct a spatiotemporal event filter that discards an event if the number of spatiotemporal neighbors is less than #numSupportingPixels. The spatial window size is defined by #windowHalfSize, while the temporal window size is defined by #timeDelta.

Note that this is similar in functionality to the *BackgroundActivityNoiseFilter*, where #timeDelta corresponds to the backgroundActivityTime, and #windowHalfSize and #numSupportingPixels are by default set to 1; but more memory efficient due to only storing values per row/column.

Parameters

- **resolution** – Sensor resolution.
- **timeDelta** – Size of the temporal window used to check the number of neighboring events.

inline *KNoiseFilter* &operator<< (const *EventStoreClass* &events)

Accept events using the input stream operator.

Parameters

events – Input events.

Returns

inline virtual bool **retain** (const typename *EventStoreClass*::value_type &event) noexcept override

Test if the given event should be filtered out by checking if the number of neighboring events within the defined spatiotemporal window is greater than #mNumSupportingPixels.

Parameters

event – Event to be checked.

Returns

True to retain event, false to discard.

inline dv::Duration **getTemporalWindowDuration** () const

Get the duration of the temporal window used to check for neighboring events.

Returns

```
inline void setTemporalWindowDuration (const dv::Duration timeDelta)
```

Set the duration of the temporal window used to check for neighboring events.

Parameters

timeDelta –

Private Functions

```
inline bool checkColumnSupport (const dv::Event event, const int8_t columnShift)
```

Check if for the column given by event.x() + columnShift, the last received event for that column is within a spatiotemporal window from the current event.

Parameters

- **event** – Event to be checked.
- **columnShift** – Shift in the column position from the current event column.

Returns

True if the last received event at the given column position is within a spatiotemporal window from the current event.

```
inline bool checkRowSupport (const dv::Event event, const int8_t rowShift)
```

Check if for the row given by event.y() + rowShift, the last received event for that row is within a spatiotemporal window from the current event.

Parameters

- **event** – Event to be checked.
- **rowShift** – Shift in the row position from the current event row.

Returns

True if the last received event at the given row position is within a spatiotemporal window from the current event.

```
inline bool doKNoiseLookup_unsafe (const dv::Event event)
```

```
inline bool doKNoiseLookup (const dv::Event event)
```

Private Members

```
cv::Size mResolutionLimits
```

Sensor resolution.

```
int64_t mTimeDelta
```

Temporal size of the spatiotemporal window (in us)

```
std::vector<KMemCell> mColumnCells
```

Vector containing the memory cells per row/column of the sensor resolution.

```
std::vector<KMemCell> mRowCells
```

```
struct Landmark : public flatbuffers::NativeTable
```

Public Types

typedef *LandmarkFlatbuffer* **TableType**

Public Functions

inline **Landmark** ()

inline **Landmark** (const Point3f &_pt, int64_t _id, int64_t _timestamp, const *std::vector*<int8_t> &_descriptor, const *std::string* &_descriptorType, const *std::vector*<float> &_covariance, const *std::vector*<*Observation*> &_observations)

Public Members

Point3f **pt**

int64_t **id**

int64_t **timestamp**

std::vector<int8_t> **descriptor**

std::string **descriptorType**

std::vector<float> **covariance**

std::vector<*Observation*> **observations**

Public Static Functions

static inline constexpr const char ***GetFullyQualifiedName** ()

struct **LandmarkBuilder**

Public Functions

inline void **add_pt** (const Point3f *pt)

inline void **add_id** (int64_t id)

inline void **add_timestamp** (int64_t timestamp)

inline void **add_descriptor** (*flatbuffers::Offset*<*flatbuffers::Vector*<int8_t>> descriptor)

inline void **add_descriptorType** (*flatbuffers::Offset*<*flatbuffers::String*> descriptorType)

inline void **add_covariance** (*flatbuffers::Offset*<*flatbuffers::Vector*<float>> covariance)

```
inline void add_observations (flat-  
                           buffers::Offset<flatbuffers::Vector<flatbuffers::Offset<ObservationFlatbuffer>>>  
                           observations)
```

```
inline explicit LandmarkBuilder (flatbuffers::FlatBufferBuilder &_fbb)
```

```
LandmarkBuilder &operator= (const LandmarkBuilder &)
```

```
inline flatbuffers::Offset<LandmarkFlatbuffer> Finish ()
```

Public Members

```
flatbuffers::FlatBufferBuilder &fbb_
```

```
flatbuffers::uoffset_t start_
```

```
struct LandmarkFlatbuffer : private flatbuffers::Table
```

Public Types

```
typedef Landmark NativeTableType
```

Public Functions

```
inline const Point3f *pt () const  
    3D coordinate of the landmark.
```

```
inline int64_t id () const  
    Landmark id (if the keypoints need to be clustered by an object they belong to).
```

```
inline int64_t timestamp () const  
    Timestamp (µs).
```

```
inline const flatbuffers::Vector<int8_t> *descriptor () const  
    Visual descriptor of the landmark.
```

```
inline const flatbuffers::String *descriptorType () const  
    Type of the visual descriptor.
```

```
inline const flatbuffers::Vector<float> *covariance () const  
    Covariance matrix, must contain 9 numbers. It is represented as a 3x3 square matrix.
```

```
inline const flatbuffers::Vector<flatbuffers::Offset<ObservationFlatbuffer>> *observations () const  
    Observation info, can be from multiple cameras if they are matched using descriptor.
```

```
inline bool Verify (flatbuffers::Verifier &verifier) const
```

```
inline Landmark *UnPack (const flatbuffers::resolver_function_t *_resolver = nullptr) const
```

```
inline void UnPackTo (Landmark *_o, const flatbuffers::resolver_function_t *_resolver = nullptr) const
```

Public Static Functions

```
static inline const flatbuffers::TypeTable *MiniReflectTypeTable ()

static inline constexpr const char *GetFullyQualifiedName ()

static inline void UnPackToFrom (Landmark *_o, const LandmarkFlatbuffer *_fb, const
                                flatbuffers::resolver_function_t *_resolver = nullptr)

static inline flatbuffers::Offset<LandmarkFlatbuffer> Pack (flatbuffers::FlatBufferBuilder &_fbb, const
                                                         Landmark *_o, const flatbuffers::rehasher_function_t
                                                         *_rehasher = nullptr)
```

```
struct LandmarksPacket : public flatbuffers::NativeTable
```

Public Types

```
typedef LandmarksPacketFlatbuffer TableType
```

Public Functions

```
inline LandmarksPacket ()

inline LandmarksPacket (const std::vector<Landmark> &_elements, const std::string &_referenceFrame)
```

Public Members

```
std::vector<Landmark> elements
```

```
std::string referenceFrame
```

Public Static Functions

```
static inline constexpr const char *GetFullyQualifiedName ()
```

Friends

```
inline friend std::ostream &operator<< (std::ostream &os, const LandmarksPacket &packet)
```

```
struct LandmarksPacketBuilder
```

Public Functions

```
inline void add_elements (flatbuffers::Offset<flatbuffers::Vector<flatbuffers::Offset<LandmarkFlatbuffer>>>
                          elements)

inline void add_referenceFrame (flatbuffers::Offset<flatbuffers::String> referenceFrame)

inline explicit LandmarksPacketBuilder (flatbuffers::FlatBufferBuilder &_fbb)

LandmarksPacketBuilder &operator= (const LandmarksPacketBuilder &)

inline flatbuffers::Offset<LandmarksPacketFlatbuffer> Finish ()
```

Public Members

flatbuffers::FlatBufferBuilder &**fbb_**

flatbuffers::uoffset_t **start_**

struct **LandmarksPacketFlatbuffer** : private *flatbuffers::Table*

Public Types

typedef *LandmarksPacket* **NativeTableType**

Public Functions

inline const *flatbuffers::Vector*<*flatbuffers::Offset*<*LandmarkFlatbuffer*>> ***elements** () const

inline const *flatbuffers::String* ***referenceFrame** () const

Coordinate reference frame of the landmarks, “world” coordinate frame by default.

inline bool **Verify** (*flatbuffers::Verifier* &verifier) const

inline *LandmarksPacket* ***UnPack** (const *flatbuffers::resolver_function_t* *_resolver = nullptr) const

inline void **UnPackTo** (*LandmarksPacket* *_o, const *flatbuffers::resolver_function_t* *_resolver = nullptr) const

Public Static Functions

static inline const *flatbuffers::TypeTable* ***MiniReflectTypeTable** ()

static inline constexpr const char ***GetFullyQualifiedName** ()

static inline void **UnPackToFrom** (*LandmarksPacket* *_o, const *LandmarksPacketFlatbuffer* *_fb, const *flatbuffers::resolver_function_t* *_resolver = nullptr)

static inline *flatbuffers::Offset*<*LandmarksPacketFlatbuffer*> **Pack** (*flatbuffers::FlatBufferBuilder* &_fbb, const *LandmarksPacket* *_o, const *flatbuffers::rehasher_function_t* *_rehasher = nullptr)

Public Static Attributes

static constexpr const char ***identifier** = "LMRS"

struct **LengthError** : public *dv::exceptions::info::EmptyException*

template<*std::floating_point* **Scalar**>

class **LinearTransformer**

#include </builds/inivation/dv/dv-processing/include/dv-processing/kinematics/linear_transformer.hpp> A buffer containing time increasing 3D transformations and capable of timewise linear interpolation between available transforms. Can be used with different underlying floating point types supported by Eigen.

Template Parameters

Scalar – Underlying floating point number type - float or double.

Public Types

using **iterator** = typename *TransformationBuffer*::iterator

using **const_iterator** = typename *TransformationBuffer*::const_iterator

Public Functions

inline explicit **LinearTransformer** (size_t capacity)

inline void **pushTransformation** (const *TransformationType* &transformation)

Push a transformation into the transformation buffer.

Throws

logic_error – exception when transformation is added out of order.

Parameters

transformation – *Transformation* to be pushed, it must contain increasing timestamp compared to latest transformation in the buffer, otherwise an exception will be thrown.

inline *iterator* **begin** ()

Generate forward iterator pointing to first transformation in the transformer buffer.

Returns

Buffer start iterator.

inline *iterator* **end** ()

Generate an iterator representing end of the buffer.

Returns

Buffer end const-iterator.

inline *const_iterator* **cbegin** () const

Generate a const forward iterator pointing to first transformation in the transformer buffer.

Returns

Buffer start const-iterator.

inline *const_iterator* **cend** () const

Generate a const iterator representing end of the buffer.

Returns

Buffer end iterator.

inline void **clear** ()

Delete all transformations from the buffer.

inline bool **empty** () const

Check whether the buffer is empty.

Returns

true if empty, false otherwise

inline *std::optional*<*TransformationType*> **getTransformAt** (int64_t timestamp) const

Get a transform at the given timestamp.

If no transform with the exact timestamp was pushed, estimates a transform assuming linear motion.

Parameters

timestamp – Unix timestamp in microsecond format.

Returns

Transformation if successful, std::nullopt otherwise.

```
inline bool isWithinTimeRange (int64_t timestamp) const
```

Checks whether the timestamp is within the range of transformations available in the buffer.

Parameters

timestamp – Unix microsecond timestamp to be checked.

Returns

true if the timestamp is within the range of transformations in the buffer.

```
inline size_t size () const
```

Return the size of the buffer.

Returns

Number of transformations available in the buffer.

```
inline const TransformationType &latestTransformation () const
```

Return transformation with highest timestamp.

Returns

Latest transformation in the buffer.

```
inline const TransformationType &earliestTransformation () const
```

Return transformation with lowest timestamp.

Returns

Earliest transformation in time available in the buffer.

```
inline void setCapacity (size_t newCapacity)
```

Set new capacity, if the size of the buffer is larger than the newCapacity, oldest transformations from the start will be removed.

Parameters

newCapacity – New transformation buffer capacity.

```
inline LinearTransformer<Scalar> getTransformsBetween (int64_t start, int64_t end) const
```

Extract transformation between two given timestamps. If timestamps are not at exact available transformations, additional transformations will be added so the resulting transformer would complete overlap over the period (if that is possible).

Parameters

- **start** – Start Unix timestamp in microseconds.
- **end** – End Unix timestamp in microseconds.

Returns

LinearTransformer containing transformations covering the given period.

```
inline LinearTransformer<Scalar> resampleTransforms (const int64_t samplingInterval) const
```

Resample containing transforms into a new transformer, containing interpolated transforms at given interval. Will contain the last transformation as well, although the interval might not be maintained for the last transform.

Parameters

samplingInterval – Interval in microseconds at which to resample the transformations.

Returns

Generated transformer with exact capacity of output transformation count.

Private Types

```
using TransformationType = Transformation<Scalar>
```

```
using TransformationBuffer = boost::circular_buffer<TransformationType,
Eigen::aligned_allocator<TransformationType>>
```

Private Functions

```
inline TransformationBuffer::const_iterator bufferLowerBound (int64_t t) const
```

Finds the lower bound iterator in the buffer.

See also

std::lower_bound

Parameters

t – Unix timestamp in microseconds to search for.

Returns

Iterator to the buffer with timestamp that is *equal or not less* than given timestamp.

```
inline TransformationBuffer::const_iterator bufferUpperBound (int64_t t) const
```

Finds the upper bound iterator in the buffer.

See also

std::upper_bound

Parameters

t – Unix timestamp in microseconds to search for.

Returns

Iterator to the buffer with timestamp that is *greater* than given timestamp or end if not available.

Private Members

TransformationBuffer **mTransforms**

Private Static Functions

```
static inline TransformationType interpolateComponentwise (const TransformationType &T_a, const
TransformationType &T_b, const int64_t
timestamp, Scalar lambda)
```

Perform linear interpolation between two transformations.

Parameters

- **T_a** – First transformation.
- **T_b** – Second transformation.
- **timestamp** – Interpolated transformation timestamp.

- **lambda** – Distance point between the two transformation to interpolate.

Returns

Interpolated transformation.

```
template<dv::concepts::EventStorage EventStoreClass = dv::EventStore>
```

```
class LowPassFilter : public dv::noise::BaseFrequencyFilter<dv::EventStore>
```

#include </builds/inivation/dv/dv-processing/include/dv-processing/noise/frequency_filters.hpp> A low-pass event frequency filter. Discards events at a pixel location with a frequency above a given cutoff frequency. This is also commonly referred to as refractory period filtering, where the refractory period is the inverse of the cutoff frequency.

Template Parameters

EventStoreClass – Type of event store.

Public Functions

```
inline explicit LowPassFilter (const cv::Size &resolution, const float cutOffFrequency)
```

A low-pass event frequency filter. Discards events at a pixel location with a frequency above a given cutoff frequency. This is also commonly referred to as refractory period filtering, where the refractory period is the inverse of #cutOffFrequency.

Parameters

- **resolution** – Sensor resolution.
- **cutOffFrequency** – Filter cutoff frequency. All events with a frequency above this given cutoff are discarded.

```
inline LowPassFilter &operator<< (const EventStoreClass &events)
```

Accept events using the input stream operator.

Parameters

events – Input events.

Returns

```
inline float getCutOffFrequency () const
```

Get the cutoff frequency for the frequency filter.

Returns

Currently configured cutoff frequency.

```
inline void setCutOffFrequency (const float frequency)
```

Set a new cutoff frequency for the frequency filter.

Parameters

frequency – New cutoff frequency value.

```
struct LucasKanadeConfig
```

#include </builds/inivation/dv/dv-processing/include/dv-processing/features/image_feature_lk_tracker.hpp>
Lucas-Kanade tracker configuration parameters.

Public Members

```
bool maskedFeatureDetect = true
```

Generate a mask which would disable image regions where features are already successfully tracked.

double **terminationEpsilon** = 0.1
Tracking termination criteria for the LK tracker.

int **numPyrLayers** = 2
Total number of pyramid layers used by the LK tracker.

cv::Size **searchWindowSize** = cv::Size(24, 24)
Size of the search around the tracked feature.

class **Lz4CompressionSupport** : public *dv::io::compression::CompressionSupport*

Public Functions

inline explicit **Lz4CompressionSupport** (const *CompressionType* type)

inline explicit **Lz4CompressionSupport** (const LZ4F_preferences_t &preferences)

LZ4 compression support with custom compression settings. Internally sets compression type to *CompressionType::LZ4*.

Parameters

preferences – LZ4 compression settings.

inline virtual void **compress** (*dv::io::support::IODataBuffer* &packet) override

Private Members

std::shared_ptr<LZ4F_cctx_s> **mContext**

const LZ4F_preferences_t **mPrefs**

size_t **mChunkSize**

size_t **mEndSize**

Private Static Attributes

static constexpr size_t **LZ4_COMPRESSION_CHUNK_SIZE** = {64 * 1024}

static constexpr LZ4F_preferences_t **lz4CompressionPreferences** = {{LZ4F_max64KB, LZ4F_blockLinked, LZ4F_noContentChecksum, LZ4F_frame}, 0, 0,}

static constexpr LZ4F_preferences_t **lz4HighCompressionPreferences** = {{LZ4F_max64KB, LZ4F_blockLinked, LZ4F_noContentChecksum, LZ4F_frame}, 9, 0,}

class **Lz4DecompressionSupport** : public *dv::io::compression::DecompressionSupport*

Public Functions

inline explicit **Lz4DecompressionSupport** (const *CompressionType* type)

inline virtual void **decompress** (*std::vector<std::byte>* &src, *std::vector<std::byte>* &target) override

Private Functions

inline void **initDecompressionContext** ()

Private Members

std::shared_ptr<LZ4F_dctx_s> **mContext**

Private Static Attributes

static constexpr size_t **LZ4_DECOMPRESSION_CHUNK_SIZE** = {64 * 1024}

class **MapOfVariants** : public *std::unordered_map<std::string, InputType>*

#include </builds/inivation/dv/dv-processing/include/dv-processing/core/multi_stream_slicer.hpp> Class that is passed to the slicer callback. It is an unordered map where key is the configured stream name and the value is a variant. The class provides convenience methods to access and cast the types.

Public Functions

template<class **Type**>

inline *Type* &**get** (const *std::string* &streamName)

Get a reference to the data packet of a given stream name.

Template Parameters

Type – Type of data for the stream.

Parameters

streamName – Stream name.

Returns

Data packet casted to the given type.

template<class **Type**>

inline const *Type* &**get** (const *std::string* &streamName) const

Get a reference to the data packet of a given stream name.

Template Parameters

Type – Type of data for the stream.

Parameters

streamName – Stream name.

Returns

Data packet casted to the given type.

template<typename **TYPE**, int32_t **ROWS** = Eigen::Dynamic, int32_t **COLUMNS** = Eigen::Dynamic, int32_t **SAMPLE_ORDER** = Eigen::ColMajor>

class **MeanShiftEigenMatrixAdaptor**

```
#include </builds/inivation/dv/dv-processing/include/dv-processing/cluster/mean_shift/eigen_matrix_adaptor.hpp>
```

This class implements the Mean Shift clustering algorithm.

As the Mean Shift algorithm performs a gradient ascent on an estimated probability density function, when applying it to integer data, which has a non-smooth probability density, the quality of the detected clusters depends significantly on the selected bandwidth hyperparameter, as well as the underlying data and the selected kernel. Generally the Gaussian Kernel yields better results for this kind of data, however it comes with a bigger performance impact.

The Mean Shift algorithm is an nonparametric estimate of the modes of the underlying probability distribution for the data. It implements an iterative search, starting from points provided by the user, or randomly selected from the data points provided. For each iteration, the current estimate of the mode is replaced by an estimate of the mean value of the surrounding data samples. If the Epanechnikov kernel is used for the underlying density estimate, its so-called “shadow kernel”, the flat kernel must be used for the estimate of the mean. This means, that we can simply compute the average value of the data points that lie within a given radius around the current estimate of the mode, and use this as the next estimate. To provide an efficient search for the neighbours of the current mode estimate, a KD tree was used.

For the underlying theory, see “The Estimation of the Gradient of a Density Function with

Applications in Pattern Recognition” by K. Fukunaga and L. Hostetler as well as “Mean shift, mode seeking, and clustering” by Yizong Cheng.

➡ See also

Eigen::Dynamic

➡ See also

Eigen::Dynamic

➡ See also

Eigen::StorageOptions

Template Parameters

- **TYPE** – the underlying data type
- **ROWS** – the number of rows in the data matrix. May be Eigen::Dynamic or ≥ 0 .
- **COLUMNS** – the number of columns in the data matrix. May be Eigen::Dynamic or ≥ 0 .
- **SAMPLE_ORDER** – the order in which samples are entered in the matrix.

Public Types

using **Matrix** = Eigen::Matrix<TYPE, ROWS, COLUMNS, STORAGE_ORDER>

using **Vector** = Eigen::Matrix<TYPE, SAMPLE_ORDER == Eigen::ColMajor ? ROWS : 1, SAMPLE_ORDER == Eigen::ColMajor ? 1 : COLUMNS, STORAGE_ORDER>

```
using VectorOfVectors = std::vector<Vector, Eigen::aligned_allocator<Vector>>
```

Public Functions

```
template<typename T, std::enable_if_t<std::is_same_v<T, Matrix>, bool> = false>
inline MeanShiftEigenMatrixAdaptor (const T &data, const TYPE bw, TYPE conv, const uint32_t maxIter,
                                     const VectorOfVectors &startingPoints, const uint32_t numLeaves =
                                     32768)
```

Constructor

See also

MeanShift::Matrix

See also

dv::containers::KDTree

Template Parameters

T – The matrix type. Must be of exact same type as MeanShift::Matrix, to avoid copy construction of a temporary variable and thereby creating dangling references.

Parameters

- **data** – The Matrix containing the data. The data is neither copied nor otherwise managed, ownership remains with the user of this class.
- **bw** – The bandwidth used for the shift. This is a hyperparameter for the kernel. For the Epanechnikov kernel this means that all values within a radius of bw are averaged.
- **conv** – For each starting point, the algorithm is stopped as soon as the absolute value of the shift is <= conv.
- **maxIter** – The maximum number of iterations. Detected modes, for which the the number of iterations exceed this value are not added to the detected clusters.
- **startingPoints** – Points from which to start the search.
- **numLeaves** – the maximum number of leaves for the KDTree.

```
template<typename T, std::enable_if_t<std::is_same_v<T, Matrix>, bool> = false>
inline MeanShiftEigenMatrixAdaptor (const T &data, const TYPE bw, TYPE conv, const uint32_t maxIter,
                                     VectorOfVectors &&startingPoints, const uint32_t numLeaves =
                                     32768)
```

Constructor

See also

MeanShift::Matrix

 See also

dv::containers::KDTree

Template Parameters

T – The matrix type. Must be of exact same type as MeanShift::Matrix, to avoid copy construction of a temporary variable and thereby creating dangling references.

Parameters

- **data** – The Matrix containing the data. The data is neither copied nor otherwise managed, ownership remains with the user of this class.
- **bw** – The bandwidth used for the shift. This is a hyperparameter for the kernel. For the Epanechnikov kernel this means that all values within a radius of bw are averaged.
- **conv** – For each starting point, the algorithm is stopped as soon as the absolute value of the shift is $\leq$ conv.
- **maxIter** – The maximum number of iterations. Detected modes, for which the the number of iterations exceed this value are not added to the detected clusters.
- **startingPoints** – Points from which to start the search.
- **numLeaves** – the maximum number of leaves for the KDTree.

```
template<typename T, std::enable_if_t<std::is_same_v<T, Matrix>, bool> = false>
inline MeanShiftEigenMatrixAdaptor (const T &data, const TYPE bw, TYPE conv, const uint32_t maxIter,
                                     const uint32_t numStartingPoints, const uint32_t numLeaves =
                                     32768)
```

Constructor

 See also

MeanShift::Matrix

 See also

dv::containers::KDTree

Template Parameters

T – The matrix type. Must be of exact same type as MeanShift::Matrix, to avoid copy construction of a temporary variable and thereby creating dangling references.

Parameters

- **data** – The Matrix containing the data. The data is neither copied nor otherwise managed, ownership remains with the user of this class.
- **bw** – The bandwidth used for the shift. This is a hyperparameter for the kernel. For the Epanechnikov kernel this means that all values within a radius of bw are averaged.
- **conv** – For each starting point, the algorithm is stopped as soon as the absolute value of the shift is $\leq$ conv.

- **maxIter** – The maximum number of iterations. Detected modes, for which the the number of iterations exceed this value are not added to the detected clusters.
- **numStartingPoints** – The number of points which are randomly selected from the data points, to be used as starting points.
- **numLeaves** – the maximum number of leaves for the KDTree.

MeanShiftEigenMatrixAdaptor () = delete

MeanShiftEigenMatrixAdaptor (const *ThisType* &other) = delete

MeanShiftEigenMatrixAdaptor (*ThisType* &&other) = delete

MeanShiftEigenMatrixAdaptor &operator= (const *ThisType* &other) = delete

MeanShiftEigenMatrixAdaptor &operator= (*ThisType* &&other) = delete

~MeanShiftEigenMatrixAdaptor () = default

template<*kernel::MeanShiftKernel* **kernel** = *kernel::Epanechnikov*>

inline auto **fit** ()

Executes the algorithm.

See also

MeanShiftKernel

Template Parameters

kernel – the kernel to be used.

Returns

The centres of each detected cluster

Returns

The labels for each data point. The labels correspond to the index of the centre to which the sample is assigned.

Returns

The number of samples in each cluster

Returns

The in-cluster variance for each cluster

Public Static Functions

static inline *VectorOfVectors* **generateStartingPointsFromData** (const uint32_t numStartingPoints, const *Matrix* &data)

Generates a vector of vectors containing the starting points by randomly selecting from provided data.

Parameters

- **numStartingPoints** – The number of points to be generated.
- **data** – the matrix to select the starting points from.

Returns

The vector of vectors containing the starting points.

```
static inline VectorOfVectors generateStartingPointsFromRange (const uint32_t numStartingPoints, const
                                                                std::vector<std::pair<TYPE, TYPE>>
                                                                &ranges)
```

Generates a vector of vectors containing the starting points by generating random points within a given range for each dimension

Parameters

- **numStartingPoints** – The number of points to be generated
- **ranges** – a vector containing one range per dimension. Each dimension is represented by a pair containing the beginning and the end of the range

Returns

The vector of vectors containing the starting points.

Private Types

```
using ThisType = MeanShiftEigenMatrixAdaptor<TYPE, ROWS, COLUMNS, SAMPLE_ORDER>
```

```
using KDTree = dv::containers::kd_tree::KDTreeMatrixAdaptor<TYPE, ROWS, COLUMNS, SAMPLE_ORDER>
```

Private Functions

```
template<kernel::MeanShiftKernel kernel>
inline auto findClusterCentres ()
```

Performs the search for the cluster centres for each given starting point. A detected centre is added to the set of centres if it isn't closer than the bandwidth to any previously detected centre.

See also

MeanShiftKernel

Template Parameters

kernel – the kernel to be used.

Returns

The centres of each detected cluster

```
inline auto assignClusters (const VectorOfVectors &clusterCentres)
```

Assigns the data samples to a cluster by means of a nearest neighbour search, and computes the number of samples as well as the in-cluster variance in the process.

Parameters

clusterCentres – The centres of each detected cluster

Returns

The labels for each data point. The labels correspond to the index of the centre to which the sample is assigned.

Returns

The number of samples in each cluster

Returns

The in-cluster variance for each cluster

```
template<kernel::MeanShiftKernel kernel>
inline std::optional<Vector> performShift (Vector currentMode)
```

Performs a search for a mode in the underlying density starting off with a provided initial point.

 See also

MeanShiftKernel

Template Parameters

kernel – the kernel to be used.

Parameters

currentMode – The starting point that is to be shifted until convergence.

Returns

An std::optional containing either a vector, if the search has converged, std::nullopt otherwise

```
template<kernel::MeanShiftKernel kernel>
inline float applyKernel (const float squaredDistance) const
```

Applies the selected kernel to the squared distance

 See also

MeanShiftKernel

Template Parameters

kernel – the kernel to be used.

Parameters

squaredDistance – the squared distance between the current mode estimate and a given sample point

Returns

the kernel value

```
template<kernel::MeanShiftKernel kernel>
inline auto getNeighbours (const Vector &currentMode)
```

Returns the neighbours surrounding a centre

 See also

MeanShiftKernel

Template Parameters

kernel – the kernel to be used.

Parameters

centre – the centre surrounding which the neighbours are to be found

Returns

the neighbours, as a vector of pairs, one pair per neighbour containing a the index of the point in the data matrix and a distance to the centre

inline auto **getSample** (const uint32_t index) const

Returns a sample at a given index

Parameters

index – the index of the sample in mData

Returns

the sample

inline *Vector* **getZeroVector** () const

Returns

a zero vector of length mNumDimensions

Private Members

const size_t **mNumSamples**

const size_t **mNumDimensions**

KDTree **mData**

const *TYPE* **mBandwidth**

const uint32_t **mMaxIter**

const *TYPE* **mConvergence**

VectorOfVectors **mStartingPoints**

Private Static Functions

template<typename T>

static inline auto **randomArrayBetween** (const uint32_t length, const *T* begin, const *T* end)

Generate an array of random values within a given range and a given length

Template Parameters

T – The data type

Parameters

- **length** – The length of the array
- **begin** – The minimum value contained in the array
- **end** – The maximum value contained in the array

Returns

The array

static inline auto **extractSample** (const *Matrix* &data, const uint32_t index)

Returns a sample at a given index

Parameters

- **data** – the data to extract the sample from

- **index** – the index of the sample in mData

Returns

the sample

static inline *Vector* **getZeroVector** (uint32_t numDimensions)

Returns

a zero vector of length mNumDimensions

Private Static Attributes

static constexpr int32_t **DIMS** = *SAMPLE_ORDER* == Eigen::ColMajor ? *ROWS* : *COLUMNS*

static constexpr int32_t **NOT_SAMPLE_ORDER** = (*SAMPLE_ORDER* == Eigen::ColMajor ? Eigen::RowMajor : Eigen::ColMajor)

static constexpr int32_t **STORAGE_ORDER** = *DIMS* == 1 ? *NOT_SAMPLE_ORDER* : *SAMPLE_ORDER*

class **MeanShiftEventStoreAdaptor**

#include </builds/inivation/dv/dv-processing/include/dv-processing/cluster/mean_shift/event_store_adaptor.hpp>

This class implements the Mean Shift clustering algorithm with an Epanechnikov Kernel for event store data.

As event data has a non-smooth probability density in x and y space, and the Mean Shift algorithm performs a gradient ascent, the quality of the detected clusters depends significantly on the selected bandwidth hyperparameter, as well as the underlying data and the selected kernel. Generally the Gaussian Kernel yields better results for this kind of data, however it comes with a bigger performance impact.

The Mean Shift algorithm is a nonparametric estimate of the modes of the underlying probability distribution for the data. It implements an iterative search, starting from points provided by the user, or randomly selected from the data points provided. For each iteration, the current estimate of the mode is replaced by an estimate of the mean value of the surrounding data samples. If the Epanechnikov kernel is used for the underlying density estimate, its so-called “shadow kernel”, the flat kernel must be used for the estimate of the mean. This means, that we can simply compute the average value of the data points that lie within a given radius around the current estimate of the mode, and use this as the next estimate. To provide an efficient search for the neighbours of the current mode estimate, a KD tree was used.

For the underlying theory, see “The Estimation of the Gradient of a Density Function with

Applications in Pattern Recognition” by K. Fukunaga and L. Hostetler as well as “Mean shift, mode seeking, and clustering” by Yizong Cheng.

Public Types

using **Vector** = *dv::TimedKeyPoint*

using **VectorOfVectors** = *std::vector<Vector, Eigen::aligned_allocator<Vector>>*

Public Functions

inline **MeanShiftEventStoreAdaptor** (const *dv::EventStore* &data, const int16_t bw, float conv, const uint32_t maxIter, const *VectorOfVectors* &startingPoints, const uint32_t numLeaves = 32768)

Constructor

 See also

dv::containers::KDTree

Parameters

- **data** – The Matrix containing the data. The data is neither copied nor otherwise managed, ownership remains with the user of this class.
- **bw** – The bandwidth used for the shift. This is a hyperparameter for the kernel. For the Epanechnikov kernel this means that all values within a radius of bw are averaged.
- **conv** – For each starting point, the algorithm is stopped as soon as the absolute value of the shift is $\leq$ conv.
- **maxIter** – The maximum number of iterations. Detected modes, for which the the number of iterations exceed this value are not added to the detected clusters.
- **startingPoints** – Points from which to start the search.
- **numLeaves** – the maximum number of leaves for the KDTree.

```
inline MeanShiftEventStoreAdaptor (const dv::EventStore &data, const int16_t bw, float conv, const
                                   uint32_t maxIter, VectorOfVectors &&startingPoints, const uint32_t
                                   numLeaves = 32768)
```

Constructor

 See also

dv::containers::KDTree

Parameters

- **data** – The Matrix containing the data. The data is neither copied nor otherwise managed, ownership remains with the user of this class.
- **bw** – The bandwidth used for the shift. This is a hyperparameter for the kernel. For the Epanechnikov kernel this means that all values within a radius of bw are averaged.
- **conv** – For each starting point, the algorithm is stopped as soon as the absolute value of the shift is $\leq$ conv.
- **maxIter** – The maximum number of iterations. Detected modes, for which the the number of iterations exceed this value are not added to the detected clusters.
- **startingPoints** – Points from which to start the search.
- **numLeaves** – the maximum number of leaves for the KDTree.

```
inline MeanShiftEventStoreAdaptor (const dv::EventStore &data, const int16_t bw, float conv, const
                                   uint32_t maxIter, const uint32_t numStartingPoints, const uint32_t
                                   numLeaves = 32768)
```

Constructor

 See also

dv::containers::KDTree

Parameters

- **data** – The Matrix containing the data. The data is neither copied nor otherwise managed, ownership remains with the user of this class.
- **bw** – The bandwidth used for the shift. This is a hyperparameter for the kernel. For the Epanechnikov kernel this means that all values within a radius of bw are averaged.
- **conv** – For each starting point, the algorithm is stopped as soon as the absolute value of the shift is $\leq$ conv.
- **maxIter** – The maximum number of iterations. Detected modes, for which the the number of iterations exceed this value are not added to the detected clusters.
- **numStartingPoints** – The number of points which are randomly selected from the data points, to be used as starting points.
- **numLeaves** – the maximum number of leaves for the KDTree.

MeanShiftEventStoreAdaptor () = delete**MeanShiftEventStoreAdaptor** (const *MeanShiftEventStoreAdaptor* &other) = delete**MeanShiftEventStoreAdaptor** (*MeanShiftEventStoreAdaptor* &&other) = delete*MeanShiftEventStoreAdaptor* &operator= (const *MeanShiftEventStoreAdaptor* &other) = delete*MeanShiftEventStoreAdaptor* &operator= (*MeanShiftEventStoreAdaptor* &&other) = delete**~MeanShiftEventStoreAdaptor** () = defaulttemplate<*kernel*::MeanShiftKernel **kernel** = *kernel*::Epanechnikov>inline auto **fit** ()

Executes the algorithm.

 See also

MeanShiftKernel

Template Parameters**kernel** – the kernel to be used.**Returns**

The centres of each detected cluster

Returns

The labels for each data point. The labels correspond to the index of the centre to which the sample is assigned.

Returns

The number of samples in each cluster

Returns

The in-cluster variance for each cluster

```
template<kernel::MeanShiftKernel kernel>
inline VectorOfVectors findClusterCentres ()
```

Performs the search for the cluster centres for each given starting point. A detected centre is added to the set of centres if it isn't closer than the bandwidth to any previously detected centre.

➔ See also

MeanShiftKernel

Template Parameters

kernel – the kernel to be used.

Returns

The centres of each detected cluster

```
inline std::tuple<std::vector<uint32_t>, std::vector<uint32_t>, std::vector<float>> assignClusters (const Vec-
                                                                                               torOfVec-
                                                                                               tors
                                                                                               &cluster-
                                                                                               Centres)
```

Assigns the data samples to a cluster by means of a nearest neighbour search, and computes the number of samples as well as the in-cluster variance in the process.

Parameters

clusterCentres – The centres of each detected cluster

Returns

The labels for each data point. The labels correspond to the index of the centre to which the sample is assigned.

Returns

The number of samples in each cluster

Returns

The in-cluster variance for each cluster

Public Static Functions

```
static inline VectorOfVectors generateStartingPointsFromData (const uint32_t numStartingPoints, const
                                                                dv::EventStore &data)
```

Generates a vector of vectors containing the starting points by randomly selecting from provided data. Data cannot be empty, and for best results, should contain at least more elements than the desired number of starting points given as a parameter.

Parameters

- **numStartingPoints** – The number of points to be generated.
- **data** – the event data to select the starting points from, cannot be empty.

Returns

The vector of vectors containing the starting points.

```
static inline VectorOfVectors generateStartingPointsFromRange (const uint32_t numStartingPoints, const
                                                                std::array<std::pair<int16_t, int16_t>,
                                                                2> &ranges)
```

Generates a vector of vectors containing the starting points by generating random points within a given range for each dimension

Parameters

- **numStartingPoints** – The number of points to be generated
- **ranges** – a vector containing one range per dimension. Each dimension is represented by a pair containing the beginning and the end of the range

Returns

The vector of vectors containing the starting points.

Private Types

using **KDTree** = *dv::containers::kd_tree::KDTreeEventStoreAdaptor*

Private Functions

```
template<kernel::MeanShiftKernel kernel>
inline std::optional<Vector> performShift (Vector currentMode)
```

Performs a search for a mode in the underlying density starting off with a provided initial point.

 See also

MeanShiftKernel

Template Parameters

kernel – the kernel to be used.

Parameters

currentMode – The starting point that is to be shifted until convergence.

Returns

An std::optional containing either a vector, if the search has converged, std::nullopt otherwise

```
template<kernel::MeanShiftKernel kernel>
inline float applyKernel (const float squaredDistance) const
```

Applies the selected kernel to the squared distance

 See also

MeanShiftKernel

Template Parameters

kernel – the kernel to be used.

Parameters

squaredDistance – the squared distance between the current mode estimate and a given sample point

Returns

the kernel value

```
template<kernel::MeanShiftKernel kernel>
inline auto getNeighbours (const Vector &centre)
    Returns the neighbours surrounding a centre
```

➡ See also

MeanShiftKernel

Template Parameters

kernel – the kernel to be used.

Parameters

centre – the centre surrounding which the neighbours are to be found

Returns

the neighbours, as a vector of pairs, one pair per neighbour containing a pointer to the event and a distance to the centre

```
inline float squaredDistance (const dv::TimedKeyPoint &k, const dv::Event &e) const
inline float squaredDistance (const dv::TimedKeyPoint &k1, const dv::TimedKeyPoint &k2) const
inline float squaredDistance (const dv::Event &e1, const dv::Event &e2) const

template<typename T>
inline T pow2 (const T val) const
```

Private Members

const size_t mNumSamples

KDTree mData

const int16_t mBandwidth

const uint32_t mMaxIter

const float mConvergence

const VectorOfVectors mStartingPoints

Private Static Functions

static inline Vector getZeroVector ()

class MeanShiftTracker : public dv::features::TrackerBase

#include </builds/inivation/dv/dv-processing/include/dv-processing/features/mean_shift_tracker.hpp> Track event blobs using mean shift algorithm on time surface event data.

Public Functions

```
inline MeanShiftTracker (const cv::Size &resolution, const int bandwidth, const dv::Duration timeWindow,
                          RedetectionStrategy::UniquePtr redetectionStrategy = nullptr,
                          std::unique_ptr<EventFeatureBlobDetector> detector = nullptr, const float stepSize
                          = 0.5f, const float weightMultiplier = 1.f, float convergenceNorm = 0.01f, int
                          maxIters = 2000)
```

Constructor for mean shift tracker using Epanechnikov kernel as weights for the time surface of events used to update track location. The kernel weights have highest value on the previous track location. This assumption is based on the idea that the new track location is “close” to last track location. The consecutive track updates are performed until the maximum number of iteration is reached or the shift between consecutive updates is below a threshold.

Parameters

- **resolution** – full image plane resolution
- **bandwidth** – search window dimension size. The search area is a square. The square side is 2 * bandwidth and center the current track location
- **timeWindow** – look back time from latest event: used to generate normalized time surface. All events older than (latestEventTime-timeWindow) will be discarded
- **redetectionStrategy** – strategy used to decide if and when to re-detect interesting points to track
- **detector** – detector used to re-detect tracks if redetection strategy uis defined and should happen
- **stepSize** – weight applied to shift to compute new track location. This value is in range (0, 1). A value of 0 means that no shift is performed. A value of 1 means that the new candidate center is directly assigned as new center
- **weightMultiplier** – scaling factor for Epanechnikov weights used in the computation of the mean shift cost update
- **convergenceNorm** – shift value below which search will not continue (this value is named “mode” in the docs)
- **maxIters** – maximum number of search iterations for one track update

```
inline void accept (const dv::EventStore &store)
```

Add events to time surface and update last batch of events fed to the tracker.

Parameters

store – new incoming events for the tracker.

```
inline virtual Result::SharedPtr track () override
```

Compute new centers based on area with highest event density. The density is weighted by the event times-tamp: newer timestamps have higher weight.

Returns

structure containing new track locations as a vector of *dv::TimedKeyPoint*

```
inline void setRedetectionStrategy (RedetectionStrategy::UniquePtr redetectionStrategy)
```

Define redetection strategy used to re-detect interesting points to track.

Parameters

redetectionStrategy – type of redetection to use (check redetection_strategy.hpp for available types of re-detections)

inline void **setDetector** (*std::unique_ptr<EventFeatureBlobDetector>* detector)

Define detector used to detect interesting points to track (if redetection should happen)

Parameters

detector – detector for new interesting points to track

inline int **getBandwidth** () const

Getter for bandwidth value that defines the search area for a new track. For detailed information on how the area is computed please check related parameter in constructor.

Returns

search window dimension size.

inline void **setBandwidth** (const int bandwidth)

Setter for bandwidth value.

Parameters

bandwidth – search window dimension size.

inline *dv::Duration* **getTimeWindow** () const

Get time window duration used to normalize time surface.

Returns

value of time window use to generate normalized time surface

inline void **setTimeWindow** (const *dv::Duration* timeWindow)

Setter for time window duration for time surface normalization.

Parameters

timeWindow – size of window

inline float **getStepSize** () const

Get multiplier value used for track location update. Given a computed shift to be applied to a track, the actual shift performed is given by $mStepSize * shift$.

Returns

scaling value applied to the spatial interval computed between current and new track position at consecutive updates

inline void **setStepSize** (const float stepSize)

Setter for learning rate for motion towards new center during one mean shift iteration. Please check the same parameter in the constructor description for detailed information.

Parameters

stepSize – weight applied to shift to compute new track location.

inline float **getWeightMultiplier** () const

Getter for weight multiplier used to adjust weight of each time surface value in the mean shift update. If multiplier is smaller than 1, the cost values for each location are shrink, whereas if the multiplier is larger than 1, the difference between time surface intensities will be larger.

Returns

weight multiplier value

inline void **setWeightMultiplier** (const float multiplier)

Setter for scaling factor used in the computation of the mean shift cost update.

Parameters

multiplier – scaling factor value

inline float **getConvergenceNorm** () const

Get norm of distance between consecutive tracks updates. If the distance is smaller than this norm, the track update is considered to be converged.

Returns

value of distance norm between consecutive updates

inline void **setConvergenceNorm** (const float norm)

Setter for threshold norm (i.e. mode) between consecutive track updates below which iterations are stopped.

Parameters

norm – threshold value

inline int **getMaxIterations** () const

Get maximum number of times track update can be run.

Returns

value of maximum number of operations for track update

inline void **setMaxIterations** (const int maxIters)

Setter for maximum number of track updates.

Parameters

maxIters – value of maximum number of operations for track update

Private Functions

inline Result::SharedPtr **updateTracks** (const cv::Mat &normalizedTimeSurface)

Compute new location for all tracks. If a new position fall inside the area of a new position computed for a previous track, the track will not be updated. Previous track with its timestamp will be kept.

Parameters

normalizedTimeSurface – image representation of event timestamps based on time surface

Returns

updated track positions

inline *std::optional*<*dv::Point2f*> **computeShift** (const *dv::Point2f* ¢er, const cv::Mat &timeSurface, const float trackSize)

Compute new track location. Note: kernel weights are updated only if the search window changed size or if it intersects the boundaries of the image plane. This decision has been made for performance reasons and should not affect the final result as long as the new track position is “close enough”, to the starting position.

Parameters

- **center** – previous track location
- **timeSurface** – Matrix containing normalized time surface values
- **trackSize** – dimension of track determining kernel size

Returns

new final track location if value is valid, *std::nullopt* is returned if the search area has no event data inside it.

inline *std::optional*<*dv::Point2f*> **updateCenterLocation** (const cv::Mat &spatialWindow, const cv::Mat &kernelWeights) const

Compute mode (i.e. track location).

Parameters

- **spatialWindow** – image plane sub-matrix in which the center will be updated
- **kernelWeights** – weights of Epanechnikov kernel applied to each time surface location inside the given spatial window

Returns

new track location

```
inline cv::Mat kernelEpanechnikovWeights (const dv::Point2f &center, const cv::Rect &>window, const float cutOffValue) const
```

Compute Epanechnikov kernel with highest peak at center location.

Parameters

- **center** – location of the observed sample, i.e. location where kernel will have highest response
- **window** – window in which kernel weights will be computed
- **cutOffValue** – distance from center after which all kernel weights will be zero. For more information about the kernel please check equation 3.54 from http://sfb649.wiwi.hu-berlin.de/fedc_homepage/xplore/ebooks/html/spm/spmhtmlnode18.html

Returns

matrix with weights of Epanechnikov kernel

```
inline std::pair<cv::Mat, cv::Rect> findSpatialWindow (const dv::Point2f &center, const cv::Mat &image) const
```

Compute area in which the new track position will be searched. This area depends on the bandwidth value. The search area is defined as the square around the center value with size of one side as 2*bandwidth. We return the selected area as first argument and the roi in the full image plane to be able to retrieve coordinates of selected area in the original image space.

Parameters

- **center** – previous track center around which we define the search area
- **image** – full image plane data

Returns

pair containing as first output the matrix block containing the data inside the image defined by the rectangle returned as second output

```
inline void runRedetection (Result::SharedPtr &result)
```

Re-detect interesting points

Parameters

result – current set of tracks to which new detections will be added

Private Members

```
int mBandwidth
```

parameter defining search window size for each track update

```
dv::TimeSurface mSurface
```

event time surface

```
dv::Duration mTimeWindow
```

time window of events to generate the normalized time surface from

float **mStepSize**

cv::Size **mResolution**

dv::EventStore **mEvents** = *dv::EventStore*()

latest batch of events fed to the tracker

std::unique_ptr<EventFeatureBlobDetector> **mDetector**

detector used if no track has been detected or redetection is expected to happen

int32_t **mLastFreeClassId** = 0

value used to keep track of first free ID for a new track

RedetectionStrategy::UniquePtr **mRedetectionStrategy** = nullptr

type of redetection strategy used to detect new interesting points to track

float **mWeightMultiplier**

Weight multiplier used to adjust weight of each point in the mean shift update. If multiplier is smaller than 1, the cost values for each location are shrink, whereas if the multiplier is larger than 1, the difference between points with lower intensity in the time surface will be increased from ones with larger intensity values

float **mConvergenceNorm**

shift value below which search will not continue

int **mMaxIters**

maximum number of search iterations for one track update

struct **Metadata**

Public Functions

Metadata () = default

inline explicit **Metadata** (const cv::Size &patternShape_, const cv::Size &internalPatternShape_, const *std::string_view* patternType_, const float patternSize_, const float patternSpacing_, const *std::optional*<float> &calibrationError_, const *std::string_view* calibrationTime_, const *std::string_view* quality_, const *std::string_view* comment_, const *std::optional*<float> &pixelPitch_)

inline explicit **Metadata** (const boost::property_tree::ptree &tree)

Create an instance of metadata from a property tree structure.

Parameters

tree – Property tree to be parsed.

Returns

Constructed *Metadata* instance.

```
inline boost::property_tree::ptree toPropertyTree() const
```

Serialize the metadata structure into a property tree.

Returns

Serialized property tree.

```
inline bool operator==(const Metadata &rhs) const
```

Equality operator.

Parameters

rhs –

Returns

Public Members

```
cv::Size patternShape
```

Shape of the calibration pattern.

```
cv::Size internalPatternShape
```

Shape of the calibration pattern in terms of internal intersections.

```
std::string patternType
```

Type of the calibration pattern used (e.g. apriltag)

```
float patternSize = -1.f
```

Size of the calibration pattern in [m].

```
float patternSpacing = -1.f
```

Ratio between tags to patternSize (apriltag only)

```
std::optional<float> calibrationError
```

Calibration reprojection error.

```
std::string calibrationTime
```

Timestamp when the calibration was conducted.

```
std::string quality
```

Description of the calibration quality (excellent/good/bad etc)

```
std::string comment
```

Any additional information.

```
std::optional<float> pixelPitch
```

Pixel pitch in meters.

```
struct Metadata
```

Public Functions

Metadata () = default

inline explicit **Metadata** (const *std::string_view* calibrationTime_, const *std::string_view* comment_)

inline explicit **Metadata** (const boost::property_tree::ptree &tree)

inline boost::property_tree::ptree **toPropertyTree** () const

inline bool **operator==** (const *Metadata* &rhs) const

Public Members

std::string **calibrationTime**

Timestamp when the calibration was conducted.

std::string **comment**

Any additional information.

struct **Metadata**

#include </builds/inivation/dv/dv-processing/include/dv-processing/camera/calibrations/stereo_calibration.hpp>
Metadata for the stereo calibration.

Public Functions

Metadata () = default

inline explicit **Metadata** (const *std::optional<float>* &epipolarError_, const *std::string_view* comment_)

inline explicit **Metadata** (const boost::property_tree::ptree &tree)

inline boost::property_tree::ptree **toPropertyTree** () const

Serialize into a property tree.

Returns

inline bool **operator==** (const *Metadata* &rhs) const

Public Members

std::optional<float> **epipolarError**

Average epipolar error.

std::string **comment**

Any additional information.

class **MonoCameraRecording** : public *dv::io::InputBase*

#include </builds/inivation/dv/dv-processing/include/dv-processing/io/mono_camera_recording.hpp> A convenience class for reading recordings containing data captured from a single camera. Looks for an event, frame, imu, and trigger streams within the supplied aedat4 file.

Public Functions

inline explicit **MonoCameraRecording** (const *std::shared_ptr<ReadOnlyFile>* &fileReader, const *std::string* &cameraName = "")

Create a reader that reads single camera data recording from a pre-constructed file reader.

Parameters

- **fileReader** – A pointer for pre-constructed file reader.
- **cameraName** – Name of the camera in the recording. If an empty string is passed (the default value), reader will try detect the name of the camera. In case recording contains more than one camera, it will choose the first encountered name and ignore streams that were recorded by a different camera.

inline explicit **MonoCameraRecording** (const *std::filesystem::path* &aedat4Path, const *std::string* &cameraName = "")

Create a reader that reads single camera data recording from an aedat4 file.

Parameters

- **aedat4Path** – Path to the aedat4 file.
- **cameraName** – Name of the camera in the recording. If an empty string is passed (the default value), reader will try detect the name of the camera. In case recording contains more than one camera, it will choose the first encountered name and ignore streams that were recorded by a different camera.

inline virtual *std::optional<dv::Frame>* **getNextFrame** () override

Sequential read of a frame, tries reading from stream named “frames”. This function increments an internal seek counter which will return the next frame at each call.

Returns

A *dv::Frame* or *std::nullopt* if the frame stream is not available or the end-of-stream was reached.

inline *std::optional<dv::Frame>* **getNextFrame** (const *std::string* &streamName)

Sequential read of a frame. This function increments an internal seek counter which will return the next frame at each call.

Parameters

streamName – Name of the stream, if an empty name is passed, it will select any one stream with frame data type.

Returns

A *dv::Frame*, *std::nullopt* if the frame stream is not available or the end-of-stream was reached.

inline virtual bool **isStreamAvailable** (const *std::string_view* streamName) const override

Check whether a given stream name is available.

Parameters

streamName – Name of the stream.

Returns

True if this stream is available, false otherwise.

inline *std::vector<std::string>* **getStreamNames** () const

Return a vector containing all available stream names.

Returns

A list of custom data type stream names.

```
template<class DataType>
```

```
inline std::optional<DataType> getNextStreamPacket (const std::string &streamName)
```

Read a custom data type packet sequentially.

Custom data types are any flatbuffer generated types that are not the following: *dv::EventPacket*, *dv::TriggerPacket*, *dv::IMUPacket*, *dv::Frame*.

Template Parameters

DataType – Custom data packet class.

Parameters

streamName – Name of the stream.

Throws

- *InvalidArgument* – An exception is thrown if a stream with given name is not found in the file.
- *InvalidArgument* – An exception is thrown if given type does not match the type identifier of the given stream.

Returns

Next packet within given stream or *std::nullopt* in case of end-of-stream.

```
inline virtual std::optional<dv::EventStore> getNextEventBatch () override
```

Sequential read of events, tries reading from stream named “events”. This function increments an internal seek counter which will return the next event batch at each call.

Returns

A *dv::EventStore* or *std::nullopt* if the frame stream is not available or the end-of-stream was reached.

```
inline std::optional<dv::EventStore> getNextEventBatch (const std::string &streamName)
```

Sequentially read a batch of recorded events. This function increments an internal seek counter which will return the next batch at each call.

Parameters

streamName – Name of the stream, if an empty name is passed, it will select any one stream with event data type.

Returns

A vector containing events, *std::nullopt* if the event stream is not available or the end-of-stream was reached.

```
inline virtual std::optional<std::vector<dv::IMU>> getNextImuBatch () override
```

Sequential read of imu data, tries reading from stream named “imu”. This function increments an internal seek counter which will return the next imu data batch at each call.

Returns

A vector or *IMU* measurements or *std::nullopt* if the imu data stream is not available or the end-of-stream was reached.

```
inline std::optional<std::vector<dv::IMU>> getNextImuBatch (const std::string &streamName)
```

Sequentially read a batch of recorded imu data. This function increments an internal seek counter which will return the next batch at each call.

Parameters

streamName – Name of the stream, if an empty name is passed, it will select any one stream with imu data type.

Returns

A vector containing imu data, `std::nullopt` if the imu data stream is not available or the end-of-stream was reached.

inline virtual `std::optional<std::vector<dv::Trigger>>` **getNextTriggerBatch()** override

Sequential read of trigger data, tries reading from stream names “triggers”. This function increments an internal seek counter which will return the next trigger data batch at each call.

Returns

A vector of trigger data or `std::nullopt` if the frame stream is not available or the end-of-stream was reached.

inline `std::optional<std::vector<dv::Trigger>>` **getNextTriggerBatch** (const `std::string` &streamName)

Sequentially read a batch of recorded triggers. This function increments an internal seek counter which will return the next batch at each call.

Parameters

streamName – Name of the stream, if an empty name is passed, it will select any one stream with trigger data type.

Returns

A vector containing triggers, `std::nullopt` if the trigger stream is not available or the end-of-stream was reached.

inline void **resetSequentialRead** ()

Reset the sequential read function to start from the beginning of the file.

inline virtual bool **isRunning** () const override

Check whether any input data streams have terminated. For a live camera this should check if the device is still connected and functioning, while for a recording file this should check if any of the data streams have reached end-of-file (EOF). For a network input, this indicates the network stream is still connected.

Returns

True if data read on all streams is still possible, false otherwise.

inline virtual bool **isRunning** (const `std::string_view` streamName) const override

Check whether the input data stream with the specified name is still active.

Returns

True if data read on this stream is possible, false otherwise.

inline virtual bool **isRunningAny** () const override

Check whether any input data streams are still available. For a live camera this should check if the device is still connected and functioning and at least one data stream is active (different than `isRunning()`), while for a recording file this should check if any of the data streams have not yet reached end-of-file (EOF) and are still readable. For a network input, this indicates the network stream is still connected.

Returns

True if data read on at least one stream is still possible, false otherwise.

inline `std::optional<dv::EventStore>` **getEventsTimeRange** (const int64_t startTime, const int64_t endTime, const `std::string` &streamName = "events")

Get events within given time range [startTime; endTime).

Parameters

- **startTime** – Start timestamp of the time range.
- **endTime** – End timestamp of the time range.

- **streamName** – Name of the stream, if an empty name is passed, it will select any one stream with event data type.

Returns

`dv::EventStore` with events in the time range if the event stream is available, `std::nullopt` otherwise.

```
inline std::optional<std::vector<dv::Frame>> getFramesTimeRange (const int64_t startTime, const int64_t  
                                                                endTime, const std::string &streamName  
                                                                = "frames")
```

Get frames within given time range [startTime; endTime).

Parameters

- **startTime** – Start timestamp of the time range.
- **endTime** – End timestamp of the time range.
- **streamName** – Name of the stream, if an empty name is passed, it will select any one stream with frame data type.

Throws

`InvalidArgument` – If frame stream doesn't exist or a stream with given name doesn't exist.

Returns

Vector containing frames and timestamps.

```
template<class DataType>
```

```
inline std::optional<std::vector<DataType>> getStreamTimeRange (const int64_t startTime, const int64_t  
                                                                endTime, const std::string  
                                                                &streamName)
```

Get packets from a stream within given period of time. Returns a vector of packets. If a packet contains elements that are outside of given time range, the internal elements will be cut to match exactly the [startTime; endTime). If stream does not contain any packets within requested time range, the function returns an empty vector.

Template Parameters

DataType – Packet type

Parameters

- **startTime** – Period start timestamp.
- **endTime** – Period end timestamp.
- **streamName** – Name of the stream, empty string will pick a first stream with matching type.

Throws

- `InvalidArgument` – An exception is thrown if a stream with given name is not found in the file.
- `InvalidArgument` – An exception is thrown if given type does not match the type identifier of the given stream.

Returns

A vector of packets containing the data only within [startTime; endTime) period.

```
inline std::optional<std::vector<dv::IMU>> getImuTimeRange (const int64_t startTime, const int64_t endTime,  
                                                            const std::string &streamName = "imu")
```

Get *IMU* data within given time range [startTime; endTime).

Parameters

- **startTime** – Start timestamp of the time range.
- **endTime** – End timestamp of the time range.
- **streamName** – Name of the stream, if an empty name is passed, it will select any one stream with imu data type.

Returns

Vector containing *IMU* data if the *IMU* stream is available, `std::nullopt` otherwise.

```
inline std::optional<std::vector<dv::Trigger>> getTriggersTimeRange (const int64_t startTime, const int64_t
                                                                    endTime, const std::string
                                                                    &streamName = "triggers")
```

Get trigger data within given time range [startTime; endTime).

Parameters

- **startTime** – Start timestamp of the time range.
- **endTime** – End timestamp of the time range.
- **streamName** – Name of the stream, if an empty name is passed, it will select any one stream with trigger data type.

Returns

Vector containing triggers if the trigger stream is available, `std::nullopt` otherwise.

```
inline virtual bool isFrameStreamAvailable () const override
```

Check whether frame stream is available. Specifically checks whether a stream named “frames” is available since it’s the default stream name for frames.

Returns

True if the frame stream is available.

```
inline bool isFrameStreamAvailable (const std::string &streamName) const
```

Checks whether a frame data stream is present in the file.

Parameters

streamName – Name of the stream, if an empty name is passed, it will select any one stream with frame data type.

Returns

True if the frames are available, false otherwise.

```
inline virtual bool isEventStreamAvailable () const override
```

Check whether event stream is available. Specifically checks whether a stream named “events” is available since it’s the default stream name for events.

Returns

True if the event stream is available, false otherwise.

```
inline bool isEventStreamAvailable (const std::string &streamName) const
```

Checks whether an event data stream is present in the file.

Parameters

streamName – Name of the stream, if an empty name is passed, it will select any one stream with event data type.

Returns

True if the events are available, false otherwise.

inline virtual bool **isImuStreamAvailable** () const override

Check whether imu data stream is available. Specifically checks whether a stream named “imu” is available since it’s the default stream name for imu data.

Returns

True if the imu stream is available, false otherwise.

inline bool **isImuStreamAvailable** (const *std::string* &streamName) const

Checks whether an imu data stream is present in the file.

Parameters

streamName – Name of the stream, if an empty name is passed, it will select any one stream with *IMU* data type.

Returns

True if the imu data is available, false otherwise.

inline virtual bool **isTriggerStreamAvailable** () const override

Check whether trigger stream is available. Specifically checks whether a stream named “triggers” is available since it’s the default stream name for trigger data.

Returns

True if the trigger stream are available, false otherwise.

inline bool **isTriggerStreamAvailable** (const *std::string* &streamName) const

Checks whether a trigger data stream is present in the file.

Parameters

streamName – Name of the stream, if an empty name is passed, it will select any one stream with trigger data type.

Returns

True if the triggers are available, false otherwise.

inline *std::pair*<int64_t, int64_t> **getTimeRange** () const

Return a pair containing start (first) and end (second) time of the recording file.

Returns

A pair containing start and end timestamps for the recording.

inline *dv::Duration* **getDuration** () const

Return the duration of the recording.

Returns

Duration value holding the total playback time of the recording.

inline virtual *std::string* **getCameraName** () const override

Return the camera name that is detected in the recording.

Returns

String containing camera name.

inline *DataReadVariant* **readNext** ()

Read next packet in the recorded stream, the function returns a *std::variant* containing one of the following types:

- *dv::EventStore*
- *dv::Frame*
- *std::vector*<*dv::IMU*>

- `std::vector<dv::Trigger>`
- `dv::io::MonoCameraRecording::OutputFlag` The `OutputFlag` is used to determine when the end of file is reached. If the reader encounters an unsupported type, the data will be skipped and will seek until a packet containing a supported type is reached.

Returns

`std::variant` containing a packet with data of one of the supported types.

inline bool **handleNext** (*DataReadHandler* &handler)

Read next packet from the recording and use a handler object to handle all types of packets. The function returns a true if end-of-file was not reached, so this function call can be used in a while loop like so:

```
while (recording.handleNext(handler)) {
    // While-loop executes after each packet
}
```

Parameters

handler –

Returns

inline void **run** (*DataReadHandler* &handler)

Sequentially read all packets from the recording and apply handler to each packet. This is a blocking call.

Parameters

handler – Handler class containing lambda functions for each supported packet type.

inline virtual *std::optional<cv::Size>* **getEventResolution** () const override

Get event stream resolution for the “events” stream.

Returns

Resolution of the “events” stream.

inline *std::optional<cv::Size>* **getEventResolution** (const *std::string* &streamName) const

Get the resolution of the event data stream if it is available.

Parameters

streamName – Name of the stream, if an empty name is passed, it will select any one stream with event data type.

Returns

Returns the resolution of the event data if available, `std::nullopt` otherwise.

inline virtual *std::optional<cv::Size>* **getFrameResolution** () const override

Get frame stream resolution for the “frames” stream.

Returns

Resolution of the “frames” stream.

inline *std::optional<cv::Size>* **getFrameResolution** (const *std::string* &streamName) const

Get the resolution of the frame data stream if it is available.

Parameters

streamName – Name of the stream, if an empty name is passed, it will select any one stream with frame data type.

Returns

Returns the resolution of the frames if available, `std::nullopt` otherwise.

```
inline const std::map<std::string, std::string> &getStreamMetadata (const std::string &streamName)
```

Get all metadata of a stream.

Parameters

streamName – Name of the stream.

Throws

out_of_range – Out of range exception is thrown if a stream with given name is not available.

Returns

A map containing key-value strings of each available metadata of a requested stream.

```
inline std::optional<std::string> getStreamMetadataValue (const std::string &streamName, const std::string &key)
```

Get a value of a given metadata key. Throws an exception if given stream doesn't exist and returns *std::nullopt* if a metadata entry with given key is not found for the stream.

Parameters

- **streamName** – Name of the stream.
- **key** – Key string of the metadata.

Throws

out_of_range – Out of range exception is thrown if a stream with given name is not available.

Returns

Metadata entry with given key is found for the stream, *std::nullopt* otherwise.

```
template<class DataType>
```

```
inline bool isStreamOfDataType (const std::string &streamName) const
```

Check whether a stream is of a given data type.

Template Parameters

DataType – Data type to be checked.

Parameters

streamName – Name of the stream.

Throws

out_of_bounds – Out of bounds exception is thrown if stream of a given name is not found.

Returns

True if the given stream contains *DataType* data.

Private Types

```
using StreamInfoMap = std::map<std::string, StreamDescriptor, std::less<>>
```

Private Functions

```
inline const dv::io::Stream *getStream (const int streamId) const
```

```
inline void parseStreamIds ()
```

```
template<class DataType>
```

```
inline StreamInfoMap::iterator getStreamInfo (const std::string &streamName)
```

```
template<class DataType>
```

```
inline StreamInfoMap::const_iterator getStreamInfo (const std::string &streamName) const

template<class DataType>
inline std::shared_ptr<DataType> getNextPacket (StreamDescriptor &streamInfo)
```

Private Members

```
std::shared_ptr<ReadOnlyFile> mReader = nullptr

FileInfo mInfo

std::string mCameraName

std::vector<FileDataDefinition>::const_iterator mPacketIter

bool eofReached = false

StreamInfoMap mStreamInfo
```

Private Static Functions

```
template<class VectorClass>
static inline void trimVector (VectorClass &vector, int64_t start, int64_t end)

Trim a vector containing elements with a timestamp. Retains only the data within [start; end).
```

Template Parameters

VectorClass – The class of the vector

Parameters

- **vector** – The vector of data
- **start** – Start timestamp (inclusive start of range)
- **end** – End timestamp (exclusive end of range)

```
class MonoCameraWriter : public dv::io::OutputBase
```

Public Functions

```
inline MonoCameraWriter (const std::filesystem::path &aedat4Path, const MonoCameraWriter::Config &config,
                        const dv::io::support::TypeResolver &resolver =
                        dv::io::support::defaultTypeResolver)
```

Create an aedat4 file writer with simplified API.

Parameters

- **aedat4Path** – Path to the output file. The file is going to be overwritten.
- **config** – *Writer* config. Defines expected output streams and recording metadata.
- **resolver** – Type resolver for the output file.

```
inline MonoCameraWriter (const std::filesystem::path &aedat4Path, const dv::io::camera::CameraInputBase
                        &capture, const CompressionType compression = CompressionType::LZ4, const
                        dv::io::support::TypeResolver &resolver = dv::io::support::defaultTypeResolver)
```

Create an aedat4 file writer that inspects the capabilities and configuration from a `dv::io::CameraCapture` class. This will enable all available data streams present from the camera capture.

Parameters

- **aedat4Path** – Path to the output file. The file is going to be overwritten.
- **capture** – Direct camera capture instance. This is used to inspect the available data streams and metadata of the camera.
- **compression** – Compression to be used for the output file.
- **resolver** – Type resolver for the output file.

```
inline void writeEventPacket (const dv::EventPacket &events, const std::string &streamName = "events")
```

Write an event packet into the output file.

The data is passed directly into the serialization procedure without performing copies. Data is serialized and the actual file IO is performed on a separate thread.

Parameters

- **events** – Packet of events.
- **streamName** – Name of the stream, an empty string will match first stream with compatible data type.

Throws

`invalid_argument` – Invalid argument exception is thrown if function is called and compatible output stream was not added during construction.

```
inline void writeEvents (const dv::EventStore &events, const std::string &streamName)
```

Write an event store into the output file. The store is written by maintaining internal data partial ordering and fragmentation.

The data is passed directly into the serialization procedure without performing copies. Data is serialized and the actual file IO is performed on a separate thread.

Parameters

- **events** – Store of events.
- **streamName** – Name of the stream, an empty string will match first stream with compatible data type.

Throws

`invalid_argument` – Invalid argument exception is thrown if function is called and compatible output stream was not added during construction.

```
inline void writeFrame (const dv::Frame &frame, const std::string &streamName)
```

Write a frame image into the file.

The data is passed directly into the serialization procedure without performing copies. Data is serialized and the actual file IO is performed on a separate thread.

NOTE: if the frame contains an empty image, it will be ignored and not recorded.

Parameters

- **frame** – A frame to be written.

- **streamName** – Name of the stream, an empty string will match first stream with compatible data type.

Throws

invalid_argument – Invalid argument exception is thrown if function is called and compatible output stream was not added during construction.

```
inline void writeImuPacket (const dv::IMUPacket &packet, const std::string &streamName = "imu")
```

Write a packet of imu data into the file.

The data is passed directly into the serialization procedure without performing copies. Data is serialized and the actual file IO is performed on a separate thread.

Parameters

- **packet** – *IMU* measurement packet.
- **streamName** – Name of the stream, an empty string will match first stream with compatible data type.

Throws

invalid_argument – Invalid argument exception is thrown if function is called and compatible output stream was not added during construction.

```
inline void writeImu (const dv::IMU &imu, const std::string &streamName = "imu")
```

Write an *IMU* measurement.

This function is not immediate, it batches the measurements until a configured amount is reached, only then the data is passed to the serialization step. Only then the data will be passed to the file write IO thread. If the file is closed (the object gets destroyed), destructor will dump the rest of the buffered measurements to the serialization step.

➡ See also

[*setPackagingCount*](#)

Parameters

- **imu** – A single *IMU* measurement.
- **streamName** – Name of the stream, an empty string will match first stream with compatible data type.

Throws

invalid_argument – Invalid argument exception is thrown if function is called and compatible output stream was added enabled during construction.

```
inline void writeTriggerPacket (const dv::TriggerPacket &packet, const std::string &streamName = "triggers")
```

Write a packet of trigger data into the file.

The data is passed directly into the serialization procedure without performing copies. Data is serialized and the actual file IO is performed on a separate thread.

Parameters

- **packet** – *Trigger* data packet.
- **streamName** – Name of the stream, an empty string will match first stream with compatible data type.

Throws

`invalid_argument` – Invalid argument exception is thrown if function is called and compatible output stream was added enabled during construction.

inline void **writeTrigger** (const *dv::Trigger* &trigger, const *std::string* &streamName = "triggers")

Write a *Trigger* measurement.

This function is not immediate, it batches the measurements until a configured amount is reached, only then the data is passed to the serialization step. Only then the data will be passed to the file write IO thread. If the file is closed (the object gets destroyed), destructor will dump the rest of the buffered measurements to the serialization step.

 See also

setPackagingCount

Parameters

- **trigger** – A single *Trigger* measurement.
- **streamName** – Name of the stream, an empty string will match first stream with compatible data type.

Throws

`invalid_argument` – Invalid argument exception is thrown if function is called and compatible output stream was not added during construction.

template<class **PacketType**>

inline void **writePacket** (const *PacketType* &packet, const *std::string* &stream)

Write a packet into a named stream.

Template Parameters

PacketType – Type of data packet.

Parameters

- **stream** – Name of the stream, an empty string will match first stream with compatible data type.
- **packet** – Data packet

Throws

- `InvalidArgument` – If a stream with given name is not configured.
- `InvalidArgument` – If a stream with given name is configured for a different type of data packet.
- `invalid_argument` – Invalid argument exception is thrown if function is called and compatible output stream was added enabled during construction.

template<class **PacketType**, class **ElementType**>

inline void **writePacketElement** (const *ElementType* &element, const *std::string* &streamName)

Write a single element into packet. A packet will be created per stream and element will be added until packaging count is reached, at that point the packet will be written do disk.

Template Parameters

- **PacketType** – Type of the packet to hold the elements.
- **ElementType** – Type of an element.

Parameters

- **element** – Element to be saved.
- **streamName** – Name of the stream, an empty string will match first stream with compatible data type.

inline void **setPackagingCount** (const size_t packagingCount)

Set the size batch size for trigger and imu buffering. The single measurements passed into `writeTrigger` and `writeImu` functions will be packed into batches of the given size before writing to the file.

A packaging value of 0 or 1 will cause each measurement to be serialized immediately.

 See also

writeTrigger

 See also

writeImu

Parameters

packagingCount – *Trigger* and *IMU* measurement packet size that is batched up using the `writeImu` and `writeTrigger` functions.

inline bool **isEventStreamConfigured** (const std::string &streamName = "events") const

Check if the event stream is configured for this writer.

Parameters

streamName – Name of the stream, an empty string will match first stream with compatible data type.

Returns

True if event stream is configured, false otherwise.

inline bool **isFrameStreamConfigured** (const std::string &streamName = "frames") const

Check if the frame stream is configured for this writer.

Parameters

streamName – Name of the stream, an empty string will match first stream with compatible data type.

Returns

True if frame stream is configured, false otherwise.

inline bool **isImuStreamConfigured** (const std::string &streamName = "imu") const

Check if the *IMU* stream is configured for this writer.

Parameters

streamName – Name of the stream, an empty string will match first stream with compatible data type.

Returns

True if *IMU* stream is configured, false otherwise.

```
inline bool isTriggerStreamConfigured (const std::string &streamName = "triggers") const
```

Check if the trigger stream is configured for this writer.

Parameters

streamName – Name of the stream, an empty string will match first stream with compatible data type.

Returns

True if trigger stream is configured, false otherwise.

```
template<class PacketType>
```

```
inline bool isStreamConfigured (const std::string &streamName) const
```

Check whether a stream with given name and compatible data type is configured.

Template Parameters

PacketType – Type of the packet to hold the elements.

Parameters

streamName – Name of the stream, an empty string will match first stream with compatible data type.

Returns

```
inline ~MonoCameraWriter ()
```

```
inline virtual void writeEvents (const dv::EventStore &events) override
```

Write event data into the output.

Parameters

events – Write events into the output.

```
inline virtual void writeFrame (const dv::Frame &frame) override
```

Write a frame into the output.

Parameters

frame – Write a frame into the output.

```
inline virtual void writeImu (const std::vector<dv::IMU> &imu) override
```

Write imu data into the output.

Parameters

imu – Write imu into the output.

```
inline virtual void writeTriggers (const std::vector<dv::Trigger> &triggers) override
```

Write trigger data into the output.

Parameters

triggers – Write trigger into the output.

```
inline virtual std::string getCameraName () const override
```

Retrieve camera name of this writer output instance.

Returns

Configured camera name.

Public Static Functions

```
static inline Config EventOnlyConfig (const std::string &cameraName, const cv::Size &resolution,  
                                         dv::CompressionType compression = dv::CompressionType::LZ4)
```

Generate a config for a writer that will expect a stream of events only.

Parameters

- **cameraName** – Name of the camera.
- **resolution** – Camera sensor resolution.
- **compression** – Compression type.

Returns

A config template for *MonoCameraWriter*.

```
static inline Config FrameOnlyConfig (const std::string &cameraName, const cv::Size &resolution,
                                     dv::CompressionType compression = dv::CompressionType::LZ4)
```

Generate a config for a writer that will expect a stream of frames only.

Parameters

- **cameraName** – Name of the camera.
- **resolution** – Camera sensor resolution.
- **compression** – Compression type.

Returns

A config template for *MonoCameraWriter*.

```
static inline Config DVSConfig (const std::string &cameraName, const cv::Size &resolution,
                               dv::CompressionType compression = dv::CompressionType::LZ4)
```

Generate a config for a writer that will expect data from a DVS camera - events, *IMU*, triggers.

Parameters

- **cameraName** – Name of the camera.
- **resolution** – Camera sensor resolution.
- **compression** – Compression type.

Returns

A config template for *MonoCameraWriter*.

```
static inline Config DAVISConfig (const std::string &cameraName, const cv::Size &resolution,
                                 dv::CompressionType compression = dv::CompressionType::LZ4)
```

Generate a config for a writer that will expect data from a DAVIS camera - frames, events, *IMU*, triggers.

Parameters

- **cameraName** – Name of the camera.
- **resolution** – Camera sensor resolution.
- **compression** – Compression type.

Returns

A config template for *MonoCameraWriter*.

```
static inline Config CaptureConfig (const dv::io::camera::CameraInputBase &capture, dv::CompressionType
                                   compression = dv::CompressionType::LZ4)
```

Generate a config from a camera capture instance, this only checks whether camera provides frame data stream or not and enables all available streams to be recorded.

Parameters

- **capture** – Camera capture class instance.
- **compression** – Compression type.

Returns

A config template for *MonoCameraWriter*.

Private Types

```
typedef std::map<std::string, StreamDescriptor> StreamDescriptorMap
```

Private Functions

```
inline std::string createHeader (const MonoCameraWriter::Config &config, const dv::io::support::TypeResolver &resolver)
```

```
template<class PacketType>
```

```
inline StreamDescriptorMap::iterator findStreamDescriptor (const std::string &streamName)
```

```
template<class PacketType>
```

```
inline StreamDescriptorMap::const_iterator findStreamDescriptor (const std::string &streamName) const
```

```
inline explicit MonoCameraWriter (const std::shared_ptr<dv::io::WriteOnlyFile> &outputFile, const dv::io::MonoCameraWriter::Config &config, const dv::io::support::TypeResolver &resolver = dv::io::support::defaultTypeResolver)
```

Preconfigured output file constructor. Internal use only, used for multi-camera recording.

Parameters

- **outputFile** – *WriteOnlyFile* instance to write data.
- **config** – Output stream configuration.
- **resolver** – Type resolver for the output file.

Private Members

```
size_t mPackagingCount = 20
```

```
MonoCameraWriter::Config inputConfig
```

```
StreamDescriptorMap mOutputStreamDescriptors
```

```
dv::io::support::XMLTreeNode mRoot
```

```
std::shared_ptr<dv::io::WriteOnlyFile> mOutput
```

Private Static Functions

```
static inline void validateConfig (const MonoCameraWriter::Config &config)
```

Friends

```
friend class StereoCameraWriter
```

```
template<class Accumulator = dv::EdgeMapAccumulator, class PixelPredictor =  
kinematics::PixelMotionPredictor>  
class MotionCompensator
```

Public Functions

```
inline const Info &getInfo () const
```

Return an info class instance containing motion compensator state for the algorithm iteration. The info object contains debug information about the execution of the motion compensator.

Returns

```
inline void accept (const Transformationf &transform)
```

Push camera pose measurement.

Parameters

transform – Transform representing camera pose in some fixed reference frame (e.g. World coordinates).

```
inline void accept (const dv::measurements::Depth &timeDepth)
```

Scene depth measurement in meters.

Parameters

timeDepth – A pair containing measured depth into the scene and a timestamp at when the measurement was performed.

```
inline void accept (const dv::EventStore &events)
```

Push event camera input.

Parameters

events – Pixel brightness changes from an event camera.

```
inline void accept (const dv::Event &event)
```

Push event camera input.

Parameters

event – Pixel brightness change from an event camera.

```
inline dv::EventStore generateEvents (const int64_t generationTime = -1)
```

Generate the motion compensated events contained in the buffer.

Parameters

generationTime – Provide a timestamp to which point in time the motion compensator compensates into, negative values will cause the function to use highest timestamp value in the event buffer.

Returns

Motion compensated events.

```
inline dv::Frame generateFrame (const int64_t generationTime = -1)
```

Generate the motion compensated frame output and reset the events contained in the buffer.

Parameters

generationTime – Provide a timestamp to which point in time the motion compensator compensates into, negative values will cause the function to use highest timestamp value in the event buffer.

Returns

Motion compensated frame.

inline void **reset** ()

Clear the event buffer.

inline *MotionCompensator* &**operator**<< (const *dv::EventStore* &store)

Accept the event data using the stream operator.

Parameters

store – Input event store.

Returns

Reference to current object instance.

inline *MotionCompensator* &**operator**<< (const *dv::Event* &event)

Accept the event data using the stream operator.

Parameters

store – Input event.

Returns

Reference to current object instance.

inline *dv::Frame* &**operator**>> (*dv::Frame* &image)

Output stream operator which generates a frame.

Parameters

image – Motion compensated frame.

Returns

Motion compensated frame.

inline **MotionCompensator** (const *camera::CameraGeometry::SharedPtr* &cameraGeometry,
std::unique_ptr<Accumulator> accumulator_ = nullptr)

Construct a motion compensator instance with custom accumulator.

Parameters

- **cameraGeometry** – Camera geometry class instance containing intrinsic calibration of the camera sensor.
- **accumulator_** – *Accumulator* instance to be used to accumulate events.

inline explicit **MotionCompensator** (const cv::Size &sensorDimensions)

Construct a motion compensator with no known calibration. This assumes that the camera is an ideal pinhole camera sensor (no distortion) with focal length equal to camera sensor width in pixels and central point is the exact geometrical center of the pixel array.

Parameters

sensorDimensions – Camera sensor resolution.

inline float **getConstantDepth** () const

Get currently assumed constant depth value. It is used if no depth measurements are provided.

 See also

setConstantDepth

Returns

Currently used aistance to the scene (depth).

inline void **setConstantDepth** (const float depth)

Set constant depth value that is assumed if no depth measurement is passed using `accept (dv::measurements::Depth)`. By default the constant depth is assumed to be 3.0 meters, which is just a reasonable guess.

Parameters

depth – Distance to the scene (depth).

Throws

`InvalidArgument` – Exception is thrown if a negative depth value is passed.

inline *dv::EventStore* &**operator>>** (*dv::EventStore* &out)

Private Functions

inline *dv::kinematics::LinearTransformerf* **generateTransforms** (const int64_t from, const int64_t to)

Generate a sequence of transformations at a fixed period (`samplingPeriod`) with an additional overhead transform before and after the given interval.

Parameters

- **from** – Start of the interest interval.
- **to** – End of the interest interval.

Returns

Transformer with resampled transformations.

inline *dv::EventStore* **compensateEvents** (const *dv::EventStore* &events, const *dv::kinematics::LinearTransformerf* &transforms, const *dv::kinematics::Transformationf* &target, const float depth)

Apply motion compensation to event store and project all event into the target transformation.

Parameters

- **events** – Input events.
- **transforms** – Transformer containing the fine grained trajectory of the camera motion.
- **target** – Target position of the camera to be projected into.
- **depth** – Scene depth to be assumed for the calculations.

Returns

Motion compensated events at the target camera pose.

inline *dv::EventStore* **generateEventsAt** (const int64_t timestamp)

Generate compensated events at a given timestamp.

Parameters

timestamp – time to compensate events at.

Returns

A motion compensated events at given time point.

inline *dv::Frame* **generateFrameAt** (const int64_t timestamp)

Generate a frame at a given timestamp.

Parameters

timestamp – Time to generate frame at.

Returns

A motion compensated frame at given time point.

Private Members

PixelPredictor **predictor**

dv::kinematics::LinearTransformerf **transformer**

std::unique_ptr<Accumulator> **accumulator**

std::map<int64_t, float> **depths**

float **constantDepth** = 3.f

dv::EventStore **eventBuffer**

int64_t **storageDuration** = 5000000LL

const int64_t **samplingPeriod** = 200LL

MotionCompensator::Info **info**

template<class **MainStreamType**, class ...**AdditionalTypes**>

class **MultiStreamSlicer**

#include </builds/inivation/dv/dv-processing/include/dv-processing/core/multi_stream_slicer.hpp> *MultiStreamSlicer* takes multiple streams of timestamped data, slices data with configured intervals and calls a given callback method on each interval. It is an extension of *StreamSlicer* class that can synchronously slice multiple streams. Each stream has to be named uniquely, the name is carried over to the callback method to identify each stream.

The class relies heavily on templating, so it supports different containers of data, as long as the container is an iterable and each element contains an accessible timestamp in microsecond format.

The slicing is driven by the main stream, which needs to be specified during construction time. The type of the main stream is the first template argument and the name for the main stream is provided as the constructor's first argument.

By default, these types are supported without additional configuration: *dv::EventStore*, *dv::EventPacket*, *dv::TriggerPacket*, *std::vector<dv::Trigger>*, *dv::IMUPacket*, *std::vector<dv::IMU>*, *std::vector<dv::Frame>*. Additional types can be supported by specifying them as additional template parameters.

Template Parameters

- **MainStreamType** – The type of the main stream.
- **AdditionalTypes** – Parameter pack to specify an arbitrary number of additional stream types to be supported.

Public Types

using **InputType** = *std::variant*<*MainType*, *dv::EventStore*, *dv::EventPacket*, *dv::IMUPacket*, *dv::TriggerPacket*, *std::vector*<*dv::Frame*>, *std::vector*<*dv::IMU*>, *std::vector*<*dv::Trigger*>, *AdditionalTypes*...>

Alias for the variant that holds a packet type.

Public Functions

inline explicit **MultiStreamSlicer** (*std::string* mainStreamName)

Initialize the multi-stream slicer, provide the type of the main stream and a name for the main stream. The slicing is performed by applying a typical slicer on the main stream, all other stream follow it. When a window of slicing executes, the slicer extracts according data from all the other streams and calls a registered callback method for data processing.

Main stream is used to evaluate the jobs, but it also waits for the other types of data to arrive. The callbacks are not executed until all data has arrived on all streams.

By default, these types are supported without additional configuration: *dv::EventStore*, *dv::EventPacket*, *dv::TriggerPacket*, *std::vector*<*dv::Trigger*>, *dv::IMUPacket*, *std::vector*<*dv::IMU*>, *std::vector*<*dv::Frame*>. Additional types can be supported by specifying them as additional template parameters.

Parameters

mainStreamName – Name of the main stream.

template<class **DataType**>

inline void **addStream** (const *std::string* &streamName)

Add a stream to the slicer.

Template Parameters

DataType – Data packet type of the stream.

Parameters

streamName – Name for the stream.

template<class **DataType**>

inline void **accept** (const *std::string* &streamName, const *DataType* &data)

Accept incoming data for a stream and evaluate processing jobs. Can be either a packet or a single timestamped element of the stream.

Parameters

- **streamName** – Name of the stream.
- **data** – Incoming data, either a data packet or timestamp data element.

Throws

RuntimeError – Exception is thrown if passed data type does not match the stream data type.

inline int **doEveryTimeInterval** (const *dv::Duration* interval, *std::function*<void(const *dv::TimeWindow*&, const *MapOfVariants*&> callback)

Register a callback to be performed at a given interval. Data is passed as an argument to the method. Callback method passes *TimeWindow* parameter along the data for the callback to be aware of time slicing windows.

Parameters

- **interval** – Interval at which the callback has to be executed.
- **callback** – Callback method that is called at the given interval, receives time window information and sliced data.

Returns

An id that can be used to modify this job.

```
inline int doEveryTimeInterval (const dv::Duration interval, std::function<void(const MapOfVariants&)>
                                callback)
```

Register a callback to be performed at a given interval. Data is passed as an argument to the method.

Parameters

- **interval** – Interval at which the callback has to be executed.
- **callback** – Callback method that is called at the given interval.

Returns

An id that can be used to modify this job.

```
inline int doEveryNumberOfElements (const size_t n, std::function<void(const dv::TimeWindow&, const
                                MapOfVariants&)> callback, const TimeSlicingApproach
                                timeSlicingApproach = TimeSlicingApproach::BACKWARD)
```

Adds a number-of-elements triggered job to the Slicer. A job is defined by its interval and callback function. The slicer calls the callback function every time *n* elements are added to the stream buffer, with the corresponding data. The (cpu) time interval between individual calls to the function depends on the physical event rate as well as the bulk sizes of the incoming data.

Parameter *timeSlicingApproach* - is an enum that defines timing approach for multi-stream slicing by number. The slicing by number happens by slicing the main stream by a given number of elements. Secondary streams are sliced by the time window of the numbered slice, this introduces a problem of gaps between two number slices - the gap values can either be assigned to current or the next slice, this enum allows to control which of the data parts these gap data will be assigned - backwards will assign all gap data from previous slice end time to current slice start time to current, the forwards approach will assign the gap data from current slice end time to next slice start time to the current slice. The forwards slice timing will result in processing delay of exactly one slice, as it requires to wait for the next slice to happen to correctly retrieve next slice start time. Backwards slicing does not wait for any additional data and processes everything immediately.

Parameters

- **n** – the interval (in number of elements) in which the callback should be called.
- **callback** – the callback function that gets called on the data every interval.
- **timeSlicingApproach** – Select approach for handling secondary stream gap data.

Returns

A handle to uniquely identify the job.

```
inline int doEveryNumberOfElements (const size_t n, std::function<void(const MapOfVariants&)> callback,
                                const TimeSlicingApproach timeSlicingApproach =
                                TimeSlicingApproach::BACKWARD)
```

Adds a number-of-elements triggered job to the Slicer. A job is defined by its interval and callback function. The slicer calls the callback function every time *n* elements are added to the stream buffer, with the corresponding data. The (cpu) time interval between individual calls to the function depends on the physical event rate as well as the bulk sizes of the incoming data.

Parameter *timeSlicingApproach* - is an enum that defines timing approach for multi-stream slicing by number. The slicing by number happens by slicing the main stream by a given number of elements. Secondary streams are sliced by the time window of the numbered slice, this introduces a problem of gaps between two number slices - the gap values can either be assigned to current or the next slice, this enum allows to control which of the data parts these gap data will be assigned - backwards will assign all gap data from previous slice end time to current slice start time to current, the forwards approach will assign the gap data from current slice end time to next slice start time to the current slice. The forwards slice timing will result in processing delay

of exactly one slice, as it requires to wait for the next slice to happen to correctly retrieve next slice start time. Backwards slicing does not wait for any additional data and processes everything immediately.

Parameters

- **n** – the interval (in number of elements) in which the callback should be called.
- **callback** – the callback function that gets called on the data every interval.
- **timeSlicingApproach** – Select approach for handling secondary stream gap data.

Returns

A handle to uniquely identify the job.

inline void **modifyTimeInterval** (const int jobId, const *dv::Duration* timeInterval)

Modify the execution interval of a job.

Parameters

- **jobId** – Callback id that is received from callback registration.
- **timeInterval** – New time interval to be executed.

Throws

invalid_argument – Exception is thrown if trying to modify a number based slicing job.

inline void **modifyNumberInterval** (const int jobId, const size_t n)

Modify the execution number of elements of a job.

Parameters

- **jobId** – Job id that is received from callback registration.
- **n** – New number of elements to slice for the given job id.

Throws

invalid_argument – Exception is thrown if trying to modify a time based slicing job.

inline bool **hasJob** (const int jobId) const

Returns true if the slicer contains the slice-job with the provided id

Parameters

jobId – the id of the slice-job in question

Returns

true, if the slicer contains the given slice-job

inline void **removeJob** (const int jobId)

Removes the given job from the list of current jobs.

Parameters

jobId – The job id to be removed

inline void **setStreamSeekTime** (const *std::string* &streamName, const int64_t seekTimestamp)

Update a stream's seek time manually and evaluate jobs.

Data synchronization is automatically inferred from received data. This works well with data streams that produce data at guaranteed periodic intervals. For aperiodic data streams, which produce data spontaneously, a manual synchronization is required. This method allows to manually instruct the slicer that the given stream has provided data up to, but not including, this given seek timestamp; even in case when there was no data. Slicer is then able to progress other streams until the given time, since it assumes no data will ever arrive for this stream until this point. Be sure to call this method when you are sure no data will arrive, otherwise that data can be lost.

Parameters

- **streamName** – Name of the stream.
- **seekTimestamp** – Seek time for this stream; all data until this time has been provided to the slicer.

Protected Attributes

`int64_t mMainBufferSeekTime = -1`

Main buffer seek time, this is the timestamp of last fed data into main slicer.

`std::map<int, SliceJob> mSliceJobs`

Storage container for configured slice jobs.

`int32_t mHashCounter = 0`

`std::map<int32_t, int32_t> mMapFromSliceJobIdsToMainSlicerIds`

Map for determining mapping from multi stream slicer job ids to main stream slicer job ids, we use this since it is not known a priori how job ids are set for the main stream slicer

`std::map<std::string, InputType> mBuffer`

Buffered data that is in queue for slicing.

`std::map<std::string, int64_t> mLastReceivedBufferTimestamps`

Placeholder for manually provided seek timestamp of stream seek times.

`std::string mMainStreamName`

Name of the main stream.

`dv::StreamSlicer<MainStreamType> mMainSlicer`

Slicer for the main stream, all other streams follow the main stream slicer.

Private Types

using **MainType** = typename `std::conditional_t<dv::concepts::is_type_one_of<MainStreamType, dv::EventStore, dv::EventPacket, dv::IMUPacket, dv::TriggerPacket, std::vector<dv::Frame>, std::vector<dv::IMU>, std::vector<dv::Trigger>, AdditionalTypes...>, std::monostate, MainStreamType>`

Private Functions

`inline int64_t getMinLastBufferTimestamps ()`

Get the minimum value of the last received buffer timestamps.

Returns

minimum last received buffer timestamp.

`inline int64_t getMinEvaluatedJobTime ()`

Get the minimum of the last evaluated job times. This is helpful for determining which data to remove from the internal buffer as any data before this minimum value is no longer needed and can, therefore, be discarded

Returns

minimum of the last evaluated job times.

inline void **evaluate** ()

Evaluate the current state of the slicer. Performs data book-keeping and executes the callback methods.

Private Static Functions

template<class **VectorType**>

static inline *VectorType* **sliceVector** (const int64_t start, const int64_t end, const *VectorType* &packet)

Slice a vector type within given time bounds [start, end). Start time is inclusive, end time is exclusive.

Template Parameters

VectorType –

Parameters

- **start** – Start timestamp
- **end** – End timestamp
- **packet** – Packet of a vector type

Returns

Copy of the data within the bounds

template<class **PacketType**>

static inline *PacketType* **slicePacketSpecific** (const int64_t start, const int64_t end, const *PacketType* &packet)

Templated method for packet slicing. Returns the data slice between given timestamps. Start time is inclusive, end time is exclusive.

Template Parameters

PacketType –

Parameters

- **start** – Start timestamp
- **end** – End timestamp
- **packet** – Packet of data

Returns

Copy of the data within the bounds

static inline *InputType* **slicePacket** (const int64_t start, const int64_t end, const *InputType* &packet)

Templated method for packet contained in a variant. Returns the data slice between given timestamps. Start time is inclusive, end time is exclusive.

Parameters

- **start** – Start of time range.
- **end** – End of time range.
- **packet** – Input data packet.

Returns

Sliced data from the packet according to given time ranges.

template<class **PacketType**>

static inline void **mergePackets** (const *PacketType* &from, *PacketType* &into)

Merge successive packets, this copies data from one to another. Performs shallow copy if possible.

Template Parameters

PacketType –

Parameters

- **from** – Source packet
- **into** – Destination packet

template<class **PacketType**>

static inline void **eraseUpToIterable** (const int64_t timeLimit, *PacketType* &packet)

Erase data within the packet up to the given time point. Specific implementation for vector containers.

Template Parameters

PacketType –

Parameters

- **timeLimit** – Timestamp to delete until, this is exclusive
- **packet** – Packet to modify

template<class **PacketType**>

static inline void **eraseUpTo** (const int64_t timeLimit, *PacketType* &packet)

Erase data within the packet up to the given time point.

Template Parameters

PacketType –

Parameters

- **timeLimit** – Timestamp to delete until, this is exclusive
- **packet** – Packet to modify

template<class **PacketType**>

static inline *dv::TimeWindow* **getPacketTimeWindow** (const *PacketType* &packet)

Retrieve highest and lowest timestamps of a given packet

Template Parameters

PacketType –

Parameters

packet –

Returns

Time window containing start and end timestamps.

template<class **PacketType**>

static inline bool **isPacketEmpty** (const *PacketType* &packet)

Check if a packet is empty.

Template Parameters

PacketType –

Parameters

packet –

Returns

True if the given packet is empty, false otherwise.

class **NetworkReader** : public *dv::io::InputBase*

#include </builds/inivation/dv/dv-processing/include/dv-processing/io/network_reader.hpp> Network capture class. Connect to a TCP or a local socket server providing a data stream. The class provides a single data stream per network capture.

Public Functions

inline **NetworkReader** (const *std::string_view* ipAddress, const uint16_t port)

Initialize a network capture object, it will connect to a given TCP port with given IP address.

Parameters

- **ipAddress** – IP address of the target TCP server.
- **port** – TCP port number.

inline **NetworkReader** (const *std::string_view* ipAddress, const uint16_t port, boost::asio::ssl::context &&encryptionContext)

Initialize an encrypted network capture object, it will connect to a given TCP port with given IP address. Provide an encryption context that is preconfigured, prefer using existing *dv::io::encrypt::defaultEncryptionClient()* method for configuring the encryption context.

Parameters

- **ipAddress** – IP address of the target TCP server.
- **port** – TCP port number.
- **encryptionContext** – Preconfigured encryption context.

inline explicit **NetworkReader** (const *std::filesystem::path* &socketPath)

Initialize a network capture object, it will connect to a given UNIX socket with a given file system path.

Parameters

socketPath – Path to the UNIX socket.

inline **~NetworkReader** () override

Destructor - disconnects from network resource, stops threads and frees any buffered data.

inline virtual *std::optional<dv::EventStore>* **getNextEventBatch** () override

Read next event batch. This is a non-blocking method, if there is no data to read, it will return a *std::nullopt*.

Returns

Next batch of events, *std::nullopt* if no data received from last read or the event stream is not available.

inline virtual *std::optional<dv::Frame>* **getNextFrame** () override

Read next frame. This is a non-blocking method, if there is no data to read, it will return a *std::nullopt*.

Returns

Next frame, *std::nullopt* if no data received from last read or the event stream is not available.

inline virtual *std::optional<std::vector<dv::IMU>>* **getNextImuBatch** () override

Read next imu measurement batch. This is a non-blocking method, if there is no data to read, it will return a *std::nullopt*.

Returns

Next batch of imu measurements, `std::nullopt` if no data received from last read or the event stream is not available.

inline virtual `std::optional<std::vector<dv::Trigger>>` **getNextTriggerBatch()** override

Read next trigger batch. This is a non-blocking method, if there is no data to read, it will return a `std::nullopt`.

Returns

Next batch of triggers, `std::nullopt` if no data received from last read or the event stream is not available.

inline virtual `std::optional<cv::Size>` **getEventResolution()** const override

Retrieve the event sensor resolution. The method returns `std::nullopt` if event stream is not available or the metadata does not contain resolution.

Returns

Event sensor resolution or `std::nullopt` if not available.

inline virtual `std::optional<cv::Size>` **getFrameResolution()** const override

Retrieve the frame sensor resolution. The method returns `std::nullopt` if frame stream is not available or the metadata does not contain resolution.

Returns

Frame sensor resolution or `std::nullopt` if not available.

template<class **PacketType**>

inline `std::shared_ptr<PacketType>` **getNextPacket()**

Read next packet, given it's type.

The given type must match the stream type exactly (it must be a flatbuffer generated type). Returns `nullptr` if no data is available for reading or stream of such type is not available.

Template Parameters

PacketType – *Stream* packet type, must be a flatbuffer type and must match stream type exactly.

Returns

Shared pointer to a packet of data, or `nullptr` if unavailable.

inline virtual bool **isEventStreamAvailable()** const override

Check whether an event stream is available in this capture class.

Returns

True if an event stream is available; false otherwise.

inline virtual bool **isFrameStreamAvailable()** const override

Check whether a frame stream is available in this capture class.

Returns

True if a frame stream is available; false otherwise.

inline virtual bool **isImuStreamAvailable()** const override

Check whether an *IMU* data stream is available in this capture class.

Returns

True if an *IMU* data stream is available; false otherwise.

inline virtual bool **isTriggerStreamAvailable()** const override

Check whether a trigger stream is available in this capture class.

Returns

True if a trigger stream is available; false otherwise.

inline virtual `std::string getCameraName ()` const override

Get camera name, which is a combination of the camera model and the serial number.

Returns

String containing the camera model and serial number separated by an underscore character.

inline virtual bool `isRunning ()` const override

Check whether any input data streams have terminated. For a live camera this should check if the device is still connected and functioning, while for a recording file this should check if any of the data streams have reached end-of-file (EOF). For a network input, this indicates the network stream is still connected.

Returns

True if data read on all streams is still possible, false otherwise.

inline virtual bool `isRunning (const std::string_view streamName)` const override

Check whether the input data stream with the specified name is still active.

Returns

True if data read on this stream is possible, false otherwise.

inline virtual bool `isRunningAny ()` const override

Check whether any input data streams are still available. For a live camera this should check if the device is still connected and functioning and at least one data stream is active (different than `isRunning()`), while for a recording file this should check if any of the data streams have not yet reached end-of-file (EOF) and are still readable. For a network input, this indicates the network stream is still connected.

Returns

True if data read on at least one stream is still possible, false otherwise.

template<class **PacketType**>

inline bool `isStreamAvailable ()` const

Check whether a stream of given type is available.

The given type must match the stream type exactly (it must be a flatbuffer generated type). Returns `nullptr` if no data is available for reading or stream of such type is not available.

Template Parameters

PacketType – *Stream* packet type, must be a flatbuffer type and must match stream type exactly.

Returns

True if stream of a given type is available, false otherwise.

inline virtual bool `isStreamAvailable (const std::string_view streamName)` const override

Check whether a stream with given name is available.

Returns

True if data stream is available, false otherwise.

inline void `close ()`

Explicitly close the communication socket, receiving data is not going to be possible after this method call.

inline const `dv::io::Stream &getStreamDefinition ()` const

Get the stream definition object, which describes the available data stream by this reader.

Returns

Data stream definition object.

Private Types

using **PacketQueue** = boost::lockfree::spsc_queue<dv::types::TypedObject*>

Private Functions

inline void **readClbk** (std::vector<std::byte> &data, const int64_t)

Read block of data from the network socket.

Parameters

data – Container for data that is going to be read.

inline void **connectTCP** (const std::string_view ipAddress, const uint16_t port, const bool tlsEnabled = false)

Initiate connection to the given IP address and port.

Parameters

- **ipAddress** – Ip address, dot separated (in format “0.0.0.0”)
- **port** – TCP port number
- **tlsEnabled** – Enable TLS encryption

inline void **connectUNIX** (const std::filesystem::path &socketPath)

Initiate a connection to UNIX socket under given filesystem path.

Parameters

socketPath – Path to a socket.

inline void **readThread** ()

inline void **initializeReader** ()

Private Members

std::function<void(std::vector<std::byte>&, const int64_t)> **mReadHandler** =
std::bind_front(&NetworkReader::readClbk, this)

Callback method that calls read method of the socket.

boost::asio::io_context **mIOService**

IO service context.

std::unique_ptr<network::SocketBase> **mSocket** = nullptr

Socket to contain the connection instance.

asioSSL::context **mTLSContext** = asioSSL::context(asioSSL::context::method::tlsv12_client)

Decryption context.

bool **mTLSEnabled**

Whether TLS encryption is enabled.

dv::io::Reader **mAedat4Reader**

AEDAT4 reader.

dv::io::Stream **mStream**

Data stream container - one per capture.

std::string **mCameraName**

Name of the camera producing the stream.

PacketQueue **mPacketQueue** = *PacketQueue*(1000)

Incoming packet queue.

std::thread **mReadingThread**

Reading thread.

std::atomic<bool> **mKeepReading** = true

Atomic bool used to stop the reading thread.

std::atomic<bool> **mExceptionThrown** = false

Boolean value that indicated whether an exception was thrown on reading thread.

std::exception_ptr **mException** = nullptr

Pointer that holds thrown exception, mExceptionThrown contains thread-safe flag indicating an exception was thrown

class **NetworkWriter** : public *dv::io::OutputBase*

#include </builds/inivation/dv/dv-processing/include/dv-processing/io/network_writer.hpp> Network server class for streaming AEDAT4 serialized data types.

Public Types

using **ErrorMessageCallback** = *std::function*<void(const boost::system::error_code&, const *std::string_view*)>

Public Functions

```
inline NetworkWriter (const std::string_view ipAddress, const uint16_t port, const dv::io::Stream &stream,
    const size_t maxClientConnections = 10, ErrorMessageCallback messageCallback =
    [] (const boost::system::error_code &, const std::string_view) {
    })
```

Create a non-encrypted server that listens for connections on a given IP address. Supports multiple clients.

Parameters

- **ipAddress** – IP address to bind the server.
- **port** – Port number.
- **stream** – AEDAT4 stream definition.
- **maxClientConnections** – Maximum number of client connections supported by this instance.
- **messageCallback** – Callback to handle any error messages received by the client connections.

```
inline NetworkWriter (const std::string_view ipAddress, const uint16_t port, const dv::io::Stream &stream,
    boost::asio::ssl::context &&encryptionContext, const size_t maxClientConnections =
    10, ErrorMessageCallback messageCallback = [] (const
    boost::system::error_code &, const std::string_view) { })
```

Create an encrypted server that listens for connections on a given IP address. Supports multiple clients.

Parameters

- **ipAddress** – IP address to bind the server.
- **port** – Port number.
- **stream** – AEDAT4 stream definition.
- **encryptionContext** – Preconfigured encryption context, use either *dv::io::encrypt::defaultEncryptionServer()* to create the context or configure custom encryption context. When a client connects to the server, it will run handshake, during which client certificates will be validated, if the handshake fails, connection is terminated.
- **maxClientConnections** – Maximum number of client connections supported by this instance.
- **messageCallback** – Callback to handle any error messages received by the client connections.

```
inline NetworkWriter (const std::filesystem::path &socketPath, const dv::io::Stream &stream, const size_t
    maxClientConnections = 10, ErrorMessageCallback messageCallback = [] (const
    boost::system::error_code &, const std::string_view) { })
```

Create a local socket server. Provide a path to the socket, if a file already exists on a given path, the connection will fail by throwing an exception. It is required that the given socket path does not point to an existing socket file. If the file *can* exist, it is up to the user of this class to decide whether it is safe to remove any existing socket files or the class should not bind to the path.

Parameters

- **socketPath** – Path to a socket file, must be a non-existent path.
- **stream** – AEDAT4 stream definition.
- **maxClientConnections** – Maximum number of client connections supported by this instance.
- **messageCallback** – Callback to handle any error messages received by the client connections.

```
inline ~NetworkWriter () override
```

Closes the socket, frees allocated memory, and removes any queued packets from write queue.

```
inline virtual void writeEvents (const EventStore &events) override
```

Write an event store to the network stream.

Parameters

events – Data to be sent out.

```
inline virtual void writeFrame (const dv::Frame &frame) override
```

Write a frame image to the network stream.

Parameters

frame – Data to be sent out.

inline virtual void **writeImu** (const *std::vector<dv::IMU>* &imu) override

Write *IMU* data to the socket.

Parameters

imu – Data to be sent out.

inline virtual void **writeTriggers** (const *std::vector<dv::Trigger>* &triggers) override

Write trigger data to the network stream.

Parameters

triggers – Data to be sent out.

template<class **PacketType**>

inline void **writePacket** (*PacketType* &&packet)

Write a flatbuffer packet to the network stream.

Template Parameters

PacketType – Type of the packet, must satisfy the *dv::concepts::FlatbufferPacket* concept.

Parameters

packet – Data to write.

inline virtual *std::string* **getCameraName** () const override

Get camera name. It is looked up from the stream definition during construction.

Returns

inline size_t **getQueuedPacketCount** () const

Get number of packets in the write queue.

Returns

Number of packets in the write queue.

inline size_t **getClientCount** () const

Get number of active connected clients.

Returns

Number of active connected clients.

Private Types

using **WriteQueue** = boost::lockfree::spsc_queue<*std::shared_ptr<dv::types::TypedObject>*>

Private Functions

template<class **SocketType**>

inline void **acceptStart** ()

inline void **writePacketToClients** (const *std::shared_ptr<dv::types::TypedObject>* &packet)

inline void **ioThread** ()

inline void **connectTCP** (const *std::string_view* ipAddress, const uint16_t port)

inline void **connectUNIX** (const *std::filesystem::path* &socketPath)

inline void **generateHeaderContent** (const *dv::io::Stream* &stream)

inline void **removeClient** (const *Connection* *const client)

Private Members

std::string **mCameraName**

size_t **mMaxConnections**

asio::io_context **mIoService**

std::unique_ptr<asioTCP::acceptor> **mAcceptorTcp** = nullptr

std::unique_ptr<asioUNIX::acceptor> **mAcceptorUnix** = nullptr

asioSSL::context **mTLSContext** = asioSSL::context(asioSSL::context::method::tlsv12_server)

bool **mTLSEnabled**

mutable std::mutex **mClientsMutex**

std::vector<Connection>* **mClients**

The client list is raw point, that is self-owned, read Connection class documentation for more details.

std::atomic<size_t> **mQueuedPackets** = 0

dv::io::Writer **mAedat4Writer**

std::string **mInfoNode**

std::atomic<bool> **mShutdownRequested** = false

std::thread **mIOThread**

int32_t **mStreamId** = 0

std::filesystem::path **mSocketPath**

WriteQueue **mWriteQueue** = *WriteQueue*(1024)

ErrorMessageCallback **mErrorMessageHandler**

Error message handler, by default: NOOP.

class **NoneCompressionSupport** : public *dv::io::compression::CompressionSupport*

Public Functions

inline explicit **NoneCompressionSupport** (const *CompressionType* type)

inline virtual void **compress** (*dv::io::support::IODataBuffer* &packet) override

class **NoneDecompressionSupport** : public *dv::io::compression::DecompressionSupport*

Public Functions

inline explicit **NoneDecompressionSupport** (const *CompressionType* type)

inline virtual void **decompress** (*std::vector<std::byte>* &source, *std::vector<std::byte>* &target) override

class **NoRedetection** : public *dv::features::RedetectionStrategy*

#include </builds/inivation/dv/dv-processing/include/dv-processing/features/redetection_strategy.hpp> No redetection strategy.

Public Functions

inline virtual bool **decideRedetection** (const *TrackerBase*&) override

Do not perform redetection.

Returns

Just return false always.

struct **NullPointer** : public *dv::exceptions::info::EmptyException*

struct **Observation** : public *flatbuffers::NativeTable*

Public Types

typedef *ObservationFlatbuffer* **TableType**

Public Functions

inline **Observation** ()

inline **Observation** (int32_t _trackId, int32_t _cameraId, const *std::string* &_cameraName, int64_t _timestamp)

Public Members

int32_t **trackId**

int32_t **cameraId**

std::string **cameraName**

int64_t **timestamp**

Public Static Functions

```
static inline constexpr const char *GetFullyQualifiedName ()
```

```
struct ObservationBuilder
```

Public Functions

```
inline void add_trackId (int32_t trackId)
```

```
inline void add_cameraId (int32_t cameraId)
```

```
inline void add_cameraName (flatbuffers::Offset<flatbuffers::String> cameraName)
```

```
inline void add_timestamp (int64_t timestamp)
```

```
inline explicit ObservationBuilder (flatbuffers::FlatBufferBuilder &_fbb)
```

```
ObservationBuilder &operator= (const ObservationBuilder&)
```

```
inline flatbuffers::Offset<ObservationFlatbuffer> Finish ()
```

Public Members

```
flatbuffers::FlatBufferBuilder &fbb_
```

```
flatbuffers::uoffset_t start_
```

```
struct ObservationFlatbuffer : private flatbuffers::Table
```

Public Types

```
typedef Observation NativeTableType
```

Public Functions

```
inline int32_t trackId () const
```

The tracking sequence ID that the landmark is observed by a camera.

```
inline int32_t cameraId () const
```

Arbitrary ID of the camera, this can be application specific.

```
inline const flatbuffers::String *cameraName () const
```

Name of the camera. Optional.

```
inline int64_t timestamp () const
```

Timestamp of the observation (µs).

```
inline bool Verify (flatbuffers::Verifier &verifier) const
```

```
inline Observation *UnPack (const flatbuffers::resolver_function_t *_resolver = nullptr) const
```

```
inline void UnPackTo (Observation *_o, const flatbuffers::resolver_function_t *_resolver = nullptr) const
```

Public Static Functions

```
static inline const flatbuffers::TypeTable *MiniReflectTypeTable ()

static inline constexpr const char *GetFullyQualifiedName ()

static inline void UnPackToFrom (Observation *_o, const ObservationFlatbuffer *_fb, const
                                flatbuffers::resolver_function_t *_resolver = nullptr)

static inline flatbuffers::Offset<ObservationFlatbuffer> Pack (flatbuffers::FlatBufferBuilder &_fbb, const
                                                            Observation *_o, const
                                                            flatbuffers::rehasher_function_t *_rehasher =
                                                            nullptr)
```

```
template<typename _Scalar, int NX = Eigen::Dynamic, int NY = Eigen::Dynamic>
```

```
class OptimizationFunctor
```

#include </builds/inivation/dv/dv-processing/include/dv-processing/optimization/optimization_functor.hpp> Basic functor class inherited by all contrastMaximization functor. This functor is used by Eigen/NumericalDiff class, which handles the non linear optimization underlying contrast maximization algorithm. For more information about contrast maximization please check “contrast_maximization_rotation.hpp” or “contrast_maximization_translation_and_depth.hpp”.

Template Parameters

- **_Scalar** – type of variable to optimize (e.g. int, float..).
- **NX** – Number of input variables (note: all variables are stored as Nx1 vector of values)
- **NY** – Number of output measurements (note: number of measurements needs to be at least as big as number of input variables - NX - otherwise the optimization problem cannot be solved.)

Public Types

Values:

```
enumerator InputsAtCompileTime
```

```
enumerator ValuesAtCompileTime
```

```
typedef _Scalar Scalar
```

```
typedef Eigen::Matrix<Scalar, InputsAtCompileTime, 1> InputType
```

```
typedef Eigen::Matrix<Scalar, ValuesAtCompileTime, 1> ValueType
```

```
typedef Eigen::Matrix<Scalar, ValuesAtCompileTime, InputsAtCompileTime> JacobianType
```

Public Functions

virtual int **operator ()** (const Eigen::VectorXf &input, Eigen::VectorXf &cost) const = 0

Base method for cost function implementation.

Parameters

- **input** – parameters to be optimized
- **cost** – cost value updated at each iteration of the optimization.

Returns

optimization result (positive if successful)

inline **OptimizationFunctor** (int inputs, int values)

Constructor for cost optimization parameters

Parameters

- **inputs** – number of inputs to be optimized
- **values** – number of functions evaluation for gradient computation

inline int **inputs** () const

getter for size of input parameters to be optimized.

Returns

number of input parameters optimized.

inline int **values** () const

getter for size of function evaluations performed at each optimization iteration.

Returns

number of function evaluations at each optimization iteration.

Private Members

int **mInputs**

int **mValues**

struct **optimizationOutput**

Public Members

int **optimizationSuccessful**

int **iter**

Eigen::VectorXf **optimizedVariable**

struct **optimizationParameters**

Public Members

float **learningRate** = float(1e-1)

float **epsfcn** = 0

float **ftol** = 0.000345267

float **gtol** = 0

float **xtol** = 0.000345267

int **maxfev** = 400

struct **OutOfRange** : public *dv::exceptions::info::EmptyException*

class **OutputBase**

#include </builds/inivation/dv/dv-processing/include/dv-processing/io/output_base.hpp> Output reader base class defining API interface for writing camera data into an IO resource.

Subclassed by *dv::io::MonoCameraWriter*, *dv::io::NetworkWriter*

Public Functions

virtual **~OutputBase** () = default

virtual void **writeEvents** (const *dv::EventStore* &events) = 0

Write event data into the output.

Parameters

events – Write events into the output.

virtual void **writeFrame** (const *dv::Frame* &frame) = 0

Write a frame into the output.

Parameters

frame – Write a frame into the output.

virtual void **writeImu** (const *std::vector<dv::IMU>* &imu) = 0

Write imu data into the output.

Parameters

imu – Write imu into the output.

virtual void **writeTriggers** (const *std::vector<dv::Trigger>* &triggers) = 0

Write trigger data into the output.

Parameters

triggers – Write trigger into the output.

virtual *std::string* **getCameraName** () const = 0

Retrieve camera name of this writer output instance.

Returns

Configured camera name.

struct **OutputError**

Public Types

using **Info** = *ErrorInfo*

Public Static Functions

static inline *std::string* **format** (const *Info* &info)

struct **ParsedData**

Public Functions

inline void **clear** ()

Public Members

dv::EventPacket **events** = {}

std::vector<dv::Frame> **frames** = {}

dv::IMUPacket **imu** = {}

dv::TriggerPacket **triggers** = {}

class **Parser** : public *dv::io::camera::parser::ParserBase*

Public Functions

inline **Parser** (const *SensorModel* sensorModel, const cv::Size dvsResolutionDevice, const cv::Size
apsResolutionDevice, const *dv::io::camera::imu::ImuModel* imuModel, const uint32_t
dvsOrientation, const uint32_t apsOrientation, const uint32_t imuOrientation, const
dv::PixelArrangement apsColorFilter)

inline bool **getDVSFlipHorizontal** () const

inline void **setDVSFlipHorizontal** (const bool flip_x)

inline bool **getDVSFlipVertical** () const

inline void **setDVSFlipVertical** (const bool flip_y)

inline bool **getDVSConvertAllOn** () const

inline void **setDVSConvertAllOn** (const bool convertAllOn)

```

inline bool getFrameFlipHorizontal () const

inline void setFrameFlipHorizontal (const bool flip_x)

inline bool getFrameFlipVertical () const

inline void setFrameFlipVertical (const bool flip_y)

inline dv::PixelArrangement getFrameColorFilter () const

inline ColorMode getFrameColorMode () const

inline void setFrameColorMode (const ColorMode mode)

inline bool getIMUFlipX () const

inline void setIMUFlipX (const bool flip_x)

inline bool getIMUFlipY () const

inline void setIMUFlipY (const bool flip_y)

inline bool getIMUFlipZ () const

inline void setIMUFlipZ (const bool flip_z)

inline virtual void parseData (const std::span<const uint8_t> buffer, const bool dataLost) override

```

Public Members

```

int64_t wrapAdd = {0}

cv::Size sizeDevice

bool invertXY = {false}

std::atomic<bool> flipXControl = {false}

std::atomic<bool> flipYControl = {false}

std::atomic<bool> convertAllOn = {false}

uint16_t lastY = {0}

cv::Size sizeUser

std::atomic<dv::PixelArrangement> colorFilterControl = {dv::PixelArrangement::MONO}

std::atomic<ColorMode> colorMode = {ColorMode::DEFAULT}

bool flipXSync = {false}

```

```
bool flipYSync = {false}

bool flipX = {false}

bool flipY = {false}

bool globalShutter = {true}

dv::PixelArrangement colorFilterSync = {dv::PixelArrangement::MONO}

ApsReadoutType currentReadoutType = {ApsReadoutType::RESET}

std::array<uint16_t, 2> countX = {}

std::array<uint16_t, 2> countY = {}

uint16_t expectedCountX = {0}

uint16_t expectedCountY = {0}

int64_t startOfExposureTimestamp = {0}

cv::Mat pixels

uint16_t tmpData = {0}

uint16_t update = {0}

uint16_t positionX = {0}

uint16_t positionY = {0}

uint16_t sizeX = {0}

uint16_t sizeY = {0}

struct dv::io::camera::parser::DAVIS::Parser roi

uint32_t currentFrameExposure = {0}

uint8_t tmpData = {0}
```

```
struct dv::io::camera::parser::DAVIS::Parser exposure

bool startPositionOdd = {false}

CDavisOffset direction = {CDavisOffset::INCREASING}

struct dv::io::camera::parser::DAVIS::Parser cDavisSupport

dv::io::camera::imu::ImuModel model

bool flipZ = {false}

std::atomic<bool> flipZControl = {false}

uint8_t type = {0}

uint8_t count = {0}

float accelScale = {}

float gyroScale = {}

dv::IMU currentEvent = {}
```

Private Types

```
enum class ApsReadoutType : bool
```

Values:

```
enumerator RESET
```

```
enumerator SIGNAL
```

```
enum class CDavisOffset : bool
```

Values:

```
enumerator INCREASING
```

```
enumerator DECREASING
```

Private Functions

```
inline void apsInitFrame ()  
  
inline void apsROIUpdateSizes ()  
  
inline void apsUpdateFrame (const uint8_t dataValue)  
  
inline void cdavisUpdateFrame (const uint8_t dataValue)  
  
inline bool apsEndFrame ()
```

Private Members

```
struct dv::io::camera::parser::DAVIS::Parser mTimestamps  
  
struct dv::io::camera::parser::DAVIS::Parser mDvs  
  
struct dv::io::camera::parser::DAVIS::Parser mFrame  
  
struct dv::io::camera::parser::DAVIS::Parser mImu  
  
SensorModel mModel
```

Private Static Functions

```
static inline float calculateIMUAccelScale (const uint8_t imuAccelScale)  
  
static inline float calculateIMUGyroScale (const uint8_t imuGyroScale)  
  
static inline void frameFixCDavis (dv::Frame &frame, const dv::PixelArrangement colorFilter)  
  
static inline void frameDebayer (dv::Frame &frame, const ColorMode colorMode, const dv::PixelArrangement  
                                colorFilter)  
  
static inline void framesSplitIntoQuadrants (dv::Frame &frame)
```

Private Static Attributes

```
static constexpr uint16_t TS_WRAP_ADD = {0x8000}  
  
static constexpr uint8_t IMU_TYPE_TEMP = {0x01}  
  
static constexpr uint8_t IMU_TYPE_GYRO = {0x02}  
  
static constexpr uint8_t IMU_TYPE_ACCEL = {0x04}  
  
static constexpr uint8_t IMU_TOTAL_COUNT = {14}
```

```
static constexpr uint8_t APS_RESET_CUTOFF = {96}
```

```
class Parser : public dv::io::camera::parser::ParserBase
```

Public Functions

```
Parser () = default
```

```
inline bool getDVSFlipHorizontal () const
```

```
inline void setDVSFlipHorizontal (const bool flip_x)
```

```
inline bool getDVSFlipVertical () const
```

```
inline void setDVSFlipVertical (const bool flip_y)
```

```
inline bool getDVSConvertAllOn () const
```

```
inline void setDVSConvertAllOn (const bool convertAllOn)
```

```
inline virtual void parseData (const std::span<const uint8_t> buffer, const bool dataLost) override
```

Public Members

```
int64_t wrapAdd = {0}
```

```
std::atomic<bool> flipX = {true}
```

```
std::atomic<bool> flipY = {true}
```

```
std::atomic<bool> convertAllOn = {false}
```

Private Members

```
struct dv::io::camera::parser::DVS128::Parser mTimestamps
```

```
struct dv::io::camera::parser::DVS128::Parser mDvs
```

Private Static Attributes

```
static constexpr uint8_t DVS128_TIMESTAMP_WRAP_MASK = {0x80}
```

```
static constexpr uint8_t DVS128_TIMESTAMP_RESET_MASK = {0x40}
```

```
static constexpr uint8_t DVS128_POLARITY_SHIFT = {0}
```

```
static constexpr uint16_t DVS128_POLARITY_MASK = {0x0001}
```

```
static constexpr uint8_t DVS128_Y_ADDR_SHIFT = {8}
```

```
static constexpr uint16_t DVS128_Y_ADDR_MASK = {0x007F}
```

```
static constexpr uint8_t DVS128_X_ADDR_SHIFT = {1}
```

```
static constexpr uint16_t DVS128_X_ADDR_MASK = {0x007F}
```

```
static constexpr uint16_t DVS128_SYNC_EVENT_MASK = {0x8000}
```

```
static constexpr uint16_t TS_WRAP_ADD = {0x4000}
```

```
class Parser : public dv::io::camera::parser::ParserBase
```

Public Functions

```
inline explicit Parser (const cv::Size eventsResolution, const uint32_t dvsOrientation, const uint32_t imuOrientation)
```

```
inline bool getDVSFlipHorizontal () const
```

```
inline void setDVSFlipHorizontal (const bool flip_x)
```

```
inline bool getDVSFlipVertical () const
```

```
inline void setDVSFlipVertical (const bool flip_y)
```

```
inline bool getDVSDualBinning () const
```

```
inline void setDVSDualBinning (const bool dual_binning)
```

```
inline bool getIMUFlipX () const
```

```
inline void setIMUFlipX (const bool flip_x)
```

```
inline bool getIMUFlipY () const
```

```
inline void setIMUFlipY (const bool flip_y)
```

```
inline bool getIMUFlipZ () const
```

```
inline void setIMUFlipZ (const bool flip_z)
```

```
inline virtual void parseData (const std::span<const uint8_t> buffer, const bool dataLost) override
```

Public Members

```
int64_t wrapAdd = {0}
```

```
cv::Size size
```

```
bool flipX = {false}

bool flipY = {false}

std::atomic<bool> flipXControl = {false}

std::atomic<bool> flipYControl = {false}

std::atomic<bool> dualBinning = {false}

uint16_t lastX = {0}

uint16_t lastYG1 = {0}

uint16_t lastYG2 = {0}

bool flipZ = {false}

std::atomic<bool> flipZControl = {false}

uint8_t type = {0}

uint8_t count = {0}

uint8_t tmpData = {0}

float accelScale = {}

float gyroScale = {}

dv::IMU currentEvent = {}
```

Private Members

```
struct dv::io::camera::parser::DVXplorer::Parser mTimestamps

struct dv::io::camera::parser::DVXplorer::Parser mDvs

struct dv::io::camera::parser::DVXplorer::Parser mImu
```

Private Static Functions

```
static inline float calculateIMUAccelScale (const uint8_t imuAccelScale)
```

```
static inline float calculateIMUGyroScale (const uint8_t imuGyroScale)
```

Private Static Attributes

```
static constexpr uint16_t TS_WRAP_ADD = {0x8000}
```

```
static constexpr uint8_t IMU_TYPE_TEMP = {0x01}
```

```
static constexpr uint8_t IMU_TYPE_GYRO = {0x02}
```

```
static constexpr uint8_t IMU_TYPE_ACCEL = {0x04}
```

```
static constexpr uint8_t IMU_TOTAL_COUNT = {14}
```

```
class Parser : public dv::io::camera::parser::ParserBase
```

Public Functions

```
Parser () = default
```

```
inline void setFlipHorizontal (const bool flip)
```

```
inline bool getFlipHorizontal () const
```

```
inline void setFlipVertical (const bool flip)
```

```
inline bool getFlipVertical () const
```

```
inline virtual void parseData (const std::span<const uint8_t> buffer, const bool dataLost) override
```

```
inline void injectImu (dv::IMU imuData)
```

Public Members

```
bool flipHorizontal = {false}
```

```
bool flipVertical = {false}
```

```
int16_t columnAddress = {-1}
```

```
std::array<bool, NUM_GROUPS> groupsUsed = {}
```

```
std::array<std::array<uint8_t, EVENT_GROUPS_FAST>, 2> columnEvents = {}
```

```
size_t resetIndex = {0}
```

```

size_t lastCommittedEvents = {0}

int64_t reference = {-1}

int64_t referenceOverflow = {0}

int32_t lastReference = {-1}

int32_t lastUsedSub = {-1}

int64_t lastUsedReference = {-1}

std::atomic<bool> flipHorizontal = {false}

std::atomic<bool> flipVertical = {false}

```

Private Types

enum class **State**

Values:

```

enumerator WAIT_FOR_TIMESTAMP_REFERENCE

enumerator WAIT_FOR_START_OF_FRAME

enumerator WAIT_FOR_SMGROUP

enumerator WAIT_FOR_COLUMN_OR_SMGROUP

```

Private Functions

```

inline void aUpdateTimestampReference (const eTimestampReference timestampRef)

inline void aFrameStart (const eColumn column)

inline void aSetColumnData (const eColumn column)

inline void bitIndexToOutput (std::vector<dv::Event> &dst, const size_t *src, const bool polarity) const
    noexcept

inline void aTransformPreviousColumn ()

inline void aGenerateEventsFromSMGroup (const eSMGroup mGroup)

inline void aResetEventFrame ()

inline bool gIsIncreasingTimestamp (const eColumn column) const

```

```
inline bool gIsSameSubTimestamp (const eColumn column) const  
  
inline bool gIsIncreasingColumnAddress (const eColumn column) const  
  
inline bool gIsGroupParsedAlready (const eSMGroup mGroup) const  
  
inline void processEvent (const eDataLost event)  
  
inline void processEvent (const eTimestampReference event)  
  
inline void processEvent (const eColumn event)  
  
inline void processEvent (const eSMGroup event)
```

Private Members

```
struct dv::io::camera::parser::S5K231Y::Parser mDvs  
  
struct dv::io::camera::parser::S5K231Y::Parser mTimestamps  
  
struct dv::io::camera::parser::S5K231Y::Parser mControls  
  
State mState = {State::WAIT_FOR_TIMESTAMP_REFERENCE}
```

Private Static Functions

```
static inline uint8_t reverseByte (const uint8_t n) noexcept  
  
static inline bool gIsFrameStart (const eColumn column)
```

Private Static Attributes

```
static constexpr std::array<uint8_t, 16> REVERSE_LOOKUP_TABLE = {0x0, 0x8, 0x4, 0xC, 0x2, 0xA, 0x6, 0xE,  
0x1, 0x9, 0x5, 0xD, 0x3, 0xB, 0x7, 0xF}  
  
static constexpr int16_t NUM_GROUPS = {HEIGHT / 8}  
  
static constexpr size_t EVENT_GROUPS_FAST = {64}  
  
static constexpr size_t EVENT_GROUPS_REPEAT = {EVENT_GROUPS_FAST / sizeof(size_t)}
```

```
class Parser : public dv::io::camera::parser::ParserBase
```

Public Functions

```
Parser () = default  
  
inline virtual void parseData (const std::span<const uint8_t> buffer, const bool dataLost) override  
  
inline void injectImu (dv::IMU imuData)
```

Public Members

`std::array<std::array<uint8_t, EVENT_GROUPS_FAST>, 2> columnEvents = {}`

`int16_t columnAddress = {-1}`

`uint8_t frameNumber = {0}`

`bool mirrorMode = {false}`

`std::array<bool, NUM_GROUPS> groupsUsed = {}`

`size_t resetIndex = {0}`

`size_t lastCommittedEvents = {0}`

`int64_t reference = {-1}`

`int64_t referenceOverflow = {0}`

`int32_t lastReference = {-1}`

Private Types

enum class **State**

Values:

enumerator **WAIT_FOR_TIMESTAMP_REFERENCE**

enumerator **WAIT_FOR_TIMESTAMP_SUB_UNIT**

enumerator **WAIT_FOR_COLUMN_START**

enumerator **WAIT_FOR_SMGROUP**

enumerator **WAIT_FOR_COLUMN_OR_SMGROUP_OR_FRAME_END**

Private Functions

inline void **aUpdateTimestampReference** (const *eTimestampReference* timestampRef)

inline void **aUpdateTimestampSubUnit** (const *eTimestampSubUnit* timestampSub)

inline void **aSetFrameNumber** (const *eColumn* column)

```
inline void aSetColumnData (const eColumn column)

inline void bitIndexToOutput (std::vector<dv::Event> &dst, const size_t *src, const bool polarity) const
    noexcept

inline void aTransformPreviousColumn ()

inline void aGenerateEventsFromSMGroup (const eSMGroup mGroup, const int16_t group2Address)

inline void aResetEventFrame ()

inline bool gIsSameFrameNumber (const eColumn column) const

inline bool gVerifyFrameNumber (const eFrameEnd frameEnd) const

inline bool gIsGroupParsedAlready (const eSMGroup mGroup, const int16_t group2Address) const

inline void processEvent (const eDataLost event)

inline void processEvent (const eTimestampReference event)

inline void processEvent (const eTimestampSubUnit event)

inline void processEvent (const eColumn event)

inline void processEvent (const eSMGroup event)

inline void processEvent (const eFrameEnd event)
```

Private Members

```
struct dv::io::camera::parser::S5KRC1S::Parser mDvs

struct dv::io::camera::parser::S5KRC1S::Parser mTimestamps

State mState = {State::WAIT_FOR_TIMESTAMP_REFERENCE}
```

Private Static Functions

```
static inline bool gIsFrameStart (const eColumn column)
```

Private Static Attributes

```
static constexpr int16_t NUM_GROUPS = {HEIGHT / 8}

static constexpr size_t EVENT_GROUPS_FAST = {96}

static constexpr size_t EVENT_GROUPS_REPEAT = {EVENT_GROUPS_FAST / sizeof(size_t)}
```

class **ParserBase**

Subclassed by *dv::io::camera::parser::DAVIS::Parser*, *dv::io::camera::parser::DVS128::Parser*, *dv::io::camera::parser::DVXplorer::Parser*, *dv::io::camera::parser::S5K231Y::Parser*, *dv::io::camera::parser::S5KRC1S::Parser*

Public Functions

```
virtual ~ParserBase () = default

virtual void parseData (std::span<const uint8_t> buffer, bool dataLost) = 0

inline void setLogger (ParserLoggerCallback loggerFunction)

inline void setDataCommitCallback (ParserDataCommitCallback dataCommitCallback)

inline void setTimeInitCallback (ParserTimeInitCallback timeInitCallback)

inline void setTimeInterval (const std::chrono::microseconds timeInterval)

inline std::chrono::microseconds getTimeInterval () const

inline void setSystemOffset (const std::chrono::microseconds systemOffset)

inline std::chrono::microseconds getSystemOffset () const

inline void adjustTimestamps (const std::chrono::microseconds adjust)

inline std::chrono::microseconds getOutstandingTimestampAdjustment () const
```

Public Members

```
std::atomic<int64_t> commitInterval = {10000}

std::atomic<int64_t> systemOffset = {0}

std::atomic<int64_t> adjust = {0}

int64_t lastDevice = {0}

int64_t current = {0}

int64_t nextCommit = {-1}

std::atomic<bool> resetTimingAsync = {false}
```

Protected Functions

```
inline int64_t getCurrentTimestamp () const

template<bool STRICT_MONOTONIC_TIMESTAMP = true>
inline void updateTimestamp (const int64_t timestamp)
```

Update timestamp tracking with the latest timestamp from the device. Always implies a *dataCommit()* too.

Template Parameters

STRICT_MONOTONIC_TIMESTAMP – whether the timestamps from device increase strictly or not.

Parameters

timestamp – latest timestamp from device.

```
inline void dataCommit ()
```

Send all data currently available to the consumer, up to the current timestamp excluded. Usually called on timestamp update. All data previous to this timestamp should be inside the data buffers, but in some cases this is not possible, currently only the case with *DAVIS* frames.

```
inline void timestampInit (const std::chrono::microseconds firstTimestamp)
```

Call on first timestamp initialization and timestamp resets. This will ensure the system offset is set correctly. NOTE: callers must guarantee their own internal timestamp tracking is fully initialized and ready when calling this function!

Parameters

firstTimestamp – first timestamp parsed from data stream from the device’s point-of-view.

Protected Attributes

```
ParserLoggerCallback mLogger = {}
```

```
ParsedData mBuffers = {}
```

Private Functions

```
inline void resetTiming ()
```

Reset timestamping. Only called in the parser thread (via async flag)! systemOffset must have been set prior to calling this.

Private Members

```
ParserDataCommitCallback mDataCommitCallback = {}
```

```
ParserTimeInitCallback mTimeInitCallback = {}
```

```
struct dv::io::camera::parser::ParserBase mTimestamps
```

```
template<concepts::AddressableEvent EventType, class EventPacketType>
```

```
class PartialEventData
```

#include </builds/inivation/dv/dv-processing/include/dv-processing/core/core.hpp> **INTERNAL USE ONLY** Internal event container class that holds a shard of events. A *PartialEventData* holds a shared pointer to an *EventPacket*, which is the underlying data structure. The underlying data can either be const, in which case no addition is allowed, or non const, in which addition of new data is allowed. Slicing is allowed in both cases, as it only modifies the control structure. All the events in the partial have to be monotonically increasing in time. A *PartialEventData* can be sliced both from the front as well as from the back. By doing so, the memory footprint of the structure is not modified, just the internal bookkeeping pointers are readjusted. The *PartialEventData* keeps track of lowest as well as highest times of events in the structure.

The data *PartialEventData* points to can be shared between multiple *PartialEventData*, each with potentially different slicings.

Public Functions

inline explicit **PartialEventData** (const size_t capacity = 10000)

Creates a new *PartialEventData* shard. Allocates new memory on the heap to keep the data. Upon constructions, the newly created object is the sole owner of the data.

Parameters

capacity – Number of events this data partial can store.

inline explicit **PartialEventData** (std::shared_ptr<const *EventPacketType*> data)

Creates a new *PartialEventData* shard from existing const data. Copies the supplied shared_ptr into the structure, acquiring shared ownership of the supplied data.

Parameters

data – The shared pointer to the data to which we want to obtain shared ownership

PartialEventData (const *PartialEventData* &other) = default

Copy constructor. Creates a shallow copy of *other* without copying the actual data over. As slicing does not alter the underlying data, the new copy may be sliced without affecting the original object.

Parameters

other –

inline *iterator* **iteratorAtTime** (const int64_t time) const

Returns an iterator to the first element that is bigger than the supplied timestamp. If every element is bigger than the supplied time, an iterator to the first element is returned (same as *begin()*). If all elements have a smaller timestamp than the supplied, the end iterator is returned (same as *end()*).

Parameters

time – The requested time. The iterator will be the first element with a timestamp larger than this time.

Returns

An iterator to the first element larger than the supplied time.

inline *iterator* **begin** () const

Returns an iterator to the first element of the *PartialEventData*. The iterator is according to the current slice and not to the underlying datastore. E.g. when slicing the shard from the front, the *begin()* will change.

Returns

Returns an iterator at the beginning data partial

inline *iterator* **end** () const

Returns an iterator to one after the last element of the *PartialEventData*. The iterator is according to the current slice and not to the underlying datastore. E.g. when slicing the shard from the back, the result of *end()* will change.

Returns

Returns an iterator at the end of the data partial

inline void **sliceFront** (const size_t number)

Slices off *number* events from the front of the *PartialEventData*. This operation just adjust the book-keeping of the datastructure without actually modifying the underlying data representation. If there are not enough events left, a *range_error* exception is thrown.

Other instances of *PartialEventData* which share the same underlying data are not affected by this.

Parameters

number – amount of events to be removed from the front.

inline void **sliceBack** (const size_t number)

Slices off `number` events from the back of the *PartialEventData*. This operation just adjust the bookkeeping of the datastructure without actually modifying the underlying data representation. If there are not enough events left, a `range_error` exception is thrown.

Other instances of *PartialEventData* which share the same underlying data are not affected by this.

Parameters

`number` – amount of events to be removed from the back.

inline size_t **sliceTimeFront** (const int64_t time)

Slices off all the events that occur before the supplied time. The resulting data structure has a `lowestTime > time` where `time` is the supplied time.

This operation just adjust the bookkeeping of the datastructure without actually modifying the underlying data representation. If there are not enough events left, a `range_error` exception is thrown.

Other instances of *PartialEventData* which share the same underlying data are not affected by this.

Parameters

`time` – the threshold time. All events `<= time` will be sliced off

Returns

number of events that actually got sliced off as a result of this operation.

inline size_t **sliceTimeBack** (const int64_t time)

Slices off all the events that occur after the supplied time. The resulting data structure has a `lowestTime < time` where `time` is the supplied time.

This operation just adjust the bookkeeping of the datastructure without actually modifying the underlying data representation. If there are not enough events left, a `range_error` exception is thrown.

Other instances of *PartialEventData* which share the same underlying data are not affected by this.

Parameters

`time` – the threshold time. All events `> time` will be sliced off

Returns

number of events that actually got sliced off as a result of this operation.

inline void **_unsafe_addEvent** (const *EventType* &event)

UNSAFE OPERATION Copies the data of the supplied event into the underlying data structure and updates the internal bookkeeping to accommodate the event.

NOTE: This function does not perform any boundary checks. Any call to function is expected to have performed the following boundary checks: *canStoreMoreEvents()* to see if there is space to accommodate the new event. *getHighestTime()* has to be smaller or equal than the new event's timestamp, as we require events to be monotonically increasing.

Parameters

`event` – The event to be added

inline void **_unsafe_moveEvent** (*EventType* &&event)

UNSAFE OPERATION Moves the data of the supplied event into the underlying data structure and updates the internal bookkeeping to accommodate the event.

NOTE: This function does not perform any boundary checks. Any call to function is expected to have performed the following boundary checks: *canStoreMoreEvents()* to see if there is space to accommodate the new event. *getHighestTime()* has to be smaller or equal than the new event's timestamp, as we require events to be monotonically increasing.

Parameters

event – The event to be added

inline *EventType* &**front** ()

Get a reference to the first available event in the partial.

Returns

Reference to first element in the partial.

inline *EventType* &**back** ()

Get a reference to the last available event in the partial.

Returns

Reference to last element in the partial.

inline size_t **getLength** () const

The length of the current slice of data. This value can be in range [0; capacity].

Returns

the current length of the slice in number of events.

inline int64_t **getLowestTime** () const

Gets the lowest timestamp of an event that is represented in this Partial. The lowest timestamp is always identical to the timestamp of the first event of the slice.

Returns

The timestamp of the first event in the slice. This is also the lowest time present in this slice.

inline int64_t **getHighestTime** () const

Gets the highest timestamp of an event that is represented in this Partial. The lowest timestamp is always identical to the timestamp of the last event of the slice.

Returns

The timestamp of the last event in the slice. This is also the highest timestamp present in this slice.

inline const *EventType* &**operator** [] (const size_t offset) const

Returns a reference to the element at the given offset of the slice.

Parameters

offset – The offset in the slice of which element a reference should be obtained

Returns

A reference to the object at offset offset

inline bool **canStoreMoreEvents** () const

Checks if it is safe to add more events to this partial. It is safe to add more events when the following conditions are fulfilled:

- The partial does not represent const data. In that case, any modification of the underlying buffer is impossible.
- The partial does not exceed the sharding count limit
- The partial hasn't been sliced from the back

If it has been sliced from the back, adding new events would put them in unreachable space.

Returns

true if there is space available to store more events in this partial.

```
inline size_t availableCapacity () const
```

Amount of space still available in this data partial.

Returns

Amount of events this data partial can store additionally.

```
inline bool merge (const PartialEventData &other)
```

Merge the other data partial into this one by copying the contents, if that is possible. If merge is not possible, the function returns false and does nothing.

Parameters

other – Other data partial to be merged into this one.

Returns

True if merge was successful, false otherwise.

Private Types

```
using iterator = typename std::vector<EventType>::const_iterator
```

Private Members

```
bool referencesConstData_
```

```
size_t start_
```

```
size_t length_
```

```
size_t capacity_
```

```
int64_t lowestTime_
```

```
int64_t highestTime_
```

```
std::shared_ptr<EventPacketType> modifiableDataPtr_
```

```
std::shared_ptr<const EventPacketType> data_
```

Friends

```
friend class dv::io::MonoCameraWriter
```

```
friend class dv::io::NetworkWriter
```

```
template<concepts::AddressableEvent EventType, class EventPacketType>
```

```
class PartialEventDataTimeComparator
```

```
#include </builds/inivation/dv/dv-processing/include/dv-processing/core/core.hpp> INTERNAL USE ONLY  
Comparator Functor that checks if a given time lies within bounds of the event packet
```

Public Functions

inline explicit **PartialEventDataTimeComparator** (const bool lower)

inline bool **operator ()** (const *PartialEventData*<*EventType*, *EventPacketType*> &partial, const int64_t time)
const

Returns true, if the comparator is set to not lower and the given time is higher than the highest timestamp of the partial, or when it is set to lower and the timestamp is higher than the lowest timestamp of the partial.

Parameters

- **partial** – The partial to be analysed
- **time** – The time to be compared against

Returns

true, if time is higher than either lowest or highest timestamp of partial depending on state

inline bool **operator ()** (const int64_t time, const *PartialEventData*<*EventType*, *EventPacketType*> &partial)
const

Returns true, if the comparator is set to not lower and the given time is higher than the lowest timestamp of the partial, or when it is set to lower and the timestamp is higher than the highest timestamp of the partial.

Parameters

- **partial** – The partial to be analysed
- **time** – The time to be compared against

Returns

true, if time is higher than either lowest or lowest timestamp of partial depending on state

Private Members

const bool **lower_**

struct **PixelDisparity**

#include </builds/inivation/dv/dv-processing/include/dv-processing/depth/sparse_event_block_matcher.hpp> Structure containing disparity results for a point of interest.

Public Functions

inline **PixelDisparity** (const cv::Point2i &coordinates, const bool valid, const *std::optional*<float> correlation = *std::nullopt*, const *std::optional*<float> score = *std::nullopt*, const *std::optional*<int32_t> disparity = *std::nullopt*, const *std::optional*<cv::Point2i> &templatePosition = *std::nullopt*, const *std::optional*<cv::Point2i> &matchedPosition = *std::nullopt*)

Initialize the disparity structure.

Parameters

- **coordinates** – Point of interest coordinates, this will contain same coordinates that were passed into the algorithm.
- **valid** – Holds true if the disparity match valid. False otherwise.
- **correlation** – Pearson correlation value for the best matching block, if available. This value is in the range [-1.0; 1.0].
- **score** – Matching score value, if available. This value is in the range [0.0; 1.0].

- **disparity** – Disparity value in pixels, if available. The value is in the range [minDisparity; maxDisparity].
- **templatePosition** – Requested coordinate of interest point in the left (rectified) image pixel space.
- **matchedPosition** – Best match coordinate on the right (rectified) image pixel space.

Public Members

`cv::Point2i` **coordinates**

Point of interest coordinates, this will contain same coordinates that were passed into the algorithm.

`bool` **valid**

Holds true if the disparity match valid. False otherwise.

`std::optional<float>` **correlation**

Pearson correlation value for the best matching block, if available. This value is in the range [-1.0; 1.0]. Correlation value of -1.0 will mean that matched patch is an inverse of the original template patch, 1.0 will be an equal match, 0.0 is no correlation. A positive value indicates a positive correlation between searched template patch and best match, which could be considered a good indication of a correct match.

`std::optional<float>` **score**

Standard score (Z-score) for the match, if available. The score is the number of standard deviations the highest probability value is above the mean of all probabilities of the matching method.

`std::optional<int32_t>` **disparity**

Disparity value in pixels, if available. The value is in the range [minDisparity; maxDisparity].

`std::optional<cv::Point2i>` **templatePosition**

Coordinates of the matching template on the left (rectified) image space. Set to `std::nullopt` if the template coordinates are out-of-bounds.

`std::optional<cv::Point2i>` **matchedPosition**

Coordinates of the matched template on the right (rectified) image space. Set to `std::nullopt` if a match cannot be reliably found, otherwise contains coordinates with highest correlation match on the right side rectified camera pixel space.

class **PixelMotionPredictor**

Public Types

using **SharedPtr** = `std::shared_ptr<PixelMotionPredictor>`

using **UniquePtr** = `std::unique_ptr<PixelMotionPredictor>`

Public Functions

inline explicit **PixelMotionPredictor** (const *camera::CameraGeometry::SharedPtr* &cameraGeometry)

Construct pixel motion predictor class.

Parameters

cameraGeometry – Camera geometry class instance containing intrinsic calibration of the camera sensor.

virtual **~PixelMotionPredictor** () = default

inline *dv::EventStore* **predictEvents** (const *dv::EventStore* &events, const *Transformationf* &dT, const float depth) const

Apply delta transformation to event input and generate new transformed event store with new events that are within the new camera perspective (after applying delta transform).

Parameters

- **events** – Input events.
- **dT** – Delta transformation to be applied.
- **depth** – Scene depth.

Returns

Transformed events.

template<*concepts::Coordinate2DMutableIterable* **Output**, *concepts::Coordinate2DIterable* **Input**>

inline *Output* **predictSequence** (const *Input* &points, const *Transformationf* &dT, const float depth) const

Apply delta transformation to coordinate input and generate new transformed coordinate array with new coordinates that are within the new camera perspective (after applying delta transform).

Parameters

- **points** – Input coordinate array.
- **dT** – Delta transformation to be applied.
- **depth** – Scene depth.

Returns

Transformed point coordinates.

template<*concepts::Coordinate2DConstructible* **Output**, *concepts::Coordinate2D* **Input**>

inline *Output* **predict** (const *Input* &pixel, const *Transformationf* &dT, const float depth) const

Reproject given pixel coordinates using the delta transformation and depth.

Parameters

- **pixel** – Input pixel coordinates.
- **dT** – Delta transformation.
- **depth** – Scene depth.

Returns

Transformed pixel coordinate using the delta transform, camera geometry and scene depth.

inline bool **isUseDistortion** () const

Is the distortion model enabled for the reprojection of coordinates.

Returns

True if the distortion model is enabled, false otherwise.

```
inline void setUseDistortion (bool useDistortion_)
```

Enable of disable the usage of a distortion model.

Parameters

useDistortion_ – Pass true to enable usage of the distortion model, false otherwise.

Private Members

```
const dv::camera::CameraGeometry::SharedPtr camera
```

```
bool useDistortion = false
```

```
struct Pose : public flatbuffers::NativeTable
```

Public Types

```
typedef PoseFlatbuffer TableType
```

Public Functions

```
inline Pose ()
```

```
inline Pose (int64_t _timestamp, const Vec3f &_translation, const Quaternion &_rotation, const std::string &_referenceFrame, const std::string &_targetFrame)
```

Public Members

```
int64_t timestamp
```

```
Vec3f translation
```

```
Quaternion rotation
```

```
std::string referenceFrame
```

```
std::string targetFrame
```

Public Static Functions

```
static inline constexpr const char *GetFullyQualifiedName ()
```

Friends

```
inline friend std::ostream &operator<< (std::ostream &os, const Pose &packet)
```

```
struct PoseBuilder
```

Public Functions

```

inline void add_timestamp (int64_t timestamp)

inline void add_translation (const Vec3f *translation)

inline void add_rotation (const Quaternion *rotation)

inline void add_referenceFrame (flatbuffers::Offset<flatbuffers::String> referenceFrame)

inline void add_targetFrame (flatbuffers::Offset<flatbuffers::String> targetFrame)

inline explicit PoseBuilder (flatbuffers::FlatBufferBuilder &_fbb)

PoseBuilder &operator= (const PoseBuilder &)

inline flatbuffers::Offset<PoseFlatbuffer> Finish ()

```

Public Members

```

flatbuffers::FlatBufferBuilder &fbb_

flatbuffers::uoffset_t start_

```

```
struct PoseFlatbuffer : private flatbuffers::Table
```

#include </builds/inivation/dv/dv-processing/include/dv-processing/data/pose_base.hpp> A struct holding timestamp and pose.

Public Types

```
typedef Pose NativeTableType
```

Public Functions

```

inline int64_t timestamp () const
    Timestamp (μs).

inline const Vec3f *translation () const
    Translational vector.

inline const Quaternion *rotation () const
    Rotation quaternion.

inline const flatbuffers::String *referenceFrame () const
    Name of the reference frame (transforming from)

inline const flatbuffers::String *targetFrame () const
    Name of the target frame (transforming into)

inline bool Verify (flatbuffers::Verifier &verifier) const

inline Pose *UnPack (const flatbuffers::resolver_function_t *_resolver = nullptr) const

inline void UnPackTo (Pose *_o, const flatbuffers::resolver_function_t *_resolver = nullptr) const

```

Public Static Functions

```
static inline const flatbuffers::TypeTable *MiniReflectTypeTable()

static inline constexpr const char *GetFullyQualifiedName()

static inline void UnPackToFrom(Pose *_o, const PoseFlatbuffer *_fb, const flatbuffers::resolver_function_t
                                *_resolver = nullptr)

static inline flatbuffers::Offset<PoseFlatbuffer> Pack(flatbuffers::FlatBufferBuilder &_fb, const Pose *_o, const
                                                    flatbuffers::rehasher_function_t *_rehasher = nullptr)
```

Public Static Attributes

```
static constexpr const char *identifier = "POSE"
```

```
class PoseVisualizer
```

```
#include </builds/inivation/dv/dv-processing/include/dv-processing/visualization/pose_visualizer.hpp> Visualize the
current and past poses of objects as an image.
```

Public Types

```
enum class ViewMode
```

Pre-defined viewing modes; poses of the virtual camera used to visualize the 3D scene. The naming of the different view modes is based on the plane that the virtual camera sees in that mode, such that the virtual camera is placed perpendicular to that plane.

The first letter in the viewing mode defines the horizontal axis of the plane, while the second letter defines the vertical axis of the plane.

Values:

```
enumerator CUSTOM
```

```
enumerator VIEW_XY
```

```
enumerator VIEW_YZ
```

```
enumerator VIEW_ZX
```

```
enumerator VIEW_XZ
```

```
enumerator VIEW_YX
```

```
enumerator VIEW_ZY
```

```
enum class GridPlane
```

Defined grid plane orientations that can be drawn into the 3D scene. The first letter defines the horizontal axis of the plane, while the second letter defines the vertical axis of the plane.

Values:

enumerator **PLANE_NONE**

enumerator **PLANE_XY**

enumerator **PLANE_YZ**

enumerator **PLANE_ZX**

Public Functions

```
inline explicit PoseVisualizer (const cv::Size &resolution=cv::Size(640, 480),
const cv::Scalar &backgroundColor=colors::darkGray,
const cv::Scalar &gridColor=colors::gray, const float gridSpacing=1.0f,
const size_t maxTrajectoryLength=10 '000, const size_t landmarkLimit=10 '000)
```

Initialize a pose visualizer to visualize the current and past poses of objects as an image. This is done by creating a virtual camera in the 3D scene, and using this camera to visualize object poses/landmarks in the 3D scene.

The virtual camera is assumed to have an ideal pinhole camera model.

Parameters

- **resolution** – Resolution of the virtual camera used for visualizing the 3D scene. This is used to set the intrinsics of the virtual camera using an ideal pinhole camera model and defines the resolution of the generated image.
- **backgroundColor** – Background color.
- **gridColor** – Color of drawn grid plane.
- **gridSpacing** – Spacing between drawn lines for grid plane, in physical units [m].
- **maxTrajectoryLength** – Maximum number of poses that are visualized for each object.
- **landmarkLimit** – Maximum number of landmarks that are visualized.

```
inline PoseVisualizer (const float fx, const float fy, const float cx, const float cy,
const cv::Size &resolution, const cv::Scalar &backgroundColor=colors::darkGray,
const cv::Scalar &gridColor=colors::gray, const float gridSpacing=1.0f,
const size_t maxTrajectoryLength=10 '000, const size_t landmarkLimit=10 '000)
```

Initialize a pose visualizer to visualize the current and past poses of objects as an image. This is done by creating a virtual camera in the 3D scene, and using this camera to visualize object poses/landmarks in the 3D scene.

The virtual camera is assumed to have an ideal pinhole camera model.

Parameters

- **fx** – Focal length X for the virtual camera, measured in pixels.
- **fy** – Focal length Y for the virtual camera, measured in pixels.
- **cx** – Central point coordinate X for the virtual camera, in pixels.
- **cy** – Central point coordinate Y for the virtual camera, in pixels.
- **resolution** – Resolution of the virtual camera. This defines the resolution of the generated image.

- **backgroundColor** – Background color.
- **gridColor** – Color of drawn grid plane.
- **gridSpacing** – Spacing between drawn lines for grid plane, in physical units [m]
- **maxTrajectoryLength** – Maximum number of poses that are visualized for each object.
- **landmarkLimit** – Maximum number of landmarks that are visualized.

```
inline explicit PoseVisualizer (const dv::camera::CameraGeometry &cameraGeometry,  
const cv::Scalar &backgroundColor=colors::darkGray,  
const cv::Scalar &gridColor=colors::gray, const float gridSpacing=1.0f,  
const size_t maxTrajectoryLength=10 '000, const size_t landmarkLimit=10 '000)
```

Initialize a pose visualizer to visualize the current and past poses of objects as an image. This is done by creating a virtual camera in the 3D scene, and using this camera to visualize object poses/landmarks in the 3D scene.

Parameters

- **cameraGeometry** – CameraGeometry instance defining the intrinsics of the virtual camera used for visualizing the 3D scene.
- **backgroundColor** – Background color.
- **gridColor** – Color of drawn grid plane.
- **gridSpacing** – Spacing between drawn lines for grid plane, in physical units [m]
- **maxTrajectoryLength** – Maximum number of poses that are visualized for each object.
- **landmarkLimit** – Maximum number of landmarks that are visualized.

```
inline void accept (const std::string &objectName, const dv::Pose &objectPose)
```

Add a new object pose to the visualization.

Parameters

- **objectName** – Name of the object for which the pose is to be added.
- **objectPose** – Added pose of the object.

```
inline void accept (const std::string &objectName, const dv::kinematics::Transformationf &objectPose)
```

Add a new object pose to the visualization.

Parameters

- **objectName** – Name of the object for which the pose is to be added.
- **objectPose** – Added pose of the object.

```
inline void accept (const dv::Pose &objectPose)
```

Add a new object pose to the visualization.

Parameters

- **objectPose** – Added pose of the object.

```
inline void accept (const dv::kinematics::Transformationf &objectPose)
```

Add a new object pose to the visualization.

Parameters

- **objectPose** – Added pose of the object.

inline void **accept** (const *dv::LandmarksPacket* &landmarks)

Add landmarks to the visualization.

Parameters

landmarks – A packet of landmarks to be added to the list of landmarks drawn.

inline void **accept** (const *dv::Landmark* &landmark)

Add a landmark to the visualization.

Parameters

landmark – A single landmark to be added to the list of landmarks drawn.

inline *dv::Frame* **generateFrame** ()

Return a visualization image of the 3D scene from the defined virtual camera perspective.

Returns

The generated image for visualization.

inline int64_t **getTimestamp** () const

Return the timestamp of the most recent object pose.

Returns

Timestamp in Unix microsecond format.

inline void **clearLandmarkWithId** (const int64_t id)

Clear landmark with given ID from the visualization.

Parameters

id – ID of landmark to be removed.

inline void **clearLandmarksInInterval** (const int64_t startTime, const int64_t endTime)

Clear all landmarks from the visualization within the time interval [startTime, endTime).

Parameters

- **startTime** – The start time of the removed slice of landmarks (inclusive).
- **endTime** – The end time of the removed slice of landmarks (exclusive).

inline void **clearLandmarks** ()

Clear all landmarks from the visualization.

inline void **clearObject** (const *std::string* &objectName)

Clear all saved poses for the given object from the visualization.

Parameters

objectName – Name of the object for which the list of poses is to be removed.

inline void **clearObjectPosesInInterval** (const int64_t startTime, const int64_t endTime)

Clear all saved object poses from the visualization with the time interval [startTime, endTime).

Parameters

- **startTime** – The start time of the removed slice of object poses (inclusive).
- **endTime** – The end time of the removed slice of object poses (exclusive).

inline void **clearAllObjects** ()

Clear all saved object poses from the visualization.

inline void **reset** ()

Reset the visualizer to a default state (default camera position, resolution, etc.).

```
inline cv::Size getResolution () const
```

Getters and setters.

Get the resolution of the virtual camera used for visualization.

Returns

Resolution of the virtual camera.

```
inline void setResolution (const cv::Size &resolution)
```

Set the resolution of the virtual camera used for visualization.

Parameters

resolution – Resolution of the virtual camera.

```
inline dv::camera::CameraGeometry getCameraIntrinsics () const
```

Get the intrinsics of the virtual camera used for visualization.

Returns

CameraGeometry instance describing the intrinsics of the virtual camera.

```
inline void setCameraIntrinsics (const dv::camera::CameraGeometry &cameraGeometry)
```

Set the intrinsics of the virtual camera used for visualization.

Parameters

cameraGeometry – CameraGeometry instance describing the intrinsics of the virtual camera.

```
inline void setCameraIntrinsics (const float fx, const float fy, const float cx, const float cy, const cv::Size  
                                &resolution)
```

Set the intrinsics of the virtual camera used for visualization assuming an ideal pinhole camera model.

Parameters

- **fx** – Focal length X for the virtual camera, measured in pixels.
- **fy** – Focal length Y for the virtual camera, measured in pixels.
- **cx** – Central point coordinate X for the virtual camera, in pixels.
- **cy** – Central point coordinate Y for the virtual camera, in pixels.
- **resolution** – Resolution of the virtual camera.

```
template<concepts::Coordinate3DConstructible Output = Eigen::Vector3f>
```

```
inline Output getCameraPosition () const
```

Get the position of the virtual camera used for visualization with respect to the world coordinate system.

Template Parameters

Output – Type of the 3D vector used for the camera position.

Returns

Position of the virtual camera in the world coordinate system.

```
template<concepts::Coordinate3D InputType>
```

```
inline void setCameraPosition (const InputType &newPosition)
```

Set the position of the virtual camera used for visualization with respect to the world coordinate system.

Template Parameters

InputType – Type of the 3D vector representing the camera position.

Parameters

newPosition – Position of the virtual camera in the world coordinate system.

```
template<concepts::Coordinate3DConstructible Output = Eigen::Vector3f>
```

inline *Output* **getCameraOrientation** () const

Get the orientation of the virtual camera used for visualization as XYZ Euler angles (in degrees) with respect to the world coordinate system.

Template Parameters

Output – Type of the 3D vector representing the camera orientation.

Returns

XYZ Euler angles representing the camera orientation with respect to the world coordinate system (in degrees).

template<*concepts::Coordinate3D* **InputType**>

inline void **setCameraOrientation** (const *InputType* &newOrientation)

Set the orientation of the virtual camera used for visualization as XYZ Euler angles (in degrees) with respect to the world coordinate system.

Template Parameters

InputType – Type of the 3D vector representing the camera orientation

Parameters

newOrientation – XYZ Euler angles representing the new camera orientation with respect to the world coordinate system (in degrees).

inline *dv::kinematics::Transformationf* **getCameraPose** ()

Get the pose of the virtual camera used for visualization with respect to the world coordinate system.

Returns

Transformationf instance describing the pose of the virtual camera with respect to the world coordinate system.

inline void **setCameraPose** (const *dv::kinematics::Transformationf* &T_W_C)

Set the pose of the virtual camera used for visualization with respect to the world coordinate system.

Parameters

T_W_C – Transformationf instance describing the pose of the virtual camera with respect to the world coordinate system.

inline *ViewMode* **getViewMode** () const

Get the view mode (pre-defined pose) of the virtual camera used for visualization.

Returns

The current viewing mode of the virtual camera.

inline void **setViewMode** (const *ViewMode* mode)

Set the view mode (pre-defined pose) of the virtual camera used for visualization to one of the pre-defined view modes.

Parameters

mode – Viewing mode of the virtual camera.

inline *GridPlane* **getGridPlane** () const

Get the grid plane orientation with respect to the world coordinate system (from a list of pre-defined orientations).

Returns

Orientation of the visualized grid plane.

inline void **setGridPlane** (const *GridPlane* plane)

Set the grid plane orientation with respect to the world coordinate system (from a list of pre-defined orientations). If set to PLANE_NONE, the grid plane is removed from the visualization.

Parameters

plane – Orientation of the visualized grid plane.

inline float **getGridSpacing** () const

Get the spacing between drawn lines used for visualizing the grid plane, in physical units [m].

Returns

Spacing between drawn lines for visualizing the grid plane.

inline void **setGridSpacing** (const float gridSpacing)

Set the spacing between drawn lines used for visualizing the grid plane, in physical units [m].

Parameters

gridSpacing – Spacing between drawn lines for visualizing the grid plane.

inline float **getWorldCoordinateSize** () const

Get the size of the visualized world coordinate frame, in physical units [m].

Returns

Size of the visualized world coordinate frame.

inline void **setWorldCoordinateSize** (const float newSize)

Set the size of the visualized world coordinate frame, in physical units [m]. To disable the visualization of the world coordinate frame, set this value to a value ≤ 0 .

Parameters

newSize – Size of the visualized world coordinate frame.

inline float **getObjectCoordinateSize** () const

Get the size of the visualized coordinate frame for object poses, in physical units [m].

Returns

Size of the visualized coordinate frame for object poses.

inline void **setObjectCoordinateSize** (const float newSize)

Get the size of the visualized coordinate frame for object poses, in physical units [m]. To disable the visualization of the coordinate frame, set this value to a value ≤ 0 .

Parameters

newSize – Size of the visualized coordinate frame for object poses.

inline int32_t **getLineThickness** () const

Get the line thickness used for drawing, in pixels.

Returns

Drawing line thickness in pixels.

inline void **setLineThickness** (const int32_t newThickness)

Set the line thickness used for drawing, in pixels.

Parameters

newThickness – Drawing line thickness in pixels.

inline const cv::Scalar &**getBackgroundColor** () const

Get the background color.

Returns

Background color.

inline void **setBackgroundColor** (const cv::Scalar &backgroundColor)

Set new background color.

Parameters

backgroundColor – OpenCV scalar for the background color.

inline const cv::Scalar &**getGridColor** () const

Get the grid line color.

Returns

Grid line color

inline void **setGridColor** (const cv::Scalar &gridColor)

Set new grid line color

Parameters

mGridColor – OpenCV scalar for the grid line color.

inline size_t **getLandmarkLimit** () const

Get the maximum number of landmarks to be drawn.

Returns

Maximum number of landmarks

inline void **setLandmarkLimit** (const size_t numLandmarks)

Set a limit for number of landmarks that are stored and drawn.

Parameters

numLandmarks – Number of landmarks

inline size_t **getNumberOfLandmarks** () const

Get the number of landmarks currently stored in the visualizer.

Returns

Number of landmarks stored in the visualizer

inline size_t **getNumberOfObjects** () const

Get the number of objects who's poses are currently stored in the visualizer.

Returns

Number of objects who's poses are stored in the visualizer

inline bool **isAutoScalingEnabled** () const

Get if auto-scaling of the visualizer is enabled; scaling of the grid and viewing camera position based on the minimum and maximum bounds from the provided data.

Returns

True if auto-scaling is enabled.

inline void **setAutoScaling** (const bool enable)

Enable/disable auto-scaling of the visualizer; scaling of the grid and viewing camera position based on the minimum and maximum bounds from the provided data.

Parameters

enable – If true, enable grid autoscaling.

inline bool **isLegendVisualizationEnabled** () const

Get if the visualization of the legend is enabled.

Returns

True if legend visualization is enabled.

inline void **setLegendVisualization** (const bool enable)

Enable the visualization of a legend showing the viewing camera position, latest timestamp of provided data, and the spacing of the drawn grid.

Parameters

enable – If true, enable legend visualization.

Private Functions

inline void **updateBounds** (const float x, const float y, const float z)

Update the minimum and maximum bounds of the visualizer data based on a provided 3D point.

Parameters

- **x** – X position of 3D point.
- **y** – Y position of 3D point.
- **z** – Z position of 3D point.

inline float **computeSpan** (const *GridPlane* grid) const

Compute the maximum span of the provided visualizer data along a given plane.

Parameters

grid – Plane along which the maximum span of the provided visualizer data is computed.

Returns

Maximum span of provided visualizer data along the given plane.

inline void **setCameraExtrinsicsFromViewMode** ()

Use the camera view to set the pose the virtual camera.

inline float **getGridHalfWidth** () const

Compute a nice value for the half width of the drawn grid based on the viewing camera position in the world coordinate system. The value is chosen so that when the grid is placed relative to the viewing camera position, and the viewing camera is oriented at 90 degrees with respect to the grid, the width of the grid is at least 5 times the minimum width so that edge of the drawn grid is visible in the camera view.

Returns

Half width of the drawn grid.

inline bool **isWithinGridBounds** (const Eigen::Vector3f &pos) const

inline void **updateGridSpacing** ()

For use in autoscaling mode, update the grid spacing based on the viewing camera position so that the number of drawn grid lines along each axis of the grid is less than LIMIT_NUM_GRID_LINES. Note that rather than continuously varying the grid spacing, the grid spacing grows/shrinks in integer multiples of GRID_SPACING_GROWTH_FACTOR.

inline void **addText** (cv::Mat &image, const cv::Point2i ptInPx, const *std::string* &text, const cv::Scalar &color) const

Add text to the visualizer image.

Parameters

- **image** – Visualization image to draw to.
- **ptInPx** – 2D pixel position of added text.
- **text** – Text to be added to the image.
- **color** – Color of the added text.

```
inline cv::Size getTextSize (const std::string &text) const
```

Get the size of the drawn text.

Parameters

text – Text to draw.

Returns

Size of the bounding box for the drawn text on the image.

```
inline cv::Point2i project (const Eigen::Vector3f &ptInCam) const
```

```
inline void drawPoint (cv::Mat &image, const Eigen::Vector3f &pt, const int32_t radiusPx, const cv::Scalar  
&color, const std::string &textToAdd = "") const
```

Draw a 3D point represented in the world coordinate system onto the visualization image.

Parameters

- **image** – Visualization image to draw to.
- **pt** – 3D point represented in the world coordinate system.
- **radiusPx** – Radius of the drawn circle for the point, represented in pixels.
- **color** – Color of the drawn point.
- **textToAdd** – Optional text to additionally add next to the drawn point.

```
inline void drawLine (cv::Mat &image, const Eigen::Vector3f &startPt, const Eigen::Vector3f &endPt, const  
cv::Scalar &color, const std::string &textToAdd = "") const
```

Draw a line in 3D space based on the endpoints of the line in the world coordinate system.

Parameters

- **image** – Visualization image to draw to.
- **startPt** – 3D point given in the world coordinate system representing the start point of the line.
- **endPt** – 3D point given in the world coordinate system representing the end point of the line.
- **color** – Color of the drawn line.
- **textToAdd** – Optional text to additionally add next to the drawn line.

```
inline void drawGrid (cv::Mat &image) const
```

Draw a uniform grid onto the visualization image representing the defined grid plane.

Parameters

image – Visualization image to draw to.

```
inline void drawCoordinateFrame (cv::Mat &image, const dv::kinematics::Transformationf &T_W_O, const  
float frameSize, const std::optional<cv::Scalar> &frameColor =  
std::nullopt, const std::string &textToAdd = "") const
```

Draw a coordinate frame onto the visualization image.

Parameters

- **image** – Visualization image to draw to.
- **T_W_O** – Transform representing the pose of the coordinate frame with respect to the world coordinate system.
- **frameSize** – Size of the frame to be drawn [m].

- **frameColor** – Optional color of the coordinate frame drawn. If not specified, defaults to red for x axis, lime for y axis, and blue for z axis.
- **textToAdd** – Optional text to additionally add next to the drawn coordinate frame.

inline void **drawWorldCoordinateSystem** (cv::Mat &image) const

Draw the world coordinate frame onto the visualization image.

Parameters

image – Visualization image to draw to.

inline void **drawObjectPoses** (cv::Mat &image) const

Draw the different object poses onto the visualization image.

Parameters

image – Visualization image to draw to.

inline void **drawLandmarks** (cv::Mat &image) const

Draw the defined landmarks onto the visualization image.

Parameters

image – Visualization image to draw to.

inline void **drawLegend** (cv::Mat &image) const

Draw a legend onto the image specifying the viewing camera position, latest timestamp, and grid spacing

Parameters

image –

Private Members

dv::camera::CameraGeometry **mViewingCameraGeometry**

Intrinsics of the virtual camera used for visualization.

dv::kinematics::Transformationf **mT_C_W**

Extrinsics of the virtual camera used for visualization.

Eigen::Vector3f **mCameraPosition**

ViewMode **mViewMode** = {*ViewMode::VIEW_XY*}

cv::Scalar **mBackgroundColor** = {*dv::visualization::colors::darkGray*}

Colors for drawing the visualization image.

cv::Scalar **mGridColor** = {*dv::visualization::colors::gray*}

GridPlane **mGridPlane** = {*GridPlane::PLANE_XY*}

Parameters for visualized grid.

bool **mEnableAutoScale** = {true}

float **mGridSpacing**

Parameters for scaling grid spacing.

bool **mBoundsSet** = {false}

Bounds for range of drawn points.

float **mMinBoundX** = {*std::numeric_limits<float>::max()*}

float **mMinBoundY** = {*std::numeric_limits<float>::max()*}

float **mMinBoundZ** = {*std::numeric_limits<float>::max()*}

float **mMaxBoundX** = {*std::numeric_limits<float>::min()*}

float **mMaxBoundY** = {*std::numeric_limits<float>::min()*}

float **mMaxBoundZ** = {*std::numeric_limits<float>::min()*}

std::unordered_map<std::string, dv::kinematics::LinearTransformer> **mObjectTrajectories**

Visualized object poses and the maximum number of poses that are visualized.

size_t **mTrajectoryLength**

std::unordered_map<int64_t, dv::Landmark> **mLandmarks**

Visualized landmarks and the maximum number landmarks that are visualized.

size_t **mLandmarkLimit**

float **mWorldCoordinateFrameSize** = {1.f}

Visualization parameters for size of coordinate frames and line thicknesses.

float **mObjectCoordinateFrameSize** = {0.2}

int32_t **mLineThickness** = {1}

bool **mEnableLegend** = {true}

Option to enable/disable legend visualization.

Private Static Attributes

static constexpr float **PIXEL_PITCH** = {9e-6f}

static constexpr float **DEFAULT_CAMERA_HEIGHT** = {5.0f}

```
static constexpr float GRID_SPACING_GROWTH_FACTOR = {2.0f}
```

```
static constexpr size_t LIMIT_NUM_GRID_LINES = {100}
```

```
static constexpr float RAD_TO_DEG = 180.0f / std::numbers::pi_v<float>
```

Static variables for converting from degrees to radians.

```
static constexpr float DEG_TO_RAD = 1.f / RAD_TO_DEG
```

Friends

```
inline friend std::ostream &operator<< (std::ostream &os, const PoseVisualizer::ViewMode &var)
```

```
inline friend std::ostream &operator<< (std::ostream &os, const PoseVisualizer::GridPlane &var)
```

class **Reader**

Public Types

```
using ReadHandler = dv::std_function_exact<void(std::vector<std::byte>&, const int64_t)>
```

Public Functions

```
inline explicit Reader (dv::io::support::TypeResolver resolver = dv::io::support::defaultTypeResolver,  
                      std::unique_ptr<dv::io::support::IOStatistics> stats = nullptr)
```

```
~Reader () = default
```

```
Reader (const Reader &other) = delete
```

```
Reader &operator= (const Reader &other) = delete
```

```
Reader (Reader &&other) noexcept = default
```

```
Reader &operator= (Reader &&other) noexcept = default
```

```
inline void verifyVersion (const ReadHandler &readHandler)
```

```
inline std::unique_ptr<const dv::IOHeader> readHeader (const ReadHandler &readHandler)
```

```
inline std::unique_ptr<const dv::FileDataTable> readFileDataTable (const uint64_t size, const int64_t  
                                                                position, const ReadHandler  
                                                                &readHandler)
```

```
inline std::tuple<dv::PacketHeader, std::unique_ptr<dv::types::TypedObject>, const dv::io::support::Sizes> readPacket (const  
                                                                Read-  
                                                                Han-  
                                                                dler  
                                                                &read-  
                                                                Han-  
                                                                dler)
```

```

inline std::tuple<dv::PacketHeader, std::unique_ptr<dv::types::TypedObject>, const dv::io::support::Sizes> readPacket (const
                                                                    int64_t
                                                                    by-
                                                                    te-
                                                                    Off-
                                                                    set,
                                                                    const
                                                                    Read-
                                                                    Han-
                                                                    dler
                                                                    &read-
                                                                    Han-
                                                                    dler)

inline dv::PacketHeader readPacketHeader (const ReadHandler &readHandler)

inline dv::PacketHeader readPacketHeader (const int64_t byteOffset, const ReadHandler &readHandler)

inline std::pair<std::unique_ptr<dv::types::TypedObject>, const dv::io::support::Sizes> readPacketBody (const
                                                                    dv::FileDataDefinition
                                                                    &packet,
                                                                    const
                                                                    Read-
                                                                    Han-
                                                                    dler
                                                                    &read-
                                                                    Han-
                                                                    dler)

inline std::pair<std::unique_ptr<dv::types::TypedObject>, const dv::io::support::Sizes> readPacketBody (const
                                                                    int32_t
                                                                    streamId,
                                                                    const
                                                                    uint64_t
                                                                    size,
                                                                    const
                                                                    Read-
                                                                    Han-
                                                                    dler
                                                                    &read-
                                                                    Han-
                                                                    dler)

```

```
inline std::pair<std::unique_ptr<dv::types::TypedObject>, const dv::io::support::Sizes> readPacketBody (const
                                                                    int32_t
                                                                    streamId,
                                                                    const
                                                                    uint64_t
                                                                    size,
                                                                    const
                                                                    int64_t
                                                                    byte-
                                                                    Offset,
                                                                    const
                                                                    Read-
                                                                    Han-
                                                                    dler
                                                                    &read-
                                                                    Han-
                                                                    dler)
```

```
inline std::unique_ptr<const dv::FileDataTable> buildFileDataTable (const uint64_t fileSize, const
                                                                    ReadHandler &readHandler)
```

```
inline std::vector<dv::io::Stream> getStreams () const
```

```
inline CompressionType getCompressionType () const
```

Private Functions

```
inline void readFromInput (const uint64_t length, const int64_t position, const ReadHandler &readHandler)
```

```
inline void decompressData ()
```

Private Members

```
dv::io::support::TypeResolver mTypeResolver
```

```
std::unique_ptr<dv::io::support::IOStatistics> mStats
```

```
std::unique_ptr<dv::io::compression::DecompressionSupport> mDecompressionSupport
```

```
std::vector<std::byte> mReadBuffer
```

```
std::vector<std::byte> mDecompressBuffer
```

```
std::unordered_map<int32_t, dv::io::Stream> mStreams
```

Private Static Functions

```
static inline std::unique_ptr<const dv::IOHeader> decodeHeader (const std::vector<std::byte> &header)
```

```
static inline std::unique_ptr<const dv::FileDataTable> decodeFileDataTable (const std::vector<std::byte>
                                                                    &table)
```

```
static inline std::unique_ptr<dv::types::TypedObject> decodePacketBody (const std::vector<std::byte>
                                                                    &packet, const dv::types::Type
                                                                    &type)
```

```
class ReadOnlyFile : private dv::io::SimpleReadOnlyFile
```

Public Functions

```
ReadOnlyFile () = delete
```

```
inline explicit ReadOnlyFile (const std::filesystem::path &filePath, const dv::io::support::TypeResolver
                                &resolver = dv::io::support::defaultTypeResolver,
                                std::unique_ptr<dv::io::support::IOStatistics> stats = nullptr)
```

```
inline const auto &getFileInfo () const
```

```
inline std::vector<std::pair<std::unique_ptr<dv::types::TypedObject>, const dv::io::support::Sizes>> read (const
                                                                                                     int64_t
                                                                                                     start-
                                                                                                     Times-
                                                                                                     tamp,
                                                                                                     const
                                                                                                     int64_t
                                                                                                     end-
                                                                                                     Times-
                                                                                                     tamp,
                                                                                                     const
                                                                                                     int32_t
                                                                                                     streamId)
```

Return all packets containing data with timestamps between a given start and end timestamp, meaning all data with a timestamp in [start, end].

Parameters

- **startTimestamp** – start timestamp of range, inclusive.
- **endTimestamp** – end timestamp of range, inclusive.
- **streamId** – data stream ID (separate logical type).

Returns

packets containing data within given timestamp range.

```
inline std::pair<std::unique_ptr<dv::types::TypedObject>, const dv::io::support::Sizes> read (const
                                                                                                     dv::FileDataDefinition
                                                                                                     &packet)
```

```
inline std::pair<std::unique_ptr<const dv::types::TypedObject>, const dv::io::support::Sizes> read (const int32_t
                                                                                                     streamId,
                                                                                                     const uint64_t
                                                                                                     size, const
                                                                                                     int64_t
                                                                                                     byteOffset)
```

Public Static Functions

```
static inline bool inRange (const int64_t rangeStart, const int64_t rangeEnd, const dv::FileDataDefinition
                             &packet)
```

```
static inline bool aheadOfRange (const int64_t rangeStart, const int64_t rangeEnd, const dv::FileDataDefinition
                                   &packet)
```

```
static inline bool pastRange (const int64_t rangeStart, const int64_t rangeEnd, const dv::FileDataDefinition
                               &packet)
```

Private Functions

```
inline void parseHeader ()
```

```
inline void loadFileDataTable ()
```

```
inline void readClbk (std::vector<std::byte> &data, const int64_t byteOffset)
```

```
inline void createFileInfo ()
```

Private Members

```
dv::io::FileInfo mFileInfo
```

```
dv::io::Reader mReader
```

Private Static Functions

```
static inline std::vector<dv::FileDataDefinition>::const_iterator getStartingPointForTimeRangeSearch (const
                                                                 int64_t
                                                                 start-
                                                                 Times-
                                                                 tamp,
                                                                 const
                                                                 dv::FileDataTable
                                                                 &stream-
                                                                 DataT-
                                                                 able)
```

```
class RedetectionStrategy
```

```
#include </builds/inivation/dv/dv-processing/include/dv-processing/features/redetection_strategy.hpp> Implementa-
tion of different redetection strategies for trackers.
```

```
Subclassed by dv::features::FeatureCountRedetection, dv::features::NoRedetection, dv::features::UpdateIntervalOr-
FeatureCountRedetection, dv::features::UpdateIntervalRedetection
```

Public Types

```
typedef std::shared_ptr<RedetectionStrategy> SharedPtr
```

```
typedef std::unique_ptr<RedetectionStrategy> UniquePtr
```

Public Functions

virtual bool **decideRedetection** (const *dv::features::TrackerBase* &tracker) = 0

Decide the redetection of tracker features depending on the state of the tracker.

Parameters

tracker – Current state of the tracker.

Returns

True to perform redetection of features, false to continue.

virtual ~**RedetectionStrategy** () = default

struct **Result**

#include </builds/inivation/dv/dv-processing/include/dv-processing/features/tracker_base.hpp> *Result* of tracking.

Public Types

typedef *std::shared_ptr*<*Result*> **SharedPtr**

typedef *std::shared_ptr*<const *Result*> **ConstPtr**

Public Functions

inline **Result** (const int64_t _timestamp, const *std::vector*<*dv::TimedKeyPoint*> &_keypoints, const bool keyframe)

Construct tracking result

Parameters

- **_timestamp** – Execution time of tracking
- **_keypoints** – The resulting features
- **keyframe** – Whether this set of features can be regarded as a keyframe (redetection was triggered)

Result () = default

Public Members

std::vector<*dv::TimedKeyPoint*> **keypoints** = {}

A vector of keypoints.

bool **asKeyFrame** = false

A flag that notifies the user of the tracker that this specific input caused redetection to happen and the tracker not only tracked the buffered events, but also detected new features.

int64_t **timestamp** = 0

Timestamp of the execution, it can be frame timestamp or last timestamp of an event slice.

class **RotationIntegrator**

Public Functions

```
inline explicit RotationIntegrator (const dv::kinematics::Transformationf &T_S_target =  
                                     dv::kinematics::Transformationf(), int64_t sensorToTargetTimeOffset =  
                                     0, const Eigen::Vector3f &gyroscopeOffset = {0.f, 0.f, 0.f})
```

Parameters

- **T_S_target** – initial target position wrt to sensor
- **sensorToTargetTimeOffset** – temporal offset between sensor (imu) and target. $t_{\text{target}} = t_{\text{sensor}} - \text{offset}$
- **gyroscopeOffset** – constant measurement offset in gyroscope samples [radians].

```
inline Eigen::Matrix3f getRotation () const
```

Getter outputting current target transformation relative to (target) initial one

Returns

[3x3] rotation matrix

```
inline void setT_S_target (const dv::kinematics::Transformationf &T_S_target)
```

Setter to update target position wrt to the sensor

Parameters

T_S_target – new target transformation wrt sensor

```
inline int64_t getTimestamp () const
```

Getter outputting timestamp of current target transformation

Returns

timestamp

```
inline dv::kinematics::Transformation<float> getTransformation () const
```

Getter returning [4x4] transformation corresponding to current target position wrt (target) initial one

Returns

4x4 transformation corresponding to current integrated rotation

```
inline void accept (const dv::IMU &imu)
```

Update sensor position with new measurement

Parameters

imu – single imu measurement

Private Functions

```
inline Eigen::Matrix3f rotationMatrixFromImu (const dv::IMU &imu, const float dt)
```

Transform gyroscope measurement into rotation matrix representation

Parameters

imu – single imu measurement

Returns

[3x3] rotation matrix corresponding to rotation measured from gyroscope

Private Members

```
Eigen::Matrix4f mT_S0_target
```

matrix storing target position wrt to the sensor (imu)

```
int64_t mSensorToTargetTimeOffset
    offset [us] between sensor and target: t_target = t_sensor - offset

Eigen::Vector3f mGyroscopeOffset
    measurement offset [radians] along each x, y, z axis of the sensor

Eigen::Matrix3f mR_S0_S = Eigen::Matrix3f::Identity(3, 3)
    matrix storing current sensor orientation wrt initial sensor orientation

int64_t mTimestamp = -1
    timestamp of current sensor position wrt initial time.
```

```
class RotationLossFuncutor : public dv::optimization::OptimizationFuncutor<float>
```

```
#include </builds/inivation/dv/dv-processing/include/dv-processing/optimization/contrast_maximization_rotation.hpp>
```

Given a chunk of events, the idea of contrast maximization is to warp events in space and time given a predefined motion model. Contrast maximization aims at finding the optimal parameters of the given motion model. The idea is that if the motion is perfectly estimated, all events corresponding to the same point in the scene, will be warped to the same image plane location, at a given point in time. If this happens, the reconstructed event image will be sharp, having high contrast. This high contrast is measured as variance in the image. For this reason, contrast maximization searches for the best motion parameters which maximize the contrast of the event image reconstructed after warping events in space to a specific point in time. In order to warp event in space and time we use the “dv::kinematics::MotionCompensator” class. This contrast maximization class assumes pure camera rotational motion model. Given a set of imu samples and events in a time range, gyroscope measurement offset is optimized. The gyroscope offset is optimized instead of each single gyroscope measurement in order to limit the search space of the non linear optimization. In addition, given the high sample rate of imu, it would be hard to achieve real time computing optimizing each single gyroscope value. For this reason, the gyroscope offset (x, y, z) is optimized and assumed to be constant among all the gyroscope samples.

Public Functions

```
inline RotationLossFuncutor (dv::camera::CameraGeometry::SharedPtr &camera, const dv::EventStore
    &events, float contribution, const std::vector<dv::IMU> &imuSamples, const
    dv::kinematics::Transformationf &T_S_target, int64_t
    imuToCamTimeOffsetUs, int inputDim, int numMeasurements)
```

This contrast maximization class assumes pure camera rotational motion model. Given a set of imu samples and events in a time range, gyroscope measurement offset is optimized. The gyroscope offset is optimized instead of each single gyroscope measurement in order to limit the search space of the non linear optimization.

Parameters

- **camera** – Camera geometry used to create motion compensator
- **events** – Events used to compute motion compensated image
- **contribution** – Contribution value of each event to the total pixel intensity
- **imuSamples** – Chunk of imu samples used to compensate events. These values (gyroscope part) are updated with the gyroscope measurement offset, which is the optimized variable.
- **T_S_target** – Transformation from sensor (imu) to target (camera). Used to convert imu motion into camera motion.
- **imuToCamTimeOffsetUs** – Time synchronization offset between imu and camera
- **inputDim** – Number of parameters to optimize

- **numMeasurements** – Number of function evaluation performed to compute the gradient

```
inline virtual int operator () (const Eigen::VectorXf &gyroscopeOffsetImu, Eigen::VectorXf &stdInverse)
    const
```

Implementation of the objective function: optimize gyroscope offset. Current cost is stored in stdInverse. Notice that since we want to maximize the contrast but optimizer minimize cost function we use as cost 1/contrast

Private Members

dv::camera::CameraGeometry::SharedPtr **mCamera**

Camera geometry data. This information is used to create motionCompensator and compensate events.

const *dv::EventStore* **mEvents**

Raw events compensated using imu data.

float **mContribution**

Event contribution for total pixel intensity. This parameter is very important since it strongly influence contrast value. It needs to be tuned based on scene and length of event chunk.

const *std::vector<dv::IMU>* **mImuSamples**

Imu data used to compensate mEvents.

const *dv::kinematics::Transformationf* **mT_S_target**

Target (i.e. camera) to imu transformation. Used to construct rotationIntegrator that keeps track of camera position.

int64_t **mImuToTargetTimeOffsetUs**

Time offset between imu and target. Check rotationIntegrator class for more information.

```
struct RuntimeError : public dv::exceptions::info::EmptyException
```

```
template<dv::concepts::EventToFrameConverter<dv::EventStore> AccumulatorClass = dv::EdgeMapAccumulator>
```

```
class SemiDenseStereoMatcher
```

```
    #include </builds/inivation/dv/dv-processing/include/dv-processing/depth/semi_dense_stereo_matcher.hpp>
```

Semi-dense stereo matcher - a class that performs disparity calculation using an OpenCV dense disparity calculation algorithm. The implementation performs accumulation of a stereo pair of images of input events and applies the given stereo disparity matcher algorithm (semi-global block matching by default).

Public Functions

```
inline SemiDenseStereoMatcher (std::unique_ptr<AccumulatorClass> leftAccumulator,
                               std::unique_ptr<AccumulatorClass> rightAccumulator, const
                               std::shared_ptr<cv::StereoMatcher> &matcher =
                               cv::StereoSGBM::create())
```

Construct a semi dense stereo matcher object by providing custom accumulators for left and right cameras and a stereo matcher class.

Parameters

- **leftAccumulator** – *Accumulator* for the left camera.

- **rightAccumulator** – *Accumulator* for the right camera.
- **matcher** – Stereo matcher algorithm, if not provided, the implementation will use a `cv::StereoSGBM` class with default parameters.

```
inline explicit SemiDenseStereoMatcher (const cv::Size &leftResolution, const cv::Size &rightResolution,
                                         const std::shared_ptr<cv::StereoMatcher> &matcher =
                                         cv::StereoSGBM::create())
```

Construct a semi dense stereo matcher with default accumulator settings and a stereo matcher class.

Parameters

- **leftResolution** – Resolution of the left camera.
- **rightResolution** – Resolution of the right camera.
- **matcher** – Stereo matcher algorithm, if not provided, the implementation will use a `cv::StereoSGBM` class with default parameters.

```
inline explicit SemiDenseStereoMatcher (std::unique_ptr<dv::camera::StereoGeometry> geometry,
                                         std::shared_ptr<cv::StereoMatcher> matcher =
                                         dv::depth::defaultStereoMatcher())
```

Construct a semi dense stereo matcher with default accumulator settings and a stereo matcher class. The calibration information about camera will be extracted from the stereo geometry class instance.

Parameters

- **geometry** – Object describing the stereo camera geometry.
- **matcher** – Stereo matcher algorithm, if not provided, the implementation will use a `cv::StereoSGBM` class with optimized parameters.

```
inline SemiDenseStereoMatcher (std::unique_ptr<dv::camera::StereoGeometry> geometry,
                               std::unique_ptr<AccumulatorClass> leftAccumulator,
                               std::unique_ptr<AccumulatorClass> rightAccumulator,
                               std::shared_ptr<cv::StereoMatcher> matcher =
                               dv::depth::defaultStereoMatcher())
```

Construct a semi dense stereo matcher object by providing custom accumulators for left and right cameras and a stereo matcher class. The calibration information about camera will be extracted from the stereo geometry class instance.

Parameters

- **geometry** – Object describing the stereo camera geometry.
- **leftAccumulator** – *Accumulator* for the left camera.
- **rightAccumulator** – *Accumulator* for the right camera.
- **matcher** – Stereo matcher algorithm, if not provided, the implementation will use a `cv::StereoSGBM` class with optimized parameters.

```
inline cv::Mat computeDisparity (const dv::EventStore &left, const dv::EventStore &right)
```

Compute disparity of the two given event stores. The events will be accumulated using the accumulators for left and right camera accordingly and disparity is computed using the configured block matching algorithm. The function is not going to slice the input events, so event streams have to be synchronized and sliced accordingly. The `dv::StereoEventStreamSlicer` class is a good option for slicing stereo event streams.

NOTE: Accumulated frames will be rectified only if a stereo geometry class was provided during construction.

 See also

dv::StereoEventStreamSlicer for synchronized slicing of a stereo event stream.

Parameters

- **left** – Events from left camera.
- **right** – Events from right camera.

Returns

Disparity map computed by the configured block matcher.

```
inline cv::Mat compute (const cv::Mat &leftImage, const cv::Mat &rightImage) const
```

Compute stereo disparity given a time synchronized pair of images. Images will be rectified before computing disparity if a StereoGeometry class instance was provided.

Parameters

- **leftImage** – Left image of a stereo pair of images.
- **rightImage** – Right image of a stereo pair of images.

Returns

Disparity map computed by the configured block matcher.

```
inline const dv::Frame &getLeftFrame () const
```

Retrieve the accumulated frame from the left camera event stream.

Returns

An accumulated frame.

```
inline const dv::Frame &getRightFrame () const
```

Retrieve the accumulated frame from the right camera event stream.

Returns

An accumulated frame.

```
inline dv::DepthEventStore estimateDepth (const cv::Mat &disparity, const dv::EventStore &events, const float  
disparityScale = 16.f) const
```

Estimate depth given the disparity map and a list of events. The coordinates will be rectified and a disparity value will be looked up in the disparity map. The depth of each event is calculated using an equation: $\text{depth} = (\text{focalLength} * \text{baseline}) / \text{disparity}$. focalLength is expressed in pixels, baseline in meters.

For practical applications, depth estimation should be evaluated prior to any use. The directly estimated depth values can contain measurable errors which should be accounted for - the errors can usually be within 10-20% fixed absolute error distance. Usually this comes from various inaccuracies and can be mitigated by introducing a correction factor for the depth estimate.

Parameters

- **disparity** – Disparity map.
- **events** – Input events.
- **disparityScale** – Scale of disparity value in the disparity map, if subpixel accuracy is enabled in the block matching, this value will be equal to 16.

Returns

A depth event store, the events will contain the same information as in the input, but additionally

will have the depth value in meters. Events whose coordinates are outside of image bounds after rectification will be skipped.

```
inline dv::DepthFrame estimateDepthFrame (const cv::Mat &disparity, const float disparityScale = 16.f)
    const
```

Convert a disparity map into a depth frame. Each disparity value is converted into depth using the equation $\text{depth} = (\text{focalLength} * \text{baseline}) / (\text{disparity} * \text{pixelPitch})$. Output frame contains distance values expressed in integer values of millimeter distance.

Parameters

- **disparity** – Input disparity map.
- **disparityScale** – Scale of disparity value in the disparity map, if subpixel accuracy is enabled in the block matching, this value will be equal to 16.

Returns

A converted depth frame.

Protected Attributes

```
std::shared_ptr<cv::StereoMatcher> mMatcher = nullptr
```

```
std::unique_ptr<AccumulatorClass> mLeftAccumulator = nullptr
```

```
std::unique_ptr<AccumulatorClass> mRightAccumulator = nullptr
```

```
dv::Frame mLeftFrame
```

```
dv::Frame mRightFrame
```

```
std::unique_ptr<dv::camera::StereoGeometry> mStereoGeometry = nullptr
```

Private Functions

```
inline void validateStereoGeometry () const
```

Validates stereo geometry pointer, throws an error if the value is unset.

```
struct ShiftedSourceBias
```

On-chip shifted-source bias current configuration. See <https://docs.inivation.com/hardware/hardware-advanced-usage/biasing.html>²⁵ for more details.

Public Functions

```
constexpr ShiftedSourceBias () = default
```

```
inline constexpr ShiftedSourceBias (const uint8_t ref, const uint8_t reg, const
    ShiftedSourceBiasOperatingMode mode =
    ShiftedSourceBiasOperatingMode::SHIFTED_SOURCE)
```

Public Members

uint8_t **refValue** = {0}

Shifted-source bias level, from 0 to 63.

uint8_t **regValue** = {0}

Shifted-source bias current for buffer amplifier, from 0 to 63.

ShiftedSourceBiasOperatingMode **operatingMode** = {*ShiftedSourceBiasOperatingMode::SHIFTED_SOURCE*}

Shifted-source operating mode (see *ShiftedSourceBiasOperatingMode*).

ShiftedSourceBiasVoltageLevel **voltageLevel** = {*ShiftedSourceBiasVoltageLevel::SPLIT_GATE*}

Shifted-source voltage level (see *ShiftedSourceBiasVoltageLevel*).

class **SimpleFile**

Subclassed by *dv::io::SimpleReadOnlyFile*, *dv::io::SimpleWriteOnlyFile*

Public Functions

constexpr **SimpleFile**() = default

inline explicit **SimpleFile**(const *std::filesystem::path* &filePath, const *ModeFlags* modeFlags, const *WriteFlags* writeFlags = *WriteFlags::NONE*, const size_t bufferSize = 65536)

Open a file for reading and/or writing, supporting extra modes for writing and buffer control. Will always do what you expect and throw an exception if there's any issue.

Parameters

- **filePath** – file path to open.
- **modeFlags** – Open file for reading, writing or both.
- **writeFlags** – If opening for writing, extra flags for truncation and append modes.
- **bufferSize** – Size of user-space buffer for file operations. Default 64KB, use 0 to disable buffering entirely.

inline ~**SimpleFile**() noexcept

SimpleFile(const *SimpleFile* &file) = delete

SimpleFile &**operator**= (const *SimpleFile* &rhs) = delete

inline **SimpleFile**(*SimpleFile* &&file) noexcept

inline *SimpleFile* &**operator**= (*SimpleFile* &&rhs) noexcept

inline bool **isOpen**() const

inline void **flush**()

inline void **write**(const *std::string_view* data)

template<typename T>

²⁵ <https://docs.inivation.com/hardware/hardware-advanced-usage/biasing.html>

```

inline void write (const std::span<const T> data)

template<typename T>
inline void write (const T *elem, size_t num)

template<typename S, typename ...Args>
inline void format (const S &format, Args&&... args)

inline void readInto (std::string &data) const

template<typename T>
inline void readInto (std::vector<T> &data) const

template<typename T>
inline void readInto (T *elem, size_t num) const

inline std::vector<uint8_t> read (const size_t upToInBytes) const

inline std::vector<uint8_t> readAll () const

inline uint64_t tell () const

inline void seek (const uint64_t offsetInBytes, const SeekFlags flags = SeekFlags::START) const

inline void rewind () const

inline uint64_t fileSize () const

inline std::filesystem::path path () const

```

Private Functions

```

inline void close () noexcept

```

Private Members

```

std::FILE *f = {nullptr}

```

```

char *fBuffer = {nullptr}

```

```

std::filesystem::path fPath = {}

```

```

class SimpleReadOnlyFile : private dv::io::SimpleFile
    Subclassed by dv::io::ReadOnlyFile

```

Public Functions

```

constexpr SimpleReadOnlyFile () = default

inline explicit SimpleReadOnlyFile (const std::filesystem::path &filePath, const size_t bufferSize = 65536)

inline uint64_t fileSize () const

inline bool isOpen () const

```

```
inline std::filesystem::path path () const

inline std::vector<uint8_t> read (const size_t upToInBytes) const

inline std::vector<uint8_t> readAll () const

inline void readInto (std::string &data) const

template<typename T>
inline void readInto (std::vector<T> &data) const

template<typename T>
inline void readInto (T *elem, size_t num) const

inline void rewind () const

inline void seek (const uint64_t offsetInBytes, const SeekFlags flags = SeekFlags::START) const

inline uint64_t tell () const
```

class **SimpleWriteOnlyFile** : private *dv::io::SimpleFile*

Subclassed by *dv::io::WriteOnlyFile*

Public Functions

```
constexpr SimpleWriteOnlyFile () = default

inline explicit SimpleWriteOnlyFile (const std::filesystem::path &filePath, const WriteFlags writeFlags =
                                     WriteFlags::NONE, const size_t bufferSize = 65536)

inline uint64_t fileSize () const

inline void flush ()

template<typename S, typename ...Args>
inline void format (const S &format, Args&&... args)

inline bool isOpen () const

inline std::filesystem::path path () const

inline void rewind () const

inline void seek (const uint64_t offsetInBytes, const SeekFlags flags = SeekFlags::START) const

inline uint64_t tell () const

inline void write (const std::string_view data)

template<typename T>
inline void write (const std::span<const T> data)

template<typename T>
inline void write (const T *elem, size_t num)
```

struct **Sizes**

Public Members

uint64_t **mPacketElements** = {0}

uint64_t **mPacketSize** = {0}

uint64_t **mDataSize** = {0}

class **SliceJob**

#include </builds/inivation/dv/dv-processing/include/dv-processing/core/multi_stream_slicer.hpp> Internal container of slice jobs.

Public Types

enum class **SliceType**

Values:

enumerator **NUMBER**

enumerator **TIME**

using **JobCallback** = *std::function<void(const dv::TimeWindow&, const MapOfVariants&)>*

Callback method signature alias.

Public Functions

inline **SliceJob** (const int64_t intervalUS, *JobCallback* callback)

Create a slice job

Parameters

- **intervalUS** – Job execution interval in microseconds
- **callback** – The callback method

inline **SliceJob** (const size_t number, const *TimeSlicingApproach* slicing, *JobCallback* callback)

Create a slice by number job

Parameters

- **number** – Number of elements to be sliced
- **slicing** – Slicing method for gaps between numeric slices
- **callback** – The callback method

inline void **run** (const *dv::TimeWindow* &timeWindow, const *MapOfVariants* &data)

Public Members

SliceType **mType**

JobCallback **mCallback**

The callback method.

int64_t mInterval = -1

Job execution interval in microseconds.

size_t mNumberOfElements = 0

Slice by number configuration value.

TimeSlicingApproach **mTimeSlicing** = *TimeSlicingApproach::BACKWARD*

Time slicing method for slicing by number.

int64_t mLastEvaluatedTimestamp = 0

Timestamp specifying the last timestamp the job evaluated over.

class **SliceJob**

INTERNAL USE ONLY A single job of the EventStreamSlicer

Public Types

enum class **SliceType**

Values:

enumerator **NUMBER**

enumerator **TIME**

Public Functions

inline **SliceJob** (const *SliceType* type, const **int64_t** timeInterval, const **size_t** numberInterval, *std::function*<void(const *dv::TimeWindow*&, **PacketType**&)> callback)

INTERNAL USE ONLY Creates a new SliceJob of a certain type, interval and callback

Parameters

- **type** – The type of periodicity. Can be either **NUMBER** or **TIME**
- **timeInterval** – The interval at which the job should be executed
- **numberInterval** – The interval at which the job should be executed
- **callback** – The callback function to call on execution.

SliceJob () = default

inline void **run** (const **PacketType** &packet)

INTERNAL USE ONLY This function establishes how much fresh data is available and how often the callback can be executed on this fresh data. It then creates slices of the data and executes the callback as often as possible.

Parameters

packet – the storage packet to slice on.

```
inline void setTimeInterval (const int64_t timeInterval)
```

INTERNAL USE ONLY Sets the time interval to the supplied value

Parameters

timeInterval – the new time interval to use

```
inline void setNumberInterval (const size_t numberInterval)
```

INTERNAL USE ONLY Sets the number interval to the supplied value

Parameters

numberInterval – the new interval to use

Public Members

```
size_t mLastCallEnd = 0
```

Private Members

```
SliceType mType = SliceType::TIME
```

```
const std::function<void(const TimeWindow&, PacketType&> mCallback
```

```
int64_t mTimeInterval = 0
```

```
size_t mNumberInterval = 0
```

```
int64_t mLastCallEndTime = 0
```

Private Static Functions

```
template<class ElementVector>
```

```
static inline ElementVector sliceByNumber (const ElementVector &packet, const size_t fromIndex, const size_t  
number)
```

```
template<class ElementVector>
```

```
static inline ElementVector sliceByTime (const ElementVector &packet, const int64_t start, const int64_t end,  
size_t &endIndex)
```

```
class SocketBase
```

#include </builds/inivation/dv/dv-processing/include/dv-processing/io/network/socket_base.hpp> Interface class to define a socket API.

Subclassed by *dv::io::network::TCPTLSSocket*, *dv::io::network::UNIXSocket*

Public Types

```
using CompletionHandler = std::function<void(const boost::system::error_code&, const size_t)>
```

Callback alias that is used to handle a completed IO operation.

Public Functions

virtual **~SocketBase** () = default

virtual bool **isOpen** () const = 0

Check whether a socket is open and active.

Returns

True if socket is open, false otherwise.

virtual void **close** () = 0

Close the underlying socket communication. Async reads/writes can be aborted during this function call.

virtual void **write** (const asio::const_buffer &buffer, *CompletionHandler* &&handler) = 0

Write a data buffer to the socket asynchronously. Completion handler is called when write to the socket is complete.

Parameters

- **buffer** – Data buffer to written to the socket.
- **handler** – Completion handler, that is called when write is complete.

virtual void **read** (const asio::mutable_buffer &buffer, *CompletionHandler* &&handler) = 0

Read a data buffer from the socket asynchronously. Completion handler is called when read from the socket is complete.

Parameters

- **buffer** – Output buffer to place data from the socket.
- **wrHandler** – Completion handler, that is called when write is complete.

virtual void **syncWrite** (const asio::const_buffer &buffer) = 0

Write data into the socket synchronously, this method is a blocking call which returns when writing data is complete.

Parameters

- **buffer** – Data to be written into the socket.

virtual void **syncRead** (const asio::mutable_buffer &buffer) = 0

Read data from the socket synchronously, this method is a blocking call which returns when reading data is complete.

Parameters

- **buffer** – Output buffer to place data from the socket.

class **SortedPacketBuffers**

Public Functions

inline void **acceptData** (*parser::ParsedData* data)

inline void **clearBuffers** ()

inline *std::optional*<*dv::EventPacket*> **popEventPacket** ()

inline *std::optional*<*dv::Frame*> **popFrame** ()

inline *std::optional*<*dv::IMUPacket*> **popImuPacket** ()

```
inline std::optional<dv::TriggerPacket> popTriggerPacket ()
```

```
inline int64_t getEventSeekTime () const
```

Get latest timestamp of event data stream that has been read from the capture class.

Returns

Latest processed event timestamp; returns -1 if no data was processed or stream is unavailable.

```
inline int64_t getFrameSeekTime () const
```

Get latest timestamp of frames stream that has been read from the capture class.

Returns

Latest processed frame timestamp; returns -1 if no data was processed or stream is unavailable.

```
inline int64_t getImuSeekTime () const
```

Get latest timestamp of imu data that has been read from the capture class.

Returns

Latest processed imu data timestamp; returns -1 if no data was processed or stream is unavailable.

```
inline int64_t getTriggerSeekTime () const
```

Get latest timestamp of trigger data stream that has been read from the capture class.

Returns

Latest processed trigger timestamp; returns -1 if no data was processed or stream is unavailable.

```
inline std::variant<std::monostate, dv::EventPacket, dv::Frame, dv::IMUPacket, dv::TriggerPacket> readNextPacket ()
```

Private Types

```
using PacketCountType = uint64_t
```

Private Members

```
mutable std::mutex mDataLock
```

```
PacketCountType mPacketCount = 0
```

```
int64_t mEventStreamSeek = -1
```

```
int64_t mFrameStreamSeek = -1
```

```
int64_t mImuStreamSeek = -1
```

```
int64_t mTriggerStreamSeek = -1
```

```
boost::circular_buffer<std::pair<PacketCountType, dv::EventPacket>> mEvents = {256}
```

```
boost::circular_buffer<std::pair<PacketCountType, dv::Frame>> mFrames = {256}
```

```
boost::circular_buffer<std::pair<PacketCountType, dv::IMUPacket>> mImu = {256}
```

```
boost::circular_buffer<std::pair<PacketCountType, dv::TriggerPacket>> mTriggers = {256}
```

```
class SparseEventBlockMatcher
```

Public Functions

```
inline explicit SparseEventBlockMatcher (const cv::Size &resolution, const cv::Size &windowSize =  
                                         cv::Size(24, 24), const int32_t maxDisparity = 40, const int32_t  
                                         minDisparity = 0, const float minScore = 1.0f)
```

Initialize sparse event block matcher. This constructor initializes the matcher in non-rectified space, so for accurate results the event coordinates should be already rectified.

Parameters

- **resolution** – Resolution of camera sensors. Assumes same resolution for left and right camera.
- **windowSize** – Matching window size.
- **maxDisparity** – Maximum disparity value.
- **minDisparity** – Minimum disparity value.
- **minScore** – Minimum matching score to consider matching valid.

```
inline explicit SparseEventBlockMatcher (std::unique_ptr<dv::camera::StereoGeometry> geometry, const  
                                         cv::Size &windowSize = cv::Size(24, 24), const int32_t  
                                         maxDisparity = 40, const int32_t minDisparity = 0, const float  
                                         minScore = 1.0f)
```

Initialize a sparse stereo block matcher with a calibrated stereo geometry. This allows event rectification while calculating the disparity.

Parameters

- **geometry** – Stereo camera geometry.
- **windowSize** – Matching window size.
- **maxDisparity** – Maximum disparity value.
- **minDisparity** – Minimum disparity value.
- **minScore** – Minimum matching score to consider matching valid.

```
template<dv::concepts::Coordinate2DIterable InputPoints>
```

```
inline std::vector<PixelDisparity> computeDisparitySparse (const dv::EventStore &left, const dv::EventStore  
                                                           &right, const InputPoints &interestPoints)
```

Compute sparse disparity on given interest points. The events are accumulated sparsely only on the selected interest point regions. Returns a list of coordinates with their according disparity values, correlations and scores for each disparity match. If rectification is enabled, the returned disparity result will have `valid` flag set to false if the interest point coordinate lies outside of valid rectified pixel space.

Input event has to be passed in synchronized batches, no time validation is performed during accumulation.

Parameters

- **left** – Synchronised event batch from left camera.

- **right** – Synchronised event batch from right camera.
- **interestPoints** – List of interest coordinates in unrectified pixel space.

Returns

A list of disparity results for each input interest point.

```
inline const cv::Mat &getLeftMask () const
```

Get the left camera image mask. The algorithm only accumulates the frames where actual matching is going to happen. The mask will contain non-zero pixel values where accumulation needs to happen.

Returns

Interest region mask for left camera.

```
inline const cv::Mat &getRightMask () const
```

Get the right camera image mask. The algorithm only accumulates the frames where actual matching is going to happen. The mask will contain non-zero pixel values where accumulation needs to happen.

Returns

Interest region mask for right camera.

```
inline dv::Frame getLeftFrame () const
```

Get the latest accumulated left frame.

Returns

Accumulated image of the left camera from last disparity computation step.

```
inline dv::Frame getRightFrame () const
```

Get the latest accumulated right frame.

Returns

Accumulated image of the right camera from last disparity computation step.

```
inline const cv::Size &getWindowSize () const
```

Get matching window size.

Returns

Currently configured matching window size.

```
inline void setWindowSize (const cv::Size &windowSize)
```

Set matching window size. This is the size of cropped template image that is matched along the epipolar line of the stereo geometry.

Parameters

windowSize – New matching window size.

```
inline int32_t getMaxDisparity () const
```

Get maximum disparity value.

Returns

Currently configured maximum disparity value.

```
inline void setMaxDisparity (const int32_t maxDisparity)
```

Set maximum measured disparity. This parameter limits the matching space in pixels on the right camera image.

Parameters

maxDisparity – New maximum disparity value.

```
inline int32_t getMinDisparity () const
```

Get minimum disparity value.

Returns

Currently configured minimum disparity value.

```
inline void setMinDisparity (const int32_t minDisparity)
```

Set minimum measured disparity. This parameter limits the matching space in pixels on the right camera image.

Parameters

minDisparity – New minimum disparity value.

```
inline float getMinScore () const
```

Get minimum matching score value.

Returns

Currently configured minimum matching score value.

```
inline void setMinScore (const float minimumScore)
```

Set minimum matching score value to consider the matching valid. If matching score is below this threshold, the value for a point will be set to an invalid value and `valid` boolean to false.

Score is calculated by applying softmax function on the discrete distribution of correlation values from matching the template left patch on the epipolar line of the right camera image. This retrieves the probability mass function of the correlations. The best match is found by finding the max probability value and score is calculated for the best match by computing z-score over the probabilities.

Parameters

minimumScore – New minimum score value.

Protected Functions

```
template<dv::concepts::Coordinate2D InputPoint>
```

```
inline cv::Rect getPointRoi (const InputPoint &center, const int32_t offsetX, const int32_t stretchX) const
```

```
inline void initializeMaskPoint (cv::Mat &mask, const int32_t offsetX, const int32_t stretchX, const  
cv::Point2i &coord, const  
std::optional<dv::camera::StereoGeometry::CameraPosition>  
cameraPosition = std::nullopt) const
```

Protected Attributes

```
cv::Mat mLeftMask
```

```
cv::Mat mRightMask
```

```
dv::Frame mLeftFrame
```

```
dv::Frame mRightFrame
```

```
dv::EdgeMapAccumulator mLeftAcc
```

```
dv::EdgeMapAccumulator mRightAcc
```

```
cv::Size mWindowSize
```

```
cv::Size mHalfWindowSize
```

```
int32_t mMaxDisparity
```

```
int32_t mMinDisparity
```

```
float mMinScore
```

```
std::unique_ptr<dv::camera::StereoGeometry> mStereoGeometry = nullptr
```

```
template<class EventStoreType, uint32_t patchDiameter = 8, typename ScalarType = uint8_t>
```

```
class SpeedInvariantTimeSurfaceBase : public dv::TimeSurfaceBase<EventStoreType, uint8_t>
```

```
#include </builds/inivation/dv/dv-processing/include/dv-processing/core/core.hpp> A speed invariant time surface,  
as described by https://arxiv.org/abs/1903.11332
```

Template Parameters

- **EventStoreType** – Type of underlying event store
- **patchDiameter** – Diameter of the patch to apply the speed invariant update. The paper defines parameter r which is half of the diameter value, so for an $r = 5$, use diameter = $2 * r$ or 10 in this case. The update is performed using eigen optimized routines, so the value has limits: it has to be in range (0; 16) and divisible by 2. By default set to 8 which gives the best performance.

Public Functions

```
inline explicit SpeedInvariantTimeSurfaceBase (const cv::Size &resolution)
```

Create a speed invariant time surface with known image dimensions.

Parameters

resolution – Dimensions of the expected event data.

```
inline virtual SpeedInvariantTimeSurfaceBase &operator<< (const EventStoreType &store) override
```

Inserts the event store into the speed invariant time surface.

Parameters

store – The event store to be added

Returns

A reference to this TimeSurface.

```
inline virtual SpeedInvariantTimeSurfaceBase &operator<< (const typename  
EventStoreType::iterator::value_type &event)  
override
```

Inserts the event into the speed invariant time surface.

Parameters

event – The event to be added

Returns

A reference to this TimeSurface.

inline virtual void **accept** (const *EventStoreType* &store) override

Inserts the event store into the speed invariant time surface.

Parameters

store – The event store to be added

inline virtual void **accept** (const typename *EventStoreType*::iterator::value_type &event) override

Inserts the event into the speed invariant time surface.

Parameters

event – The event to be added

Protected Types

using **BaseClassType** = *TimeSurfaceBase*<*EventStoreType*, *ScalarType*>

Private Members

int64_t **mLatestPixelValue**

struct **SPIConfigurationParameters**

Public Members

uint8_t **moduleAddress**

uint8_t **parameterAddress**

uint32_t **parameterValue**

template<typename>

struct **std_function_exact**

std::function substitute with exact signature matching.

template<typename **R**, typename ...**Args**>

struct **std_function_exact**<*R*(*Args*...)> : public *std::function*<*R*(*Args*...)>

Public Functions

template<typename **T**, *std::enable_if_t*<*std::is_invocable_v*<*T*, *Args*...> && *std::is_same_v*<*R*,
std::invoke_result_t<*T*, *Args*...>, bool> = true>

inline **std_function_exact** (*T* &&t)

struct **StereoCalibration**

Public Functions

StereoCalibration () = default

inline explicit **StereoCalibration** (const *std*::string_view leftCameraName_, const *std*::string_view rightCameraName_, const *std*::span<const float> fundamentalMatrix_, const *std*::span<const float> essentialMatrix_, const *std*::optional<*Metadata*> &metadata_)

inline explicit **StereoCalibration** (const boost::property_tree::ptree &tree)

inline boost::property_tree::ptree **toPropertyTree** () const

inline bool **operator==** (const *StereoCalibration* &rhs) const

inline Eigen::Matrix3f **getFundamentalMatrix** () const

Retrieve the fundamental matrix as Eigen::Matrix3f.

Returns

Fundamental matrix.

inline Eigen::Matrix3f **getEssentialMatrix** () const

Retrieve the essential matrix as Eigen::Matrix3f.

Returns

Essential matrix.

Public Members

std::string **leftCameraName**

Name of the left camera.

std::string **rightCameraName**

Name of the right camera.

std::vector<float> **fundamentalMatrix**

Stereo calibration Fundamental Matrix.

std::vector<float> **essentialMatrix**

Stereo calibration Essential Matrix.

std::optional<*Metadata*> **metadata**

Metadata.

Friends

inline friend *std*::ostream &**operator<<** (*std*::ostream &os, const *StereoCalibration* &calibration)

class **StereoCameraRecording**

Public Functions

inline **StereoCameraRecording** (const *std::filesystem::path* &aedat4Path, const *std::string* &leftCameraName, const *std::string* &rightCameraName)

Create a reader for stereo camera recording. Expects at least one stream from two cameras available. Prior knowledge of stereo setup is required, otherwise it is not possible to differentiate between left and right cameras. This is just a convenience class that gives access to distinguished data streams in the recording.

Parameters

- **aedat4Path** – Path to the aedat4 file.
- **leftCameraName** – Name of the left camera.
- **rightCameraName** – Name of the right camera.

inline *MonoCameraRecording* &**getLeftReader** ()

Access the left camera.

Returns

A reference to the left camera reader.

inline *MonoCameraRecording* &**getRightReader** ()

Access the right camera.

Returns

A reference to the right camera reader.

Private Members

std::shared_ptr<ReadOnlyFile> **mReader** = nullptr

MonoCameraRecording **mLeftCamera**

MonoCameraRecording **mRightCamera**

class **StereoCameraWriter**

Public Functions

inline **StereoCameraWriter** (const *std::filesystem::path* &aedat4Path, const *MonoCameraWriter::Config* &leftConfig, const *MonoCameraWriter::Config* &rightConfig, const *dv::io::support::TypeResolver* &resolver = *dv::io::support::defaultTypeResolver*)

Open a file pass left / right camera configuration manually.

Parameters

- **aedat4Path** – Path to output file.
- **leftConfig** – Left camera output stream configuration.
- **rightConfig** – Right camera output stream configuration.
- **resolver** – Type resolver for the output file.

```
inline StereoCameraWriter (const std::filesystem::path &aedat4Path, camera::SyncCameraInputBase
    &leftCamera, camera::SyncCameraInputBase &rightCamera, const
    CompressionType compression = CompressionType::LZ4, const
    dv::io::support::TypeResolver &resolver = dv::io::support::defaultTypeResolver)
```

Open a file and use capture device to inspect the capabilities of the cameras. This will create all possible output streams the devices can supply.

Parameters

- **aedat4Path** – Path to output file.
- **leftCamera** – Capture object to inspect capabilities of the left camera.
- **rightCamera** – Capture object to inspect capabilities of the right camera.
- **compression** – Compression to be used for the output file.
- **resolver** – Type resolver for the output file.

Public Members

MonoCameraWriter **left**

Left writing instance.

MonoCameraWriter **right**

Right writing instance.

Private Functions

```
inline std::string createStereoHeader (const dv::io::support::TypeResolver &resolver)
```

```
inline void configureStreamIds ()
```

Private Members

MonoCameraWriter::Config **leftUpdatedConfig**

MonoCameraWriter::Config **rightUpdatedConfig**

StreamIdContainer **leftIds**

StreamIdContainer **rightIds**

MonoCameraWriter::StreamDescriptorMap **mLeftOutputStreamDescriptors**

MonoCameraWriter::StreamDescriptorMap **mRightOutputStreamDescriptors**

std::shared_ptr<WriteOnlyFile> **file**

Private Static Functions

```
static inline void configureCameraOutput (int32_t &index, dv::io::support::XMLTreeNode &mRoot,
                                           MonoCameraWriter::Config &config, const std::string
                                           &compression, StreamIdContainer &ids,
                                           MonoCameraWriter::StreamDescriptorMap &streamDescriptors,
                                           const dv::io::support::TypeResolver &resolver, const std::string
                                           &outputPrefix)
```

class **StereoGeometry**

#include </builds/inivation/dv/dv-processing/include/dv-processing/camera/stereo_geometry.hpp> A class that performs stereo geometry operations and rectification of a stereo camera.

Public Types

enum class **CameraPosition**

Position enum for a single camera in a stereo configuration.

Values:

enumerator **LEFT**

enumerator **RIGHT**

enum class **FunctionImplementation**

Values:

enumerator **LUT**

enumerator **SUB_PIXEL**

using **UniquePtr** = *std::unique_ptr<StereoGeometry>*

using **SharedPtr** = *std::shared_ptr<StereoGeometry>*

Public Functions

```
inline StereoGeometry (const CameraGeometry &leftCamera, const CameraGeometry &rightCamera, const
                        dv::kinematics::Transformationf &transformToLeft, std::optional<cv::Size>
                        rectifiedResolution = std::nullopt)
```

Initialize a stereo geometry class using two camera geometries for each of the stereo camera pair and a transformation matrix that describes the transformation from right camera to the left.

Parameters

- **leftCamera** – Left camera geometry.
- **rightCamera** – Right camera geometry.
- **transformToLeft** – A vector containing a homogenous transformation from right to the left camera. Vector should contain exactly 16 numbers (as per 4x4 homogenous transformation matrix) in a row-major ordering.

- **rectifiedResolution** – Resolution of the rectified image plane when remapping events/points/images from either the left or right camera (see [remapEvents\(\)](#)/[remapImage\(\)](#)). This can be the same, smaller, or larger than either the left or right camera resolutions, where downsampling/upsampling occurs if the `#rectifiedResolution` is smaller/larger than the camera resolution. Defaults to the left camera resolution if not provided.

```
inline StereoGeometry (const calibrations::CameraCalibration &leftCalibration, const
calibrations::CameraCalibration &rightCalibration, std::optional<cv::Size>
rectifiedResolution = std::nullopt)
```

Create a stereo geometry class from left and right camera calibration instances.

Parameters

- **leftCalibration** – Left camera calibration.
- **rightCalibration** – Right camera calibration.
- **rectifiedResolution** – Resolution of the rectified image plane when remapping events/points/images from either the left or right camera (see above constructor).

```
inline cv::Mat remapImage (const CameraPosition cameraPosition, const cv::Mat &image) const
```

Apply remapping to an input image to rectify it.

Parameters

- **cameraPosition** – Indication whether image is from left or right camera.
- **image** – Input image.

Returns

Rectified image.

```
inline dv::EventStore remapEvents (const CameraPosition cameraPosition, const dv::EventStore &events) const
```

Apply remapping on input events.

Parameters

- **cameraPosition** – Indication whether image is from left or right camera.
- **events** – Input events.

Returns

Event with rectified coordinates.

```
template<dv::concepts::Coordinate2DConstructible OutputPoint = cv::Point2i, dv::concepts::Coordinate2D
InputPoint>
```

```
inline std::optional<OutputPoint> remapPoint (const CameraPosition position, const InputPoint &point) const
```

Remap a point coordinate from original camera pixel space into undistorted and rectified pixel space.

Parameters

- **cameraPosition** – Camera position in the stereo setup.
- **point** – Coordinates in original camera pixel space.

Template Parameters

Point –

Returns

Undistorted and rectified coordinates or `std::nullopt` if the resulting coordinates are outside of valid output pixel range.

```
template<dv::concepts::Coordinate2DConstructible OutputPoint = cv::Point2i, FunctionImplementation
Implementation = FunctionImplementation::LUT, dv::concepts::Coordinate2D InputPoint>
```

inline *OutputPoint* **unmapPoint** (const *CameraPosition* position, const *InputPoint* &point) const

Unmap a point coordinate from undistorted and rectified pixel space into original distorted pixel.

Parameters

- **position** – Camera position in the stereo setup
- **point** – Coordinates in undistorted rectified pixel space.

Template Parameters

- **OutputPoint** – Output point class
- **Implementation** – Implementation type: LUT - performs a look-up operation on a pre-computed look-up table, SubPixel - performs full computations and retrieves exact coordinates.
- **InputPoint** – Input point class (automatically inferred)

Returns

Coordinates of the pixel in original pixel space.

inline *dv::camera::CameraGeometry* **getLeftCameraGeometry** () const

Retrieve left camera geometry class that can project coordinates into stereo rectified space.

Returns

Camera geometry instance.

inline *dv::camera::CameraGeometry* **getRightCameraGeometry** () const

Retrieve right camera geometry class that can project coordinates into stereo rectified space.

Returns

Camera geometry instance.

inline *dv::DepthEventStore* **estimateDepth** (const cv::Mat &disparity, const *dv::EventStore* &events, const float disparityScale = 16.f) const

Estimate depth given the disparity map and a list of events. The coordinates will be rectified and a disparity value will be looked up in the disparity map. The depth of each event is calculated using an equation: $\text{depth} = (\text{focalLength} * \text{baseline}) / \text{disparity}$. focalLength is expressed in pixels, baseline in meters.

For practical applications, depth estimation should be evaluated prior to any use. The directly estimated depth values can contain measurable errors which should be accounted for - the errors can usually be within 10-20% fixed absolute error distance. Usually this comes from various inaccuracies and can be mitigated by introducing a correction factor for the depth estimate.

Parameters

- **disparity** – Disparity map.
- **events** – Input events.
- **disparityScale** – Scale of disparity value in the disparity map, if subpixel accuracy is enabled in the block matching, this value will be equal to 16.

Returns

A depth event store, the events will contain the same information as in the input, but additionally will have the depth value in meters. Events whose coordinates are outside of image bounds after rectification will be skipped.

inline float **convertDisparityToDepth** (const float disparity) const

Convert a disparity value (in rectified coordinates) [px] to a depth value [mm] using the mapping $\text{depth} = (\text{focalLength} * \text{baseline}) / \text{disparity}$

Parameters

disparity – Input disparity value [px]

Returns

Corresponding depth value [mm]

inline *dv::DepthFrame* **toDepthFrame** (const cv::Mat &disparity, const float disparityScale = 16.f) const

Convert a disparity map into a depth frame. Each disparity value is converted into depth using the equation $\text{depth} = (\text{focalLength} * \text{baseline}) / \text{disparity}$. Output frame contains distance values expressed in integer values of millimeter distance.

NOTE: Output depth frame will not have a timestamp value, it is up to the user of this method to set correct timestamp of the disparity map.

Parameters

- **disparity** – Input disparity map.
- **disparityScale** – Scale of disparity value in the disparity map, if subpixel accuracy is enabled in the block matching, this value will be equal to 16.

Returns

A converted depth frame.

Public Static Functions

static inline *dv::kinematics::Transformationf* **computeTransformBetween** (const
calibrations::CameraCalibration
&src, const
calibrations::CameraCalibration
&target)

Compute the homogeneous transformation that transforms a point from a source camera to a target camera based on their respective calibrations.

Parameters

- **src** – Camera calibration for the source camera.
- **target** – Camera calibration for the target camera.

Returns

4x4 transformation from source to target.

Private Functions

inline void **createLUTs** (const cv::Size &resolution, const cv::Matx33f &cameraMatrix, const cv::Mat
 &distortion, const cv::Mat &R, const cv::Mat &P, *std::vector<uint8_t>* &outputMask,
std::vector<cv::Point2i> &outputRemapLUT) const

template<*concepts::Coordinate3DConstructible* **Output**, *concepts::Coordinate2D* **Input**>

inline *Output* **backProject** (const *StereoGeometry::CameraPosition* position, const *Input* &pixel) const

Private Members

cv::Mat **mLeftRemap1**

cv::Mat **mLeftRemap2**

```
cv::Mat mRightRemap1

cv::Mat mRightRemap2

cv::Mat mLeftProjection

cv::Mat mRightProjection

std::vector<uint8_t> mLeftValidMask

std::vector<uint8_t> mRightValidMask

std::vector<cv::Point2i> mLeftRemapLUT

std::vector<cv::Point2i> mRightRemapLUT

std::vector<cv::Point2i> mLeftUnmapLUT

std::vector<cv::Point2i> mRightUnmapLUT

cv::Size mLeftResolution

cv::Size mRightResolution

std::vector<float> mDistLeft

DistortionModel mLeftDistModel

std::vector<float> mDistRight

DistortionModel mRightDistModel

cv::Mat RN[2]

cv::Mat Q

dv::kinematics::Transformationf mLeftRectifierInverse

dv::kinematics::Transformationf mRightRectifierInverse

const dv::camera::CameraGeometry mOriginalLeft
```

```
const dv::camera::CameraGeometry mOriginalRight
```

```
float mBaselineFocal
```

Private Static Functions

```
template<dv::concepts::Coordinate2DConstructible PointType = cv::Point2d>
```

```
static inline std::vector<PointType> initCoordinateList (const cv::Size &resolution)
```

```
static inline dv::EventStore remapEventsInternal (const dv::EventStore &events, const cv::Size &resolution,
                                                  const std::vector<uint8_t> &mask, const
                                                  std::vector<cv::Point2i> &remapLUT)
```

Friends

```
inline friend std::ostream &operator<< (std::ostream &os, const CameraPosition &var)
```

```
inline friend std::ostream &operator<< (std::ostream &os, const FunctionImplementation &var)
```

```
struct Stream
```

#include </builds/inivation/dv/dv-processing/include/dv-processing/io/stream.hpp> Structure defining a stream of data. This class holds metadata information of a stream - id, name, source, resolution (if applicable), as well as data type, compression, and other technical information needed for application to be able send or receive streams of data.

Public Functions

Stream () = default

Default constructor with no information about the stream.

```
inline Stream (const int32_t id, const std::string_view name, const std::string_view sourceName, const
               std::string_view typeIdentifier, const dv::io::support::TypeResolver &resolver =
               dv::io::support::defaultTypeResolver)
```

Manual stream configuration.

Parameters

- **id** – *Stream* ID.
- **name** – Name of the stream.
- **sourceName** – *Stream* source, usually a camera name.
- **typeIdentifier** – Flatbuffer compiler generated type identifier string, unique for the stream type.
- **resolver** – Type resolver, supports default streams, used only for custom generated type support.

```
inline void addMetadata (const std::string &name, const dv::io::support::VariantValueOwning &value)
```

Add metadata to the stream. If an entry already exists, it will be replaced with the new value.

Parameters

- **name** – Name of the metadata entry.
- **value** – Metadata value.

```
inline std::optional<dv::io::support::VariantValueOwning> getMetadataValue (const std::string_view name)
                                                                    const
```

Get a metadata attribute value.

Parameters

name – Name of a metadata attribute.

Returns

Metadata value in a variant or *std::nullopt* if it's not found.

```
inline void setTypeDescription (const std::string &description)
```

Set type description. This only sets type description metadata field.

Parameters

description – Metadata string that describes the type in this stream.

```
inline void setModuleName (const std::string &moduleName)
```

Set module name that originally produces the data. This only sets the original module name metadata field.

Parameters

moduleName – Module name that originally produces the data.

```
inline void setOutputName (const std::string &outputName)
```

Set original output name. This only sets the original output name metadata field.

Parameters

outputName – Name of the output that produces the data, usually referring to DV module output.

```
inline void setCompression (const dv::CompressionType compression)
```

Set compression metadata field for this stream. This only sets the metadata field of this stream.

Parameters

compression – Type of compression.

```
inline std::optional<std::string> getTypeDescription () const
```

Get type description.

Returns

Type description string if available, *std::nullopt* otherwise.

```
inline std::optional<std::string> getModuleName () const
```

Get module name.

Returns

Module name string if available, *std::nullopt* otherwise.

```
inline std::optional<std::string> getOutputName () const
```

Get output name.

Returns

Output name string if available, *std::nullopt* otherwise.

```
inline std::optional<dv::CompressionType> getCompression () const
```

Get compression type string.

Returns

compression type string if available, *std::nullopt* otherwise.

inline void **setAttribute** (const *std::string_view* name, const *dv::io::support::VariantValueOwning* &value)

Set an attribute of this stream, if the attribute field does not exist, it will be created.

Parameters

- **name** – Name of the attribute.
- **value** – Attribute value.

inline *std::optional<dv::io::support::VariantValueOwning>* **getAttribute** (const *std::string_view* name) const

Get attribute value given it's name.

Parameters

name – Name of the attribute.

Returns

Return variant of the value if the an attribute with given name exists, *std::nullopt* otherwise.

template<typename **Type**>

inline *std::optional<Type>* **getAttributeValue** (const *std::string_view* name) const

Get attribute value given it's name.

Template Parameters

Type – Type of the attribute.

Parameters

name – Name of the attribute.

Returns

Return the attribute value if the an attribute with given name exists, *std::nullopt* otherwise.

inline *std::optional<cv::Size>* **getResolution** () const

Get resolution of this stream by parsing metadata.

Returns

Stream resolution or *std::nullopt* if resolution is not available.

inline void **setResolution** (const *cv::Size* &resolution)

Set the stream resolution in the metadata of this stream.

Parameters

resolution – *Stream* resolution.

inline *std::optional<std::string>* **getSource** () const

Get source name (usually the camera name) from metadata of the stream.

Returns

Stream source or *std::nullopt* if a source name is not available.

inline void **setSource** (const *std::string* &source)

Set a source name of this stream, usually camera name.

Parameters

source – Source name, usually camera name string.

Public Members

int32_t **mId** = 0

Stream ID.

std::string **mName**

Name of the stream.

std::string **mTypeIdentifier**

Stream type identifier.

dv::types::Type **mType**

Internal type definition.

dv::io::support::XMLTreeNode **mXMLNode**

XML tree node that can be used to encode information about the stream.

Public Static Functions

static inline *Stream* **EventStream** (const int32_t id, const *std::string* &name, const *std::string* &sourceName, const cv::Size &resolution)

Create an event stream.

Parameters

- **id** – *Stream* ID.
- **name** – Name of the stream.
- **sourceName** – *Stream* source, usually a camera name.
- **resolution** – Event sensor resolution.

Returns

Stream definition.

static inline *Stream* **FrameStream** (const int32_t id, const *std::string* &name, const *std::string* &sourceName, const cv::Size &resolution)

Create a frame stream.

Parameters

- **id** – *Stream* ID.
- **name** – Name of the stream.
- **sourceName** – *Stream* source, usually a camera name.
- **resolution** – *Frame* sensor resolution.

Returns

Stream definition.

static inline *Stream* **IMUStream** (const int32_t id, const *std::string* &name, const *std::string* &sourceName)

Create an *IMU* stream.

Parameters

- **id** – *Stream* ID.
- **name** – Name of the stream.
- **sourceName** – *Stream* source, usually a camera name.

Returns*Stream* definition.

```
static inline Stream TriggerStream (const int32_t id, const std::string &name, const std::string &sourceName)
```

Create an trigger stream.

Parameters

- **id** – *Stream* ID.
- **name** – Name of the stream.
- **sourceName** – *Stream* source, usually a camera name.

Returns*Stream* definition.

```
template<class Type>
static inline Stream TypedStream (const int32_t id, const std::string &name, const std::string &sourceName,
                                   const dv::io::support::TypeResolver &resolver =
                                   dv::io::support::defaultTypeResolver)
```

Create a stream by providing providing a stream type packet type as a template parameter.

Template Parameters**Type** – Type of the stream.**Parameters**

- **id** – *Stream* ID.
- **name** – Name of the stream.
- **sourceName** – *Stream* source, usually a camera name.
- **resolver** – Type resolver, supports default streams, used only for custom generated type support.

Returns*Stream* definition.

```
struct StreamDescriptor
```

Public Functions

```
inline explicit StreamDescriptor (const Stream &stream)
```

Public Members

```
size_t mSeekIndex = 0
```

```
dv::io::Stream mStream
```

```
std::map<std::string, std::string> mMetadata
```

```
struct StreamDescriptor
```

Public Functions

```
inline ~StreamDescriptor ()
```

```
inline StreamDescriptor (uint32_t id, const types::Type *type)
```

Public Members

```
uint32_t id
```

```
const dv::types::Type *type
```

```
int64_t lastTimestamp
```

```
void *elementBuffer
```

```
std::function<void(void*)> freeElementBufferCall = nullptr
```

```
struct StreamIdContainer
```

Public Members

```
int32_t mEventStreamId = -1
```

```
int32_t mImuStreamId = -1
```

```
int32_t mTriggerStreamId = -1
```

```
int32_t mFrameStreamId = -1
```

```
template<class PacketType>
```

```
class StreamSlicer
```

#include </builds/inivation/dv/dv-processing/include/dv-processing/core/stream_slicer.hpp> The *StreamSlicer* is a class that takes on incoming timestamped data, stores them in a minimal way and invokes functions at individual periods.

Public Functions

```
StreamSlicer () = default
```

```
inline void accept (const PacketType &data)
```

Add a full packet to the streaming buffer and evaluate jobs. This function copies the data over.

Parameters

data – the packet to be added to the buffer.

```
template<class ElementType>
```

inline void **accept** (const *ElementType* &element)

Adds a single element of a stream to the slicer buffer and evaluate jobs.

Parameters

element – the element to be added to the buffer

inline void **accept** (*PacketType* &&packet)

Adds full stream packet of data to the buffer and evaluates jobs.

Parameters

packet – the packet to be added to the buffer

inline int **doEveryNumberOfElements** (const size_t n, *std::function*<void(*PacketType*&)> callback)

Adds a number-of-elements triggered job to the Slicer. A job is defined by its interval and callback function. The slicer calls the callback function every *n* elements are added to the stream buffer, with the corresponding data. The (cpu) time interval between individual calls to the function depends on the physical event rate as well as the bulk sizes of the incoming data.

Parameters

- **n** – the interval (in number of elements) in which the callback should be called
- **callback** – the callback function that gets called on the data every interval

Returns

A handle to uniquely identify the job.

inline int **doEveryNumberOfElements** (const size_t n, *std::function*<void(const *dv::TimeWindow*&, *PacketType*&)> callback)

Adds a number-of-elements triggered job to the Slicer. A job is defined by its interval and callback function. The slicer calls the callback function every *n* elements are added to the stream buffer, with the corresponding data. The (cpu) time interval between individual calls to the function depends on the physical event rate as well as the bulk sizes of the incoming data.

Parameters

- **n** – the interval (in number of elements) in which the callback should be called
- **callback** – the callback function that gets called on the data every interval, also passes time window containing the inter

Returns

A handle to uniquely identify the job.

inline int **doEveryTimeInterval** (const *dv::Duration* interval, *std::function*<void(const *PacketType*&)> callback)

Adds an element-timestamp-interval triggered job to the Slicer. A job is defined by its interval and callback function. The slicer calls the callback whenever the timestamp difference of an incoming event to the last time the function was called is bigger than the interval. As the timing is based on event times rather than CPU time, the actual time periods are not guaranteed, especially with a low event rate. The (cpu) time interval between individual calls to the function depends on the physical event rate as well as the bulk sizes of the incoming data.

Parameters

- **interval** – the interval in which the callback should be called
- **callback** – the callback function that gets called on the data every interval

Returns

A handle to uniquely identify the job.

```
inline int doEveryTimeInterval (const dv::Duration interval, std::function<void(const dv::TimeWindow&,
const PacketType&)> callback)
```

Adds an element-timestamp-interval triggered job to the Slicer. A job is defined by its interval and callback function. The slicer calls the callback whenever the timestamp difference of an incoming event to the last time the function was called is bigger than the interval. As the timing is based on event times rather than CPU time, the actual time periods are not guaranteed, especially with a low event rate. The (cpu) time interval between individual calls to the function depends on the physical event rate as well as the bulk sizes of the incoming data.

Parameters

- **interval** – the interval in which the callback should be called
- **callback** – the callback function that gets called with the time window information and the data as arguments every interval

Returns

An id to uniquely identify the job.

```
inline bool hasJob (const int jobId) const
```

Returns true if the slicer contains the slicejob with the provided id

Parameters

jobId – the id of the slicejob in question

Returns

true, if the slicer contains the given slicejob

```
inline void removeJob (const int jobId)
```

Removes the given job from the list of current jobs.

Parameters

jobId – The job id to be removed

```
inline void modifyTimeInterval (const int jobId, const dv::Duration timeInterval)
```

Modifies the time interval of the supplied job to the requested value

Parameters

- **jobId** – the job whose time interval should be changed
- **timeInterval** – the new time interval value

```
inline void modifyNumberInterval (const int jobId, const size_t numberInterval)
```

Modifies the number interval of the supplied job to the requested value

Parameters

- **jobId** – the job whose number interval should be changed
- **numberInterval** – the new number interval value

Private Functions

```
inline void evaluate ()
```

Should get called as soon as there is fresh data available. It loops through all jobs and determines if they can run on the new data. The jobs get executed as often as possible. Afterwards, all data that has been processed by all jobs gets discarded.

Private Members

PacketType **mStorePacket**

Global storage packet that holds just as many data elements as minimally required for all outstanding calls.

std::map<int, SliceJob> **mSliceJobs**

List of all the sliceJobs.

int **mHashCounter** = 0

class **SyncCameraInputBase** : public *dv::io::camera::CameraInputBase*

#include </builds/inivation/dv/dv-processing/include/dv-processing/io/camera/sync_camera_input_base.hpp> Camera input base class to abstract live camera and recorded files with a common interface.

Subclassed by *dv::io::camera::DAVIS*, *dv::io::camera::DVS128*, *dv::io::camera::DVXplorer*

Public Functions

virtual bool **isMaster** () const = 0

Report if this camera is a clock synchronization master.

Returns

true if clock master, false otherwise.

inline bool **isSynchronized** () const

Report if this camera is properly synchronized with others in a multi-camera scenario.

Returns

the camera is synchronized with others.

inline void **synchronizeWith** (const *std::unique_ptr<SyncCameraInputBase>* &secondary)

Synchronize this camera with another. This camera is expected to be the clock synchronization master.

Parameters

secondary – secondary camera.

inline void **synchronizeWith** (*SyncCameraInputBase* &secondary)

Synchronize this camera with another. This camera is expected to be the clock synchronization master.

Parameters

secondary – secondary camera.

inline void **synchronizeWith** (*SyncCameraInputBase* *secondary)

Synchronize this camera with another. This camera is expected to be the clock synchronization master.

Parameters

secondary – secondary camera.

inline void **synchronizeWith** (const *std::span<SyncCameraInputBase*>* secondaryCameras)

Synchronize this camera with multiple others. This camera is expected to be the clock synchronization master.

Parameters

secondaryCameras – secondary cameras (one or more).

Protected Types

enum class **SyncState**

Values:

enumerator **WAIT_RESET**

enumerator **GOT_RESET**

enumerator **SYNC_OK**

Protected Functions

virtual void **sendTimestampReset** () = 0

Send a timestamp reset command to the device.

virtual void **setTimestampOffset** (*std::chrono::microseconds* timestampOffset) = 0

Set a new timestamp offset value for the camera.

Parameters

timestampOffset – New timestamp offset value in microseconds.

inline void **waitForTimestampReset** () const

Wait until a timestamp reset is observed from this camera.

inline bool **gotReset** () const

Report if we observed the timestamp reset from the device.

Returns

wether we got a timestamp reset or not.

Protected Attributes

std::atomic<SyncState> **mSyncState** = {*SyncState::WAIT_RESET*}

Protected Static Attributes

static constexpr *std::chrono::microseconds* **TIME_SYNC_TIMEOUT** = {5'000'000}

Friends

friend class dv_capture_node::CaptureNode

friend class dv_runtime::TimeSync

class **TCPTLSSocket** : public *dv::io::network::SocketBase*

#include </builds/inivation/dv/dv-processing/include/dv-processing/io/network/tcp_tls_socket.hpp> Minimal wrapper of TCP socket with optional TLS encryption.

Public Types

using **socketType** = *asioTCP*::socket

Public Functions

inline **TCPTLSSocket** (*asioTCP*::socket &&socket, const bool tlsEnabled, const asioSSL::stream_base::handshake_type tlsHandshake, asioSSL::context &tlsContext)

Create a TCP socket with optional TLS encryption.

Parameters

- **socket** – A connected TCP socket instance.
- **tlsEnabled** – Whether TLS encryption is enabled, if true, TLS handshake will be immediately performed during construction.
- **tlsHandshake** – Type of TLS handshake, this is ignored if TLS is disabled.
- **tlsContext** – Pre-configured TLS context for encryption.

inline **~TCPTLSSocket** () override

inline virtual bool **isOpen** () const override

Check whether socket is open and active.

Returns

True if socket is open, false otherwise.

inline bool **isSecured** () const

Check whether socket has encryption enabled.

Returns

True if socket has encryption enabled, false otherwise.

inline virtual void **close** () override

Close underlying TCP socket cleanly.

inline virtual void **write** (const asio::const_buffer &buf, *SocketBase::CompletionHandler* &&wrHandler) override

Write handler needs following signature: void (const boost::system::error_code &, size_t)

inline virtual void **read** (const asio::mutable_buffer &buf, *SocketBase::CompletionHandler* &&rdHandler) override

Read handler needs following signature: void (const boost::system::error_code &, size_t)

inline virtual void **syncWrite** (const asio::const_buffer &buf) override

Blocking write data to the socket.

Parameters

buf – Data to write.

inline virtual void **syncRead** (const asio::mutable_buffer &buf) override

Blocking read from socket.

Parameters

buf – Buffer for data to be read into.

```
inline asioTCP::endpoint local_endpoint () const
```

Retrieve local endpoint.

Returns

Local endpoint.

```
inline asioIP::address local_address () const
```

Get the local address of the current endpoint.

Returns

IP address of the local connection.

```
inline uint16_t local_port () const
```

Get local port number.

Returns

Local port number.

```
inline asioTCP::endpoint remote_endpoint () const
```

```
inline asioIP::address remote_address () const
```

Remote endpoint IP address.

Returns

Remote endpoint IP address.

```
inline uint16_t remote_port () const
```

Get remote endpoint port number.

Returns

Remote endpoint port number.

Private Functions

```
inline asioTCP::socket &baseSocket ()
```

Private Members

```
asioTCP::endpoint mLocalEndpoint
```

```
asioTCP::endpoint mRemoteEndpoint
```

```
asioSSL::stream<asioTCP::socket> mSocket
```

```
bool mSocketClosed = false
```

```
bool mSecureConnection = false
```

```
class ThreadExtra
```

Public Static Functions

```
static inline std::string getName ()

static inline bool setName (const std::string &name)

static inline bool setPriorityHighest ()
```

```
class ThreadNameSwitch
```

Public Functions

```
inline explicit ThreadNameSwitch (const std::string &temporaryName)

inline ~ThreadNameSwitch ()
```

Private Members

```
std::string originalName
```

```
struct TimedKeyPoint : public flatbuffers::NativeTable
```

Public Types

```
typedef TimedKeyPointFlatbuffer TableType
```

Public Functions

```
inline TimedKeyPoint ()

inline TimedKeyPoint (const Point2f &_pt, float _size, float _angle, float _response, int32_t _octave, int32_t
    _class_id, int64_t _timestamp)
```

Public Members

```
Point2f pt
```

```
float size
```

```
float angle
```

```
float response
```

```
int32_t octave
```

```
int32_t class_id
```

```
int64_t timestamp
```

Public Static Functions

```
static inline constexpr const char *GetFullyQualifiedName ()
```

```
struct TimedKeyPointBuilder
```

Public Functions

```
inline void add_pt (const Point2f *pt)

inline void add_size (float size)

inline void add_angle (float angle)

inline void add_response (float response)

inline void add_octave (int32_t octave)

inline void add_class_id (int32_t class_id)

inline void add_timestamp (int64_t timestamp)

inline explicit TimedKeyPointBuilder (flatbuffers::FlatBufferBuilder &_fbb)

TimedKeyPointBuilder &operator= (const TimedKeyPointBuilder &)

inline flatbuffers::Offset<TimedKeyPointFlatbuffer> Finish ()
```

Public Members

```
flatbuffers::FlatBufferBuilder &fbb_

flatbuffers::uoffset_t start_
```

```
struct TimedKeyPointFlatbuffer : private flatbuffers::Table
```

Public Types

```
typedef TimedKeyPoint NativeTableType
```

Public Functions

```
inline const Point2f *pt () const
    coordinates of the keypoints.

inline float size () const
    diameter of the meaningful keypoint neighborhood.

inline float angle () const
    computed orientation of the keypoint (-1 if not applicable); it's in [0,360) degrees and measured relative to
    image coordinate system, ie in clockwise.
```

```

inline float response () const
    the response by which the most strong keypoints have been selected. Can be used for the further sorting or
    subsampling.

inline int32_t octave () const
    octave (pyramid layer) from which the keypoint has been extracted.

inline int32_t class_id () const
    object class (if the keypoints need to be clustered by an object they belong to).

inline int64_t timestamp () const
    Timestamp (µs).

inline bool Verify (flatbuffers::Verifier &verifier) const

inline TimedKeyPoint *UnPack (const flatbuffers::resolver_function_t *_resolver = nullptr) const

inline void UnPackTo (TimedKeyPoint *_o, const flatbuffers::resolver_function_t *_resolver = nullptr) const

```

Public Static Functions

```

static inline const flatbuffers::TypeTable *MiniReflectTypeTable ()

static inline constexpr const char *GetFullyQualifiedName ()

static inline void UnPackToFrom (TimedKeyPoint *_o, const TimedKeyPointFlatbuffer *_fb, const
    flatbuffers::resolver_function_t *_resolver = nullptr)

static inline flatbuffers::Offset<TimedKeyPointFlatbuffer> Pack (flatbuffers::FlatBufferBuilder &_fbb, const
    TimedKeyPoint *_o, const
    flatbuffers::rehasher_function_t *_rehasher =
    nullptr)

```

```

struct TimedKeyPointPacket : public flatbuffers::NativeTable

```

Public Types

```

typedef TimedKeyPointPacketFlatbuffer TableType

```

Public Functions

```

inline TimedKeyPointPacket ()

inline TimedKeyPointPacket (const std::vector<TimedKeyPoint> &_elements)

```

Public Members

```

std::vector<TimedKeyPoint> elements

```

Public Static Functions

```

static inline constexpr const char *GetFullyQualifiedName ()

```

Friends

```
inline friend std::ostream &operator<< (std::ostream &os, const TimedKeyPointPacket &packet)
```

```
struct TimedKeyPointPacketBuilder
```

Public Functions

```
inline void add_elements (flatbuffers::Offset<flatbuffers::Vector<flatbuffers::Offset<TimedKeyPointFlatbuffer>>>
                        elements)
```

```
inline explicit TimedKeyPointPacketBuilder (flatbuffers::FlatBufferBuilder &fbb)
```

```
TimedKeyPointPacketBuilder &operator= (const TimedKeyPointPacketBuilder&)
```

```
inline flatbuffers::Offset<TimedKeyPointPacketFlatbuffer> Finish ()
```

Public Members

```
flatbuffers::FlatBufferBuilder &fbb_
```

```
flatbuffers::uoffset_t start_
```

```
struct TimedKeyPointPacketFlatbuffer : private flatbuffers::Table
```

Public Types

```
typedef TimedKeyPointPacket NativeTableType
```

Public Functions

```
inline const flatbuffers::Vector<flatbuffers::Offset<TimedKeyPointFlatbuffer>> *elements () const
```

```
inline bool Verify (flatbuffers::Verifier &verifier) const
```

```
inline TimedKeyPointPacket *UnPack (const flatbuffers::resolver_function_t *_resolver = nullptr) const
```

```
inline void UnPackTo (TimedKeyPointPacket *_o, const flatbuffers::resolver_function_t *_resolver = nullptr)
const
```

Public Static Functions

```
static inline const flatbuffers::TypeTable *MiniReflectTypeTable ()
```

```
static inline constexpr const char *GetFullyQualifiedName ()
```

```
static inline void UnPackToFrom (TimedKeyPointPacket *_o, const TimedKeyPointPacketFlatbuffer *_fb, const
                                flatbuffers::resolver_function_t *_resolver = nullptr)
```

```
static inline flatbuffers::Offset<TimedKeyPointPacketFlatbuffer> Pack (flatbuffers::FlatBufferBuilder &fbb,
                                                                    const TimedKeyPointPacket *_o, const
                                                                    flatbuffers::rehasher_function_t
                                                                    *_rehasher = nullptr)
```

Public Static Attributes

```
static constexpr const char *identifier = "TKPS"
```

```
struct TimeElementExtractor
```

Public Functions

```
inline constexpr TimeElementExtractor () noexcept
```

```
inline constexpr TimeElementExtractor (const int64_t startTimestamp_, const int64_t endTimestamp_)
noexcept
```

```
~TimeElementExtractor () = default
```

```
TimeElementExtractor (const TimeElementExtractor &t) = default
```

```
TimeElementExtractor &operator= (const TimeElementExtractor &rhs) = default
```

```
TimeElementExtractor (TimeElementExtractor &&t) = default
```

```
TimeElementExtractor &operator= (TimeElementExtractor &&rhs) = default
```

```
inline constexpr bool operator== (const TimeElementExtractor &rhs) const noexcept
```

```
inline constexpr bool operator!= (const TimeElementExtractor &rhs) const noexcept
```

Public Members

```
int64_t startTimestamp
```

```
int64_t endTimestamp
```

```
int64_t numElements
```

```
template<class EventStoreType, typename ScalarType = int64_t>
```

```
class TimeSurfaceBase
```

#include </builds/inivation/dv/dv-processing/include/dv-processing/core/core.hpp> TimeSurface class that builds the surface of the occurrences of the last timestamps.

Subclassed by *dv::SpeedInvariantTimeSurfaceBase< EventStoreType, patchDiameter, ScalarType >*

Public Types

```
using Scalar = ScalarType
```

Public Functions

```
TimeSurfaceBase () = default
```

Dummy constructor Constructs a new, empty TimeSurface without any data allocated to it.

inline explicit **TimeSurfaceBase** (const uint32_t rows, const uint32_t cols)

Creates a new TimeSurface with the given size. The Mat is zero initialized

Parameters

- **rows** – The number of rows of the TimeSurface
- **cols** – The number of cols of the TimeSurface

inline explicit **TimeSurfaceBase** (const cv::Size &size)

Creates a new TimeSurface of the given size. The Mat is zero initialized.

Parameters

size – The opencv size to be used to initialize

TimeSurfaceBase (const *TimeSurfaceBase* &other) = default

Copy constructor, constructs a new time surface with shared ownership of the data.

Parameters

other – The time surface to be copied. The data is not copied but takes shared ownership.

virtual ~**TimeSurfaceBase** () = default

Destructor

inline virtual *TimeSurfaceBase* &**operator<<** (const *EventStoreType* &store)

Inserts the event store into the time surface.

Parameters

store – The event store to be added

Returns

A reference to this *TimeSurfaceBase*.

inline virtual *TimeSurfaceBase* &**operator<<** (const typename *EventStoreType*::iterator::value_type &event)

Inserts the event into the time surface.

Parameters

event – The event to be added

Returns

A reference to this *TimeSurfaceBase*.

inline *dv::Frame* &**operator>>** (*dv::Frame* &mat) const

Generates a frame from the data contained in the event store

Parameters

mat – The storage where the frame should be generated

Returns

A reference to the generated frame.

inline virtual void **accept** (const *EventStoreType* &store)

Inserts the event store into the time surface.

Parameters

store – The event store to be added

inline virtual void **accept** (const typename *EventStoreType*::iterator::value_type &event)

Inserts the event into the time surface.

Parameters

event – The event to be added

inline const *ScalarType* &**at** (const int16_t y, const int16_t x) const

Returns a const reference to the element at the given coordinates. The element can only be read from

Parameters

- **y** – The y coordinate of the element to be accessed.
- **x** – The x coordinate of the element to be accessed.

Returns

A const reference to the element at the requested coordinates.

inline *ScalarType* &**at** (const int16_t y, const int16_t x)

Returns a reference to the element at the given coordinates. The element can both be read from as well as written to.

Parameters

- **y** – The y coordinate of the element to be accessed.
- **x** – The x coordinate of the element to be accessed.

Returns

A reference to the element at the requested coordinates.

inline const *ScalarType* &**operator** () (const int16_t y, const int16_t x) const noexcept

Returns a const reference to the element at the given coordinates. The element can only be read from

Parameters

- **y** – The y coordinate of the element to be accessed.
- **x** – The x coordinate of the element to be accessed.

Returns

A const reference to the element at the requested coordinates.

inline *ScalarType* &**operator** () (const int16_t y, const int16_t x) noexcept

Returns a reference to the element at the given coordinates. The element can both be read from as well as written to.

Parameters

- **y** – The y coordinate of the element to be accessed.
- **x** – The x coordinate of the element to be accessed.

Returns

A reference to the element at the requested coordinates.

inline auto **block** (const int16_t topRow, const int16_t leftCol, const int16_t height, const int16_t width) const

Returns a block of the time surface

Parameters

- **topRow** – the row coordinate at the top of the block
- **leftCol** – the column coordinate at the left of the block
- **height** – the height of the block
- **width** – the width of the block

Returns

the block

```
inline auto block (const int16_t topRow, const int16_t leftCol, const int16_t height, const int16_t width)
```

Returns a block of the time surface

Parameters

- **topRow** – the row coordinate at the top of the block
- **leftCol** – the column coordinate at the left of the block
- **height** – the height of the block
- **width** – the width of the block

Returns

the block

```
inline dv::Frame generateFrame () const
```

Generates a frame from the data contained in the event store

Returns

The generated frame.

```
template<class T = uint8_t>
```

```
inline std::pair<cv::Mat, int64_t> getOCVMat () const
```

Creates a new OpenCV matrix of the type given and copies the time data into this OpenCV matrix. This version does only subtracts an offset from the values for them to fit into the value range of the requested frame type. Therefore this method preserves the units of the timestamps that are contained in the time surface.

The data in the time surface is of signed 64bit integer type. There is no OpenCV type that can hold the full range of these values. Therefore, the returned data is a pair of an OpenCV Mat, of a type that can be chosen by the user, and an offset of signed 64bit integer, which contains the offset that can be added to each pixel value so that their values are in units of microseconds.

Template Parameters

T – The type of the OpenCV Mat to be generated.

Returns

An OpenCV Mat of the requested type, as well as an offset which can be added to the matrix in order for the data to be in microseconds.

```
template<typename T = uint8_t>
```

```
inline cv::Mat getOCVMatScaled (const std::optional<int64_t> lookBackOverride = std::nullopt) const
```

Creates a new OpenCV matrix of the type given and copies the time data into this OpenCV matrix. This version scales the values for them to fit into the value range of the requested frame type. Therefore the units of the timestamps are not preserved.

The data in the time surface is of signed 64bit integer type. There is no OpenCV type that can hold the full range of these values. Therefore, the returned data is a pair of an OpenCV Mat, of a type that can be chosen by the user, and an offset of signed 64bit integer, which contains the offset that can be added to each pixel value so that their values are in units of microseconds.

Template Parameters

T – The type of the OpenCV Mat to be generated.

Parameters

lookBackOverride – override the amount of time to look back into the past. Defaults to the complete range contained in the time surface. The unit of the parameter is the unit of time contained in the TimeSurface.

Returns

An OpenCV Mat of the requested type, as well as an offset which can be added to the matrix in order for the data to be in microseconds.

```
inline void reset ()
```

Sets all values in the time surface to zero

```
template<typename T>
```

```
inline TimeSurfaceBase operator+ (const T &s) const
```

Adds a constant to the time surface. Values are bounds checked to 0. If the new time would become negative, it is set to 0.

Template Parameters

T – The type of the constant. Accepts any numeric type.

Parameters

s – The constant to be added

Returns

A new *TimeSurfaceBase* with the changed times

```
template<typename T>
```

```
inline TimeSurfaceBase &operator+= (const T &s)
```

Adds a constant to the TimeSurface. Values are bounds checked to 0. If the new time would become negative, it is set to 0.

Template Parameters

T – The type of the constant. Accepts any numeric type.

Parameters

s – The constant to be added

Returns

A reference to the *TimeSurfaceBase*

```
template<typename T>
```

```
inline TimeSurfaceBase operator- (const T &s) const
```

Subtracts a constant from the TimeSurface. Values are bounds checked to 0. If the new time would become negative, it is set to 0.

Template Parameters

T – The type of the constant. Accepts any numeric type.

Parameters

s – The constant to be subtracted

Returns

A reference to the *TimeSurfaceBase*

```
template<typename T>
```

```
inline TimeSurfaceBase &operator-= (const T &s)
```

Subtracts a constant from the TimeSurface. Values are bounds checked to 0. If the new time would become negative, it is set to 0.

Template Parameters

T – The type of the constant. Accepts any numeric type.

Parameters

s – The constant to be subtracted

Returns

A reference to the *TimeSurfaceBase*

```
template<typename T>
```

```
inline TimeSurfaceBase &operator= (const T &s)
```

Assigns constant to the TimeSurface. Values are bounds checked to 0. If the new time would become negative, it is set to 0.

Template Parameters

T – The type of the constant. Accepts any numeric type.

Parameters

s – The constant to be subtracted

Returns

A reference to the *TimeSurfaceBase*

```
inline cv::Size size () const noexcept
```

The size of the TimeSurface.

Returns

Returns the size of this time matrix as an opencv size

```
inline int16_t rows () const noexcept
```

Returns the number of rows of the TimeSurface

Returns

the number of rows

```
inline int16_t cols () const noexcept
```

Returns the number of columns of the TimeSurface

Returns

the number of columns

```
inline bool isEmpty () const noexcept
```

Returns true if the TimeSurface has zero size. In this case, it was not allocated with a size.

Returns

true if the TimeSurface does not have a size > 0

Protected Functions

```
inline void addImpl (const ScalarType a, TimeSurfaceBase &target) const
```

Protected Attributes

```
Eigen::Matrix<ScalarType, Eigen::Dynamic, Eigen::Dynamic> mData
```

```
struct TimeWindow
```

Public Functions

```
inline TimeWindow (const int64_t timestamp, const dv::Duration duration)
```

```
inline TimeWindow (const int64_t startTime, const int64_t endTime)
```

```
inline dv::Duration duration () const
```

Public Members

int64_t **startTime**

int64_t **endTime**

class **TrackerBase**

#include </builds/inivation/dv/dv-processing/include/dv-processing/features/tracker_base.hpp> A base class for implementing feature trackers, that track sets of features against streams of various inputs. This class specifically does not define an input type, so it could be defined by the specific implementation.

Subclassed by *dv::features::ImageFeatureLKTracker*, *dv::features::MeanShiftTracker*

Public Types

typedef *std::shared_ptr<TrackerBase>* **SharedPtr**

typedef *std::unique_ptr<TrackerBase>* **UniquePtr**

Public Functions

inline void **setMaxTracks** (size_t _maxTracks)

Set the maximum number of tracks.

Parameters

_maxTracks – Maximum number of tracks

inline size_t **getMaxTracks** () const

Get the maximum number of tracks.

Returns

Maximum number of tracks

inline const *Result::SharedPtr* &**getLastFrameResults** () const

Retrieve cached last frame detection results.

Returns

Detection result from the last processed frame.

inline *Result::ConstPtr* **runTracking** ()

Performed the tracking and cache the results.

Returns

Tracking result.

virtual ~**TrackerBase** () = default

inline virtual void **removeTracks** (const *std::vector<int>* &trackIds)

Remove tracks from cached results, so the wouldn't be tracked anymore. TrackIds are the *class_id* value of the keypoint structure.

Parameters

trackIds – Track *class_id* values to be removed from cached tracker results.

Protected Functions

virtual *Result::SharedPtr* **track**() = 0

Virtual function that is called after all inputs were set. This function should perform tracking against `lastFrameResults`.

Returns

Tracking result.

Protected Attributes

size_t **maxTracks** = 200

Maximum number of tracks.

Result::SharedPtr **lastFrameResults**

Cached results of last tracker execution.

template<std::floating_point **Scalar**>

class **Transformation**

#include </builds/inivation/dv/dv-processing/include/dv-processing/kinematics/transformation.hpp> Basic transformation wrapper containing homogenous 3D transformation and timestamp.

Template Parameters

Scalar – Customizable storage type - float or double.

Public Functions

inline EIGEN_MAKE_ALIGNED_OPERATOR_NEW **Transformation**(int64_t timestamp, const Eigen::Matrix<*Scalar*, 4, 4> &T)

Construct the transformation from a timestamp and 4x4 transformation matrix

Parameters

- **timestamp** – Unix timestamp in microsecond format
- **T** – Homogenous 3D transformation matrix

inline **Transformation**()

Construct an identity transformation from with timestamp.

Parameters

timestamp – Unix timestamp in microsecond format

inline **Transformation**(int64_t timestamp, const Eigen::Matrix<*Scalar*, 3, 1> &translation, const Eigen::Quaternion<*Scalar*> &rotation)

Construct the transformation from timestamp, 3D translation vector and quaternion describing the rotation.

Parameters

- **timestamp** – Unix timestamp in microsecond format
- **translation** – 3D translation vector
- **rotation** – Quaternion describing the rotation

inline **Transformation**(int64_t timestamp, const Eigen::Matrix<*Scalar*, 3, 1> &translation, const Eigen::Matrix<*Scalar*, 3, 3> &rotationMatrix)

Construct the transformation from timestamp, 3D translation vector and quaternion describing the rotation.

Parameters

- **timestamp** – Unix timestamp in microsecond format
- **translation** – 3D translation vector
- **rotationMatrix** – Rotation matrix describing the rotation

inline **Transformation** (int64_t timestamp, const cv::Mat &translation, const cv::Mat &rotation)

Construct the transformation from timestamp, 3D translation vector and quaternion describing the rotation.

Parameters

- **timestamp** – Unix timestamp in microsecond format
- **translation** – 3D translation vector
- **rotation** – 3x3 rotation matrix

inline int64_t **getTimestamp** () const

Get timestamp.

Returns

Unix timestamp of the transformation in microseconds.

inline const Eigen::Matrix<*Scalar*, 4, 4> &**getTransform** () const

Get the transformation matrix.

Returns

Transformation matrix in 4x4 format

inline Eigen::Matrix<*Scalar*, 3, 3> **getRotationMatrix** () const

Retrieve a copy of 3x3 rotation matrix.

Returns

3x3 rotation matrix

inline Eigen::Quaternion<*Scalar*> **getQuaternion** () const

Retrieve rotation expressed as a quaternion.

Returns

Quaternion containing rotation.

template<concepts::Coordinate3DConstructible **Output** = Eigen::Matrix<*Scalar*, 3, 1>>

inline *Output* **getTranslation** () const

Retrieve translation as 3D vector.

Returns

Vector containing translation.

template<concepts::Coordinate3DConstructible **Output** = Eigen::Matrix<*Scalar*, 3, 1>, concepts::Coordinate3D **Input**>

inline *Output* **transformPoint** (const *Input* &point) const

Transform a point using this transformation.

Parameters

point – Point to be transformed

Returns

Transformed point

template<concepts::Coordinate3DConstructible **Output** = Eigen::Matrix<*Scalar*, 3, 1>, concepts::Coordinate3D **Input**>

inline *Output* rotatePoint (const *Input* &point) const

Apply rotation only transformation on the given point.

Parameters

point – Point to be transformed

Returns

Transformed point

inline *Transformation*<*Scalar*> inverse () const

Calculate the inverse homogenous transformation of this transform.

Returns

Inverse transformation with the current timestamp.

inline *Transformation*<*Scalar*> delta (const *Transformation*<*Scalar*> &target) const

Find the transformation from current to target. ($T_{\text{target_current}}$ s.t. $p_{\text{target}} = T_{\text{target_current}} * p_{\text{current}}$).

Parameters

target – Target transformation.

Returns

Transformation from this to target.

inline bool operator== (const *Transformation*<*Scalar*> &rhs) const

Public Static Functions

static inline *Transformation* fromNonHomogenous (int64_t timestamp, const Eigen::Matrix<*Scalar*, 3, 4> &T)

Construct the transformation from a timestamp and 3x4 non-homogenous transformation matrix.

Parameters

- **timestamp** – Unix timestamp in microsecond format
- **T** – 3x4 3D transformation matrix

Private Members

int64_t mTimestamp

Timestamp of the transformation, Unix timestamp in microseconds.

Eigen::Matrix<*Scalar*, 4, 4> mT

The transformation itself, stored in 4x4 format:

$R|T$

0|1

class **TranslationLossFuncutor** : public *dv::optimization::OptimizationFuncutor*<float>

#include </builds/inivation/dv/dv-processing/include/dv-processing/optimization/contrast_maximization_translation_and_depth.hpp>

Given a chunk of events, the idea of contrast maximization is to warp events in space and time given a predefined motion model. Contrast maximization aims at finding the optimal parameters of the given motion model. The idea is that if the motion is perfectly estimated, all events corresponding to the same point in the scene, will be warped to the same image plane location, at a given point in time. If this happens, the reconstructed event image will be sharp, having high contrast. This high contrast is measured as variance in the image. For this reason, contrast maximization searches for the best motion parameters which maximize the contrast of the event image reconstructed after warping events in space to a specific point in time. In order to warp event in space

and time we use the “dv::kinematics::MotionCompensator” class. This contrast maximization class assumes pure camera translation motion model. Given a set of events in a time range (init_time, end_time), assuming a constant translational speed between init_time and end_time, translation (x, y, z) and scene depth are optimized to maximize contrast of event image. Since the speed is assumed to be constant between init_time and end_time, the camera position at time t_k is computed as : $t_k = \text{speed} * dt$, where $dt = t_k - \text{init_time}$. The scene depth is included in the optimization since it is strongly correlated to the camera translation. Scene depth is assumed to be constant between init_time and end_time.

Public Functions

inline **TranslationLossFunction** (dv::camera::CameraGeometry::SharedPtr &camera, const dv::EventStore &events, float contribution, int inputDim, int numMeasurements)

This contrast maximization class assumes pure camera translation motion model. Given a set of events in a time range (init_time, end_time), assuming a constant translational speed between init_time and end_time, translation (x, y, z) and scene depth are optimized to maximize contrast of event image.

Parameters

- **camera** – Camera geometry used to create motion compensator
- **events** – Events used to compute motion compensated image
- **contribution** – Contribution value of each event to the total pixel intensity
- **inputDim** – Number of parameters to optimize
- **numMeasurements** – Number of function evaluation performed to compute the gradient

inline virtual int **operator ()** (const Eigen::VectorXf &translationAndDepth, Eigen::VectorXf &stdInverse) const

Implementation of the objective function: optimize camera translation (x, y, z) and scene depth. Current cost is stored in stdInverse. Notice that since we want to maximize the contrast but optimizer minimize cost function we use as cost 1/contrast

Private Members

dv::camera::CameraGeometry::SharedPtr **mCamera**

Camera geometry data. This information is used to create motionCompensator and compensate events.

const dv::EventStore **mEvents**

Raw events compensated using translation along x, y, z and current scene depth.

const float **mContribution**

Event contribution for total pixel intensity. This parameter is very important since it strongly influence contrast value. It needs to be tuned based on scene and length of event chunk.

struct **Trigger** : public flatbuffers::NativeTable

Public Types

typedef TriggerFlatbuffer **TableType**

Public Functions

inline **Trigger** ()

inline **Trigger** (int64_t _timestamp, *TriggerType* _type)

Public Members

int64_t **timestamp**

TriggerType **type**

Public Static Functions

static inline constexpr const char ***GetFullyQualifiedName** ()

struct **TriggerBuilder**

Public Functions

inline void **add_timestamp** (int64_t timestamp)

inline void **add_type** (*TriggerType* type)

inline explicit **TriggerBuilder** (*flatbuffers::FlatBufferBuilder* &_fbb)

TriggerBuilder &**operator=** (const *TriggerBuilder*&)

inline *flatbuffers::Offset<TriggerFlatbuffer>* **Finish** ()

Public Members

flatbuffers::FlatBufferBuilder &**fbb_**

flatbuffers::uoffset_t **start_**

struct **TriggerFlatbuffer** : private *flatbuffers::Table*

Public Types

typedef *Trigger* **NativeTableType**

Public Functions

inline int64_t **timestamp** () const
Timestamp (μs).

inline *TriggerType* **type** () const
Type of trigger that occurred.

inline bool **Verify** (*flatbuffers::Verifier* &verifier) const

```
inline Trigger *UnPack (const flatbuffers::resolver_function_t *_resolver = nullptr) const
inline void UnPackTo (Trigger *_o, const flatbuffers::resolver_function_t *_resolver = nullptr) const
```

Public Static Functions

```
static inline const flatbuffers::TypeTable *MiniReflectTypeTable ()
static inline constexpr const char *GetFullyQualifiedName ()
static inline void UnPackToFrom (Trigger *_o, const TriggerFlatbuffer *_fb, const
                                flatbuffers::resolver_function_t *_resolver = nullptr)
static inline flatbuffers::Offset<TriggerFlatbuffer> Pack (flatbuffers::FlatBufferBuilder &_fbb, const Trigger *_o,
                                                         const flatbuffers::rehasher_function_t *_rehasher =
                                                         nullptr)
```

```
struct TriggerPacket : public flatbuffers::NativeTable
```

Public Types

```
typedef TriggerPacketFlatbuffer TableType
```

Public Functions

```
inline TriggerPacket ()
inline TriggerPacket (const std::vector<Trigger> &_elements)
```

Public Members

```
std::vector<Trigger> elements
```

Public Static Functions

```
static inline constexpr const char *GetFullyQualifiedName ()
```

Friends

```
inline friend std::ostream &operator<< (std::ostream &os, const TriggerPacket &packet)
```

```
struct TriggerPacketBuilder
```

Public Functions

```
inline void add_elements (flatbuffers::Offset<flatbuffers::Vector<flatbuffers::Offset<TriggerFlatbuffer>>>
                          elements)
inline explicit TriggerPacketBuilder (flatbuffers::FlatBufferBuilder &_fbb)
TriggerPacketBuilder &operator= (const TriggerPacketBuilder &)
inline flatbuffers::Offset<TriggerPacketFlatbuffer> Finish ()
```

Public Members

flatbuffers::FlatBufferBuilder &**fbb_**

flatbuffers::uoffset_t **start_**

struct **TriggerPacketFlatbuffer** : private *flatbuffers::Table*

Public Types

typedef *TriggerPacket* **NativeTableType**

Public Functions

inline const *flatbuffers::Vector*<*flatbuffers::Offset*<*TriggerFlatbuffer*>> ***elements** () const

inline bool **Verify** (*flatbuffers::Verifier* &verifier) const

inline *TriggerPacket* ***UnPack** (const *flatbuffers::resolver_function_t* *_resolver = nullptr) const

inline void **UnPackTo** (*TriggerPacket* *_o, const *flatbuffers::resolver_function_t* *_resolver = nullptr) const

Public Static Functions

static inline const *flatbuffers::TypeTable* ***MiniReflectTypeTable** ()

static inline constexpr const char ***GetFullyQualifiedName** ()

static inline void **UnPackToFrom** (*TriggerPacket* *_o, const *TriggerPacketFlatbuffer* *_fb, const
flatbuffers::resolver_function_t *_resolver = nullptr)

static inline *flatbuffers::Offset*<*TriggerPacketFlatbuffer*> **Pack** (*flatbuffers::FlatBufferBuilder* &**fbb**, const
TriggerPacket *_o, const
flatbuffers::rehasher_function_t *_rehasher =
nullptr)

Public Static Attributes

static constexpr const char ***identifier** = "TRIG"

struct **Type**

Public Functions

inline constexpr **Type** () noexcept

inline constexpr **Type** (const *std::string_view* identifier_, const size_t sizeOfType_, *PackFuncPtr* pack_,
UnpackFuncPtr unpack_, *ConstructPtr* construct_, *DestructPtr* destruct_,
TimeElementExtractorPtr timeElementExtractor_, *TimeRangeExtractorPtr*
timeRangeExtractor_)

`~Type () = default`

`Type (const Type &t) = default`

`Type &operator= (const Type &rhs) = default`

`Type (Type &&t) = default`

`Type &operator= (Type &&rhs) = default`

`inline constexpr bool operator== (const Type &rhs) const noexcept`

`inline constexpr bool operator!= (const Type &rhs) const noexcept`

Public Members

`uint32_t id`

`size_t sizeOfType`

`PackFuncPtr pack`

`UnpackFuncPtr unpack`

`ConstructPtr construct`

`DestructPtr destruct`

`TimeElementExtractorPtr timeElementExtractor`

`TimeRangeExtractorPtr timeRangeExtractor`

`struct TypedObject`

Public Functions

`inline constexpr TypedObject (const Type &type_)`

`inline ~TypedObject () noexcept`

`TypedObject (const TypedObject &t) = delete`

`TypedObject &operator= (const TypedObject &rhs) = delete`

`inline TypedObject (TypedObject &&t)`

`inline TypedObject &operator= (TypedObject &&rhs)`

`inline constexpr bool operator== (const TypedObject &rhs) const noexcept`

`inline constexpr bool operator!= (const TypedObject &rhs) const noexcept`

`template<class TargetType>`

```
inline std::shared_ptr<TargetType> moveToSharedPtr ()
```

Cast and move the pointer to the data into a shared pointer. The underlying data is not affected, but it invalidates this instance and passes the ownership of the data to the shared pointer - it will take care of memory management from the point of this method call.

Template Parameters

TargetType – Target type to cast the typed object into

Returns

Public Members

```
void *obj
```

```
Type type
```

```
struct TypeError
```

Public Types

```
using Info = std::string
```

Public Static Functions

```
static inline std::string format (const Info &info)
```

```
class UNIXSocket : public dv::io::network::SocketBase
```

#include </builds/inivation/dv/dv-processing/include/dv-processing/io/network/unix_socket.hpp> Minimal wrapper of UNIX socket. It follows RAII principle, the socket will be closed and released when this object is released.

Public Types

```
using socketType = asioUNIX::socket
```

Public Functions

```
inline explicit UNIXSocket (asioUNIX::socket &&s)
```

Initial a socket wrapper by taking ownership of a connected socket.

Parameters

s –

```
inline ~UNIXSocket () override
```

```
inline virtual bool isOpen () const override
```

Check whether socket is open and active.

Returns

True if socket is open, false otherwise.

```
inline virtual void close () override
```

Close underlying UNIX socket cleanly.

inline virtual void **write** (const asio::const_buffer &buf, CompletionHandler &&wrHandler) override

Write handler needs following signature: void (const boost::system::error_code &, size_t)

inline virtual void **read** (const asio::mutable_buffer &buf, CompletionHandler &&rdHandler) override

Read handler needs following signature: void (const boost::system::error_code &, size_t)

inline virtual void **syncWrite** (const asio::const_buffer &buf) override

Blocking write data to the socket.

Parameters

buf – Data to write.

inline virtual void **syncRead** (const asio::mutable_buffer &buf) override

Blocking read from socket.

Parameters

buf – Buffer for data to be read into.

Private Members

asioUNIX::socket **socket**

bool **socketClosed** = false

class **UpdateIntervalOrFeatureCountRedetection** : public *dv::features::RedetectionStrategy*

#include </builds/inivation/dv/dv-processing/include/dv-processing/features/redetection_strategy.hpp> Redetection strategy based on interval from last detection or minimum number of tracks. This class combines redetection logic from *UpdateIntervalRedetection* and *FeatureCountRedetection*.

Public Functions

inline explicit **UpdateIntervalOrFeatureCountRedetection** (const *dv::Duration* updateInterval, const float minimumProportionOfTracks)

Redetection strategy based on updating if specific amount of time from last detection has passed or minimum number of tracks to follow.

inline virtual bool **decideRedetection** (const *TrackerBase* &tracker) override

Check whether to perform redetection.

Private Members

UpdateIntervalRedetection **updateIntervalRedetection**

FeatureCountRedetection **featureCountRedetection**

class **UpdateIntervalRedetection** : public *dv::features::RedetectionStrategy*

#include </builds/inivation/dv/dv-processing/include/dv-processing/features/redetection_strategy.hpp> Redetection strategy based on interval from last detection.

Public Functions

inline explicit **UpdateIntervalRedetection** (const *dv::Duration* updateInterval)

Redetection strategy based on updating if specific amount of time from last detection has passed.

inline virtual bool **decideRedetection** (const *TrackerBase* &tracker) override

Check whether to perform redetection.

Protected Attributes

const int64_t **mUpdateTime**

int64_t **mLastDetectionTime** = -*std::numeric_limits<int64_t>::infinity*()

class **USBDevice**

Subclassed by *dv::io::camera::DAVIS*, *dv::io::camera::DVS128*, *dv::io::camera::DVXplorer*, *dv::io::camera::USBDeviceNextGen*

Public Types

enum class **LogLevel** : uint8_t

Values:

enumerator **LVL_NONE**

enumerator **LVL_ERROR**

enumerator **LVL_WARNING**

enumerator **LVL_INFO**

enumerator **LVL_DEBUG**

using **loggerCallbackType** = *std::function*<void(*LogLevel* level, *std::string_view* deviceName, *std::string_view* message)>

Public Functions

inline *DeviceDescriptor* **getDeviceDescriptor** () const

Get device descriptor. Describes this device uniquely.

Returns

this device's descriptor structure

inline uint8_t **getFirmwareVersion** () const

Get camera firmware version.

Returns

camera firmware version

inline *std::string* **getSerialNumber** () const

Get camera serial number.

Returns

camera serial number

inline *std::string* **getDeviceName** () const

Get camera name.

Returns

camera name

inline libusb_speed **getConnectionSpeed** () const

Get USB connection speed.

Returns

USB connection speed

inline void **setLogLevel** (const *LogLevel* level)

Set device's log-level.

Parameters

level – device log-level

inline *LogLevel* **getLogLevel** () const

Get device's log-level.

Returns

device log-level

inline void **setTransfersNumber** (const uint32_t transfersNumber)

Set number of USB buffers used for data transfer.

Parameters

transfersNumber – number of USB buffers used for data transfer.

inline uint32_t **getTransfersNumber** () const

Get number of USB buffers used for data transfer.

Returns

number of USB buffers used for data transfer.

inline void **setTransfersSize** (const uint32_t transfersSize)

Set size in bytes of USB buffers used for data transfer.

Parameters

transfersSize – size in bytes of USB buffers used for data transfer.

inline uint32_t **getTransfersSize** () const

Get size in bytes of USB buffers used for data transfer.

Returns

size in bytes of USB buffers used for data transfer.

Protected Types

using **compatibleCameraCallbackType** = *std::function*<*std::optional*<*CameraModel*>(uint16_t vid, uint16_t pid, *USBDeviceType* deviceType)>

```
using controlOutCallbackType = std::function<void(libusb_transfer_status status)>

using controlInCallbackType = std::function<void(libusb_transfer_status status, std::span<const uint8_t>
buffer)>

using usbTransferCallbackType = std::function<void(std::span<const uint8_t> data)>

using usbShutdownCallbackType = std::function<void()>
```

Protected Functions

```
inline USBDevice (const DeviceDescriptor &deviceToOpen, loggerCallbackType deviceLogger, const LogLevel
deviceLogLevel, const std::string_view shortName, const std::string_view longName, const
compatibleCameraCallbackType &expectedDevice, const int32_t requiredFirmwareVersion
= -1)
```

Construct and open a USB device. Match is based on VID, PID, USB Bus Address. If present, serial number is also verified to match.

Parameters

- **deviceToOpen** – description of which device to open exactly. VID, PID, USB Bus Address must be defined.
- **deviceLogger** – per-device logging callback. Any such logger *must* be thread-safe.
- **deviceLogLevel** – initial per-device log-level for info logging, can be changed later with *setLogLevel()*.
- **shortName** – brief name for related device threads. Should be at most 6 characters for maximum compatibility.
- **longName** – long name for full device name, used together with serial number in logging.
- **expectedDevice** – decide if given descriptor is valid to select a compatible camera model on opening.
- **requiredFirmwareVersion** – required firmware version for successful operation of this device (-1 disables the check, else the exact version given must correspond to the device one).

```
inline ~USBDevice ()
```

```
inline void cleanupBuffers ()
```

```
inline void usbControlTransferOutAsync (const uint8_t bRequest, const uint16_t wValue, const uint16_t
wIndex, const std::span<const uint8_t> data,
controlOutCallbackType controlOutCallback) const
```

```
inline void usbControlTransferInAsync (const uint8_t bRequest, const uint16_t wValue, const uint16_t
wIndex, const size_t dataSize, controlInCallbackType
controlInCallback) const
```

```
inline void usbControlTransferOut (const uint8_t bRequest, const uint16_t wValue, const uint16_t wIndex,
const std::span<const uint8_t> data) const
```

```
inline void usbControlTransferIn (const uint8_t bRequest, const uint16_t wValue, const uint16_t wIndex,
const std::span<const uint8_t> data) const
```

```

inline void spiConfigSendMultiple (const std::span<SPIConfigurationParameters> configs) const

inline void spiConfigSendMultipleAsync (const std::span<SPIConfigurationParameters> configs,
                                         controlOutCallbackType callback) const

inline void spiConfigSend (const uint8_t moduleAddr, const uint16_t paramAddr, const uint32_t param)
                           const

inline void spiConfigSendAsync (const uint8_t moduleAddr, const uint16_t paramAddr, const uint32_t
                                param, controlOutCallbackType callback) const

inline uint32_t spiConfigReceive (const uint8_t moduleAddr, const uint16_t paramAddr) const

inline void spiConfigReceiveAsync (const uint8_t moduleAddr, const uint16_t paramAddr,
                                    std::function<void(libusb_transfer_status status, uint32_t param)>
                                    callback) const

inline void log (const LogLevel level, const std::string_view message) const

inline void resetCameraName (const std::string_view cameraName)
    Update camera name base, append serial number after it. ONLY CALL IN CONSTRUCTOR!

```

Parameters

cameraName – updated camera name base

```

inline void setDataCallback (usbTransferCallbackType callback)

```

Set data handling callback, receives the buffers filled with data from the BULK endpoint. Not thread-safe, call before `usbThreadStart()` only.

Parameters

callback – data handling callback

```

inline void setShutdownCallback (usbShutdownCallbackType callback)

```

Set exceptional shutdown handling callback, called when the device goes away unexpectedly. Not thread-safe, call before `usbThreadStart()` only. Called from a context where locks are held, so it cannot call any of the USB data transfer control functions: `usbThreadStart()`, `usbThreadStop()`, `usbDataTransfersStart()`, `usbDataTransfersStop()`, `setTransfersNumber()`, `getTransfersNumber()`, `setTransfersSize()`, `getTransfersSize()`. You should only use this to set some asynchronous notification flag to be safe!

Parameters

callback – exceptional shutdown handling callback

```

inline void setDebugCallback (usbTransferCallbackType callback)

```

Set debug handling callback, receives the buffers filled with debug data from the INTERRUPT endpoint. Not thread-safe, call before `usbThreadStart()` only.

Parameters

callback – debug data handling callback

```

inline void setDataEndpoint (const uint8_t dataEndPoint)

```

Set data handling callback, receives the buffers filled with data from the BULK endpoint. Not thread-safe, call before `usbThreadStart()` only.

Parameters

dataEndPoint – new data endpoint address

```

inline uint8_t getDataEndpoint () const

```

```

inline bool isUSBThreadRunning () const

```

```
inline bool isUSBDataTransferRunning () const

inline void usbThreadStart ()

inline void usbThreadStop ()

inline void usbDataTransfersStart ()

inline void usbDataTransfersStop ()

inline void usbDebugTransfersStart ()

inline void usbDebugTransfersStop ()
```

Protected Static Functions

```
static inline std::vector<DeviceDescriptor> findCompatibleDevices (const compatibleCameraCallbackType
                                                                &compatibleDevice,
                                                                loggerCallbackType logger)

static inline std::vector<DeviceDescriptor> filterCompatibleDevices (const std::vector<DeviceDescriptor>
                                                                &list, const std::string_view
                                                                filterBySerialNumber)
```

Protected Static Attributes

```
static constexpr uint8_t VENDOR_REQUEST_SPI_CONFIG = {0xBF}

static constexpr uint16_t VID_INIVATION = {0x152A}
```

Private Types

```
using LibusbManagedContext = std::unique_ptr<libusb_context, decltype(libusbContextDeleter)>

using LibusbManagedDeviceHandle = std::unique_ptr<libusb_device_handle,
decltype(libusbDeviceHandleDeleter)>

using LibusbManagedDeviceList = std::unique_ptr<libusb_device*[], decltype(libusbDeviceListDeleter)>

using LibusbManagedTransfer = std::unique_ptr<libusb_transfer, decltype(libusbTransferDeleter)>
```

Private Functions

```
inline std::vector<LibusbManagedTransfer> usbAllocateTransfers (const size_t transfersNumber, const
                                                                size_t transfersSize, const uint8_t
                                                                endpoint, const libusb_transfer_type
                                                                endpointType, const
                                                                libusb_transfer_cb_fn callback)

inline void usbCancelTransfers (std::vector<LibusbManagedTransfer> &transfers)

inline void usbAllocateDataTransfersNLCK ()
```

```
inline void usbCancelAndDeallocateDataTransfersNLCK ()
```

```
inline void usbAllocateDebugTransfers ()
```

```
inline void usbCancelAndDeallocateDebugTransfers ()
```

Private Members

```
std::atomic<LogLevel> mUsbLogLevel = {LogLevel::LVL_WARNING}
```

```
mutable std::shared_mutex mUsbDeviceNameLock = {}
```

```
std::string mUsbDeviceName = {}
```

```
loggerCallbackType mUsbDeviceLogger = {}
```

```
LibusbManagedContext mDeviceContext = {nullptr, libusbContextDeleter}
```

```
LibusbManagedDeviceHandle mDeviceHandle = {nullptr, libusbDeviceHandleDeleter}
```

```
DeviceDescriptor mDescriptor = {}
```

```
std::string mUsbThreadName = {}
```

```
std::thread mUsbThread = {}
```

```
std::atomic<bool> mUsbThreadRun = {false}
```

```
mutable std::mutex mUsbOpsLock = {}
```

```
uint8_t mDataEndPoint = {DEFAULT_DATA_ENDPOINT}
```

```
std::atomic<bool> mDataTransfersRun = {false}
```

```
mutable std::mutex mDataTransfersLock = {}
```

```
std::vector<LibusbManagedTransfer> mDataTransfersNLCK = {}
```

```
uint32_t mDataTransfersActiveNLCK = {0}
```

```
uint32_t mDataTransfersFailedNLCK = {0}
```

```
uint32_t mDataTransfersNumberNLCK = {32}
```

```
uint32_t mDataTransfersSizeNLCK = {8 * 1024}

usbTransferCallbackType mUsbDataCallback = {}

usbShutdownCallbackType mUsbShutdownCallback = {}

std::vector<LibusbManagedTransfer> mDebugTransfers = {}

std::atomic<uint32_t> mDebugTransfersActive = {0}

usbTransferCallbackType mUsbDebugCallback = {}
```

Private Static Functions

```
static inline void LIBUSB_CALL usbControlOutCallback (struct libusb_transfer *transfer)

static inline void LIBUSB_CALL usbControlInCallback (struct libusb_transfer *transfer)

static inline void LIBUSB_CALL libusbLogCallback (libusb_context *ctx,
const enum libusb_log_level level, const char *str)

static inline std::string errorPrint (const std::string_view msg, const int code)

static inline std::string generateRepeatableSerialNumber (const DeviceDescriptor &desc)

static inline std::optional<std::string> fetchSerialNumber (libusb_device_handle *const handle, const uint8_t
serialNumberIndex)

static inline void usbThreadFunc (USBDevice *state)

static inline void LIBUSB_CALL usbDataTransferCallback (struct libusb_transfer *transfer)

static inline void LIBUSB_CALL usbDebugTransferCallback (struct libusb_transfer *transfer)
```

Private Static Attributes

```
static constexpr size_t MAX_CONTROL_TRANSFER_SIZE = {4 * 1024}

static constexpr uint8_t MAX_SERIAL_NUMBER_LENGTH = {8}

static constexpr uint8_t DEFAULT_DATA_ENDPOINT = {0x82}

static constexpr uint8_t DEBUG_ENDPOINT = {0x81}
```

```

static constexpr uint32_t DEBUG_TRANSFERS_NUMBER = {8}

static constexpr uint32_t DEBUG_TRANSFERS_SIZE = {64}

static constexpr uint8_t VENDOR_REQUEST_LOG_LEVEL = {0xB1}

static constexpr uint8_t VENDOR_REQUEST_DATA_CLEANUP = {0xC6}

static constexpr uint8_t VENDOR_REQUEST_SPI_CONFIG_MULTIPLE = {0xC2}

static std::shared_mutex LIBUSB_LOG_MAP_LOCK = {}

static std::unordered_map<libusb_context*, std::function<void(LogLevel level, std::string_view message)>>
LIBUSB_LOG_MAP = {}

static constexpr auto libusbContextDeleter = [] (libusb_context *ctx)
{libusb_exit(ctx); {std::unique_lock
lock{LIBUSB_LOG_MAP_LOCK}; LIBUSB_LOG_MAP.erase(ctx); }}

static constexpr auto libusbDeviceHandleDeleter = [] (libusb_device_handle *handle)
{libusb_close(handle); }

static constexpr auto libusbDeviceListDeleter = [] (libusb_device **list)
{libusb_free_device_list(list, true); }

static constexpr auto libusbTransferDeleter = [] (libusb_transfer *transfer)
{libusb_free_transfer(transfer); }

```

Friends

```
inline friend std::ostream &operator<< (std::ostream &os, const LogLevel &var)
```

class **USBDeviceNextGen** : public *dv::io::camera::USBDevice*

Subclassed by *dv::io::camera::DVXplorerM*

Protected Functions

```

inline USBDeviceNextGen (const DeviceDescriptor &deviceToOpen, loggerCallbackType deviceLogger, const
                        LogLevel deviceLogLevel, const std::string_view shortName, const std::string_view
                        longName, const compatibleCameraCallbackType &expectedDevice, const int32_t
                        requiredFirmwareVersion = -1)

void spiConfigSendMultiple (std::span<SPICongfigurationParameters>) const = delete

void spiConfigSendMultipleAsync (std::span<SPICongfigurationParameters>, controlOutCallbackType)
                                const = delete

```

```
inline void spiConfigSend (const uint16_t moduleAddr, const uint16_t paramAddr, const uint64_t param)
    const
```

```
inline void spiConfigSendAsync (const uint16_t moduleAddr, const uint16_t paramAddr, const uint64_t
    param, controlOutCallbackType callback) const
```

```
inline uint64_t spiConfigReceive (const uint16_t moduleAddr, const uint16_t paramAddr) const
```

```
inline void spiConfigReceiveAsync (const uint16_t moduleAddr, const uint16_t paramAddr,
    std::function<void(libusb_transfer_status status, uint64_t param)>
    callback) const
```

struct **VDACBias**

On-chip voltage digital-to-analog converter configuration. See '<https://docs.inivation.com/hardware/hardware-advanced-usage/biasing.html>'²⁶ for more details.

Public Functions

```
constexpr VDACBias () = default
```

```
inline constexpr VDACBias (const uint8_t voltage, const uint8_t current = 7)
```

Public Members

```
uint8_t voltageValue = {0}
```

Voltage, between 0 and 63, as a fraction of 1/64th of VDD=3.3V.

```
uint8_t currentValue = {0}
```

Current, between 0 and 7, that drives the voltage.

struct **WriteJob**

Public Functions

```
inline WriteJob (const asio::const_buffer &buffer, SocketBase::CompletionHandler handler)
```

Public Members

```
asio::const_buffer mBuffer
```

```
SocketBase::CompletionHandler mHandler
```

```
class WriteOnlyFile : private dv::io::SimpleWriteOnlyFile
```

²⁶ <https://docs.inivation.com/hardware/hardware-advanced-usage/biasing.html>

Public Functions

WriteOnlyFile () = delete

inline **WriteOnlyFile** (const *std::filesystem::path* &filePath, const *std::string_view* outputInfo,
std::unique_ptr<dv::io::compression::CompressionSupport> compression,
std::unique_ptr<dv::io::support::IOStatistics> stats = nullptr)

inline **WriteOnlyFile** (const *std::filesystem::path* &filePath, const *std::string_view* outputInfo, const
CompressionType compression = *CompressionType::NONE*,
std::unique_ptr<dv::io::support::IOStatistics> stats = nullptr)

inline **~WriteOnlyFile** ()

inline void **write** (const *dv::types::TypedObject* *const packet, const int32_t streamId)

inline void **write** (const void *ptr, const *dv::types::Type* &type, const int32_t streamId)

Private Functions

inline void **pushVersion** (const *std::shared_ptr<const dv::io::support::IODataBuffer>* version)

inline void **pushHeader** (const *std::shared_ptr<const dv::io::support::IODataBuffer>* header)

inline void **pushPacket** (const *std::shared_ptr<const dv::io::support::IODataBuffer>* packet)

inline void **pushFileDataTable** (const *std::shared_ptr<const dv::io::support::IODataBuffer>* fileDataTable)

inline void **writeThread** ()

inline void **stop** ()

inline void **emptyWriteBuffer** ()

inline void **writeVersion** (const *std::shared_ptr<const dv::io::support::IODataBuffer>* packet)

inline void **writeHeader** (const *std::shared_ptr<const dv::io::support::IODataBuffer>* packet)

inline void **writePacket** (const *std::shared_ptr<const dv::io::support::IODataBuffer>* packet)

inline void **writeFileDataTable** (const *std::shared_ptr<const dv::io::support::IODataBuffer>* packet)

Private Members

std::string **mOutputInfo**

dv::io::Writer **mWriter**

std::mutex **mMutex**

std::queue<std::function<void(void)>> **mWriteBuffer**

std::atomic<bool> **mStopRequested** = {false}

`std::thread mWriteThread`

class **WriteOrderedSocket**

`#include </builds/inivation/dv/dv-processing/include/dv-processing/io/network/write_ordered_socket.hpp>` Write ordered socket. Implemented because in asio simultaneous `async_writes` are not allowed.

 See also

<https://stackoverflow.com/questions/45813835/boostasio-ordering-of-data-sent-to-a-socket>

Public Functions

inline explicit **WriteOrderedSocket** (`std::unique_ptr<SocketBase>` &&socket)

inline void **write** (const asio::const_buffer &buf, `SocketBase::CompletionHandler` &&wrHandler)

Add a buffer to be written out to the socket. This call adds the buffer to a ordered queue that guarantees that will chain multiple `write_async` calls to the socket so no simultaneous calls would happen.

Parameters

- **buf** – Buffers to be written into the socket.
- **wrHandler** – Write handler that is called when buffer write is completed.

inline void **close** ()

Close the underlying socket.

inline bool **isOpen** () const

Check whether underlying socket is open

Returns

inline void **read** (const asio::mutable_buffer &buf, `SocketBase::CompletionHandler` &&rdHandler)

Read data from the socket. This only wraps the read call of the underlying socket.

Parameters

- **buf** –
- **rdHandler** –

Private Members

`std::deque<WriteJob> mWriteQueue`

No locking for writeQueue because all changes are posted to `io_service` thread.

`std::unique_ptr<dv::io::network::SocketBase> mSocket`

Underlying socket.

class **Writer**

Public Types

```
using WriteHandler = dv::std_function_exact<void(const std::shared_ptr<const
dv::io::support::IODataBuffer>)>>
```

Public Functions

```
Writer() = delete
```

```
inline explicit Writer(std::unique_ptr<dv::io::compression::CompressionSupport> compression,
                      std::unique_ptr<dv::io::support::IOStatistics> stats = nullptr,
                      std::unique_ptr<dv::FileDataTable> dataTable = nullptr)
```

```
inline explicit Writer(const dv::CompressionType compression, std::unique_ptr<dv::io::support::IOStatistics>
                      stats = nullptr, std::unique_ptr<dv::FileDataTable> dataTable = nullptr)
```

```
~Writer() = default
```

```
Writer(const Writer &other) = delete
```

```
Writer &operator=(const Writer &other) = delete
```

```
Writer(Writer &&other) noexcept = default
```

```
Writer &operator=(Writer &&other) noexcept = default
```

```
inline auto getCompressionType()
```

```
inline size_t writeAedatVersion(const WriteHandler &writeHandler)
```

```
inline size_t writeHeader(const int64_t dataTablePosition, const std::string_view infoNode, const
                        WriteHandler &writeHandler)
```

```
inline size_t writePacket(const dv::types::TypedObject *const packet, const int32_t streamId, const
                        WriteHandler &writeHandler)
```

```
inline size_t writePacket(const void *ptr, const dv::types::Type &type, const int32_t streamId, const
                        WriteHandler &writeHandler)
```

```
inline int64_t writeFileDataTable(const WriteHandler &writeHandler)
```

Public Static Functions

```
static inline std::shared_ptr<dv::io::support::IODataBuffer> encodeAedat4Version()
```

```
static inline std::shared_ptr<dv::io::support::IODataBuffer> encodeFileHeader(const int64_t
                                                                    dataTablePosition, const
                                                                    std::string_view infoNode,
                                                                    const dv::CompressionType
                                                                    compressionType)
```

```
static inline void encodePacketHeader(const std::shared_ptr<dv::io::support::IODataBuffer> packet, const
                                                                    int32_t streamId)
```

```
static inline std::shared_ptr<dv::io::support::IODataBuffer> encodePacketBody(const void *ptr, const
                                                                    dv::types::Type &type)
```

```
static inline std::shared_ptr<dv::io::support::IODataBuffer> encodeFileDataTable (const dv::FileDataTable
                                                                    &table)
```

Private Functions

```
inline void writeToDestination (const std::shared_ptr<const dv::io::support::IODataBuffer> data, const
                                WriteHandler &writeHandler)
```

```
inline void compressData (dv::io::support::IODataBuffer &packet)
```

```
inline void updateFileDataTable (const uint64_t byteOffset, const uint64_t numElements, const int64_t
                                timestampStart, const int64_t timestampEnd, const dv::PacketHeader
                                &header)
```

Private Members

```
std::unique_ptr<dv::io::support::IOStatistics> mStats
```

```
std::unique_ptr<dv::io::compression::CompressionSupport> mCompressionSupport
```

```
std::unique_ptr<dv::FileDataTable> mFileDataTable
```

```
uint64_t mByteOffset = {0}
```

```
class XMLConfigReader
```

Public Functions

```
XMLConfigReader () = delete
```

```
inline XMLConfigReader (const std::string_view xmlContent)
```

```
inline XMLConfigReader (const std::string_view xmlContent, const std::string_view expectedRootName)
```

```
inline const XMLTreeNode &getRoot () const
```

Private Functions

```
inline void parseXML (const std::string_view xmlContent, const std::string_view expectedRootName)
```

Private Members

```
XMLTreeNode mRoot
```

Private Static Functions

```
static inline std::vector<std::reference_wrapper<const boost::property_tree::ptree>> xmlFilterChildNodes (const
                                                                    boost::prop-
                                                                    erty_tree::ptree
                                                                    &con-
                                                                    tent,
                                                                    const
                                                                    std::string
                                                                    &name)
```

```
static inline void consumeXML (const boost::property_tree::ptree &content, XMLTreeNode &node)
```

```
static inline dv::io::support::VariantValueOwning stringToValueConverter (const std::string &typeStr, const
                                                                    std::string &valueStr)
```

```
class XMLConfigWriter
```

Public Functions

```
XMLConfigWriter () = delete
```

```
inline XMLConfigWriter (const XMLTreeNode &root)
```

```
inline const std::string &getXMLContent () const
```

Private Functions

```
inline void writeXML (const XMLTreeNode &root)
```

Private Members

```
std::string mXMLOutputContent
```

Private Static Functions

```
static inline boost::property_tree::ptree generateXML (const XMLTreeNode &node, const std::string
                                                                    &prevPath)
```

```
static inline std::pair<std::string, std::string> valueToStringConverter (const
                                                                    dv::io::support::VariantValueOwning
                                                                    &value)
```

```
struct XMLTreeAttribute : public dv::io::support::XMLTreeCommon
```

Public Functions

```
XMLTreeAttribute () = delete
```

```
inline explicit XMLTreeAttribute (const std::string_view name)
```

Public Members

dv::io::support::VariantValueOwning **mValue**

struct **XMLTreeCommon**

Subclassed by *dv::io::support::XMLTreeAttribute*, *dv::io::support::XMLTreeNode*

Public Functions

XMLTreeCommon () = delete

inline explicit **XMLTreeCommon** (const *std::string_view* name)

inline bool **operator==** (const *XMLTreeCommon* &rhs) const noexcept

inline auto **operator<=>** (const *XMLTreeCommon* &rhs) const noexcept

inline bool **operator==** (const *std::string_view* &rhs) const noexcept

inline auto **operator<=>** (const *std::string_view* &rhs) const noexcept

Public Members

std::string **mName**

struct **XMLTreeNode** : public *dv::io::support::XMLTreeCommon*

Public Functions

inline explicit **XMLTreeNode** ()

inline explicit **XMLTreeNode** (const *std::string_view* name)

Public Members

std::vector<XMLTreeNode> **mChildren**

std::vector<XMLTreeAttribute> **mAttributes**

class **ZstdCompressionSupport** : public *dv::io::compression::CompressionSupport*

Public Functions

inline explicit **ZstdCompressionSupport** (const *CompressionType* type)

inline explicit **ZstdCompressionSupport** (const int compressionLevel)

Create a Zstd compression support class with custom compression. Internally sets compression type to *CompressionType::ZSTD*.

 See also

For more info on compression level values see here: https://facebook.github.io/zstd/zstd_manual.html

Parameters

compressionLevel – Compression level, recommended range is [1, 22].

inline virtual void **compress** (*dv::io::support::IODataBuffer* &packet) override

Private Members

std::shared_ptr<ZSTD_CCtx_s> **mContext**

int **mLevel** = {3}

class **ZstdDecompressionSupport** : public *dv::io::compression::DecompressionSupport*

Public Functions

inline explicit **ZstdDecompressionSupport** (const *CompressionType* type)

inline virtual void **decompress** (*std::vector*<*std::byte*> &src, *std::vector*<*std::byte*> &target) override

Private Functions

inline void **initDecompressionContext** ()

Private Members

std::shared_ptr<ZSTD_DCtx_s> **mContext**

template<class **T**>
concept **MeanShiftKernel**

template<class **T1**, class **T2**>
concept **Accepts**

template<class **T**>
concept **AddressableEvent**

template<class **T**>
concept **BlockAccessible**

template<class **Type**>
concept **CompatibleWithSlicer**

#include </builds/inivation/dv/dv-processing/include/dv-processing/core/stream_slicer.hpp> Concept that verifies that given type is compatible for use with stream slicer.

tparam Type

Type to verify

template<class **T**>

```
concept Coordinate2D

template<class T>
concept Coordinate2DAccessors

template<class T>
concept Coordinate2DConstructible

template<class T>
concept Coordinate2DIterable

template<class T>
concept Coordinate2DMembers

template<class T>
concept Coordinate2DMutableIterable

template<class T>
concept Coordinate3D

template<class T>
concept Coordinate3DAccessors

template<class T>
concept Coordinate3DConstructible

template<class T>
concept Coordinate3DIterable

template<class T>
concept Coordinate3DMembers

template<class T>
concept Coordinate3DMutableIterable

template<class Packet>
concept DataPacket

template<class T, class Input>
concept DVFeatureDetectorAlgorithm

template<class T>
concept EigenType

template<class T>
concept Enum

template<class T, class EventStoreType>
concept EventFilter

template<class T, class EventStoreType>
concept EventOutputGenerator

template<class T>
concept EventStorage

template<class T, class EventStoreType>
```

```
concept EventToEventConverter
```

```
template<class T, class EventStoreType>
concept EventToFrameConverter
```

```
template<class T, class Input>
concept FeatureDetectorAlgorithm
```

```
template<class T>
concept FlatbufferPacket
```

```
template<class T>
concept FrameOutputGenerator
```

```
template<class T, class EventStoreType>
concept FrameToEventConverter
```

```
template<class T>
concept FrameToFrameConverter
```

```
template<class T>
concept HasElementsVector
```

```
template<class T>
concept HasTimestampedElementsVector
```

```
template<class T>
concept HasTimestampedElementsVectorByAccessor
```

```
template<class T>
concept HasTimestampedElementsVectorByMember
```

```
template<class T1, class T2>
concept InputStreamableFrom
```

```
template<class T1, class T2>
concept InputStreamableTo
```

```
template<typename FUNC, typename RETURN_TYPE, typename ...ARGUMENTS_TYPES>
concept InvocableReturnArgumentsStrong
```

#include </builds/inivation/dv/dv-processing/include/dv-processing/core/concepts.hpp> Checks if function is invocable with the given argument types exactly and its return value is the same as the given return type.

tparam **FUNC**
function-like object to check.

tparam **RETURN_TYPE**
required return type.

tparam **ARGUMENTS_TYPES**
required argument types.

```
template<typename FUNC, typename RETURN_TYPE, typename ...ARGUMENTS_TYPES>
concept InvocableReturnArgumentsWeak
```

#include </builds/inivation/dv/dv-processing/include/dv-processing/core/concepts.hpp> Checks if function is invocable with the given argument types and its return value is convertible to the given return type.

tparam **FUNC**
function-like object to check.

tparam RETURN_TYPE

required return type.

tparam ARGUMENTS_TYPES

required argument types.

template<class **T1**, class **T2**>
concept **IStreamableFrom**

template<class **T1**, class **T2**>
concept **IStreamableTo**

template<typename **T**>
concept **Iterable**

template<class **T**>
concept **KeyPoint**

template<class **T**>
concept **KeyPointConstructible**

template<class **T**>
concept **KeyPointIterable**

template<class **T**>
concept **KeyPointMutableIterable**

template<typename **T**>
concept **MutableIterable**

template<typename **T**>
concept **number**

template<class **T**>
concept **OpenCVFeatureDetectorAlgorithm**

template<class **T1**, class **T2**>
concept **OutputStreamableFrom**

template<class **T1**, class **T2**>
concept **OutputStreamableTo**

template<class **T**>
concept **SupportsConstantDepth**

template<class **T**>
concept **TimedImageContainer**

template<class **T**>
concept **Timestamped**

template<class **T**>
concept **TimestampedByAccessor**

template<class **T**>
concept **TimestampedByMember**

template<class **T**>

```

concept TimestampedIterable

template<class T>
concept TimestampMatrixContainer

template<class T, class EventStoreType>
concept TimeSurface

template<typename T>
concept HasCustomExceptionFormatter

template<typename T>
concept HasExtraExceptionInfo

template<typename T>
concept NoCustomExceptionFormatter

namespace dv

```

Typedefs

```

using EventStore = dv::AddressableEventStorage<dv::Event, dv::EventPacket>

using DepthEventStore = dv::AddressableEventStorage<dv::DepthEvent, dv::DepthEventPacket>

using EventStreamSlicer = dv::StreamSlicer<dv::EventStore>

using FrameStreamSlicer = dv::StreamSlicer<std::vector<dv::Frame>>

using IMUStreamSlicer = dv::StreamSlicer<std::vector<dv::IMU>>

using TriggerStreamSlicer = dv::StreamSlicer<std::vector<dv::Trigger>>

using TimeSurface = TimeSurfaceBase<EventStore>

using SpeedInvariantTimeSurface = dv::SpeedInvariantTimeSurfaceBase<dv::EventStore>

using StereoEventStreamSlicer = AddressableStereoEventStreamSlicer<dv::EventStore>

using TimestampClock = std::chrono::system_clock

using TimestampResolution = std::chrono::microseconds

using Duration = TimestampResolution
    Duration type that stores microsecond time period.

using TimePoint = std::chrono::time_point<TimestampClock, TimestampResolution>
    Timepoint type that stores microsecond time point related to system clock.

```

Enums

enum class **EventColor** : uint8_t

The EventColor enum contains the color of the Bayer color filter for a specific event address. WHITE means White/No Filter. Please take into account that there are usually twice as many green pixels as there are red or blue ones.

Values:

enumerator **WHITE**

enumerator **RED**

enumerator **GREEN1**

enumerator **BLUE**

enumerator **GREEN2**

enum class **PixelArrangement** : uint8_t

Color pixel block arrangement on the sensor. The sensor usually contain one red, one blue, and two green pixels. They can be arranged in different order, so exact color extraction, the pixel arrangement needs to be known.

Values:

enumerator **MONO**

No color filter present, all light passes.

enumerator **RGBG**

Standard Bayer color filter, 1 red 2 green 1 blue. Variation 1.

enumerator **GRGB**

Standard Bayer color filter, 1 red 2 green 1 blue. Variation 2.

enumerator **GBGR**

Standard Bayer color filter, 1 red 2 green 1 blue. Variation 3.

enumerator **BGRG**

Standard Bayer color filter, 1 red 2 green 1 blue. Variation 4.

enum class **TimeSlicingApproach**

Time handling approaches for number based slicing.

Values:

enumerator **BACKWARD**

Assign gap elements between previous numeric slice and current one.

enumerator **FORWARD**

Assign gap elements between current numeric slice and next one.

enum class **FrameFormat** : int8_t

Format values are compatible with OpenCV. Pixel layout follows OpenCV standard.

Values:

enumerator **OPENCV_8U_C1**

enumerator **OPENCV_8S_C1**

enumerator **OPENCV_16U_C1**

enumerator **OPENCV_16S_C1**

enumerator **OPENCV_32S_C1**

enumerator **OPENCV_32F_C1**

enumerator **OPENCV_64F_C1**

enumerator **OPENCV_16F_C1**

enumerator **OPENCV_8U_C2**

enumerator **OPENCV_8S_C2**

enumerator **OPENCV_16U_C2**

enumerator **OPENCV_16S_C2**

enumerator **OPENCV_32S_C2**

enumerator **OPENCV_32F_C2**

enumerator **OPENCV_64F_C2**

enumerator **OPENCV_16F_C2**

enumerator **OPENCV_8U_C3**

enumerator **OPENCV_8S_C3**

enumerator **OPENCV_16U_C3**

enumerator **OPENCV_16S_C3**

enumerator **OPENCV_32S_C3**

enumerator **OPENCV_32F_C3**

enumerator **OPENCV_64F_C3**

enumerator **OPENCV_16F_C3**

enumerator **OPENCV_8U_C4**

enumerator **OPENCV_8S_C4**

enumerator **OPENCV_16U_C4**

enumerator **OPENCV_16S_C4**

enumerator **OPENCV_32S_C4**

enumerator **OPENCV_32F_C4**

enumerator **OPENCV_64F_C4**

enumerator **OPENCV_16F_C4**

enumerator **MIN**

enumerator **MAX**

enum class **FrameSource** : int8_t

Image data source.

Values:

enumerator **UNDEFINED**

Undefined source, this value indicates that source field shouldn't be considered at all.

enumerator **SENSOR**

Image comes from a frame sensor.

enumerator **ACCUMULATION**

Image was accumulated.

enumerator **MOTION_COMPENSATION**

Image was accumulated using motion compensation.

enumerator **SYNTHETIC**

Image is synthetic, it does not represent any real data.

enumerator **RECONSTRUCTION**

Reconstructed image, it may come from a neural network that convert events to images.

enumerator **VISUALIZATION**

The image is designated for visualization only.

enumerator **OTHER**

Other sources, can be used to indicate a custom algorithm for image generation.

enumerator **MIN**

enumerator **MAX**

enum class **TriggerType** : int8_t

Values:

enumerator **TIMESTAMP_RESET**

A timestamp reset occurred.

enumerator **EXTERNAL_SIGNAL_RISING_EDGE**

A rising edge was detected (External Input module on device).

enumerator **EXTERNAL_SIGNAL_FALLING_EDGE**

A falling edge was detected (External Input module on device).

enumerator **EXTERNAL_SIGNAL_PULSE**

A pulse was detected (External Input module on device).

enumerator **EXTERNAL_GENERATOR_RISING_EDGE**

A rising edge was generated (External Generator module on device).

enumerator **EXTERNAL_GENERATOR_FALLING_EDGE**

A falling edge was generated (External Generator module on device).

enumerator **APS_FRAME_START**

An APS frame capture has started (*Frame* Event will follow).

enumerator **APS_FRAME_END**

An APS frame capture has completed (*Frame* Event is contemporary).

enumerator **APS_EXPOSURE_START**

An APS frame exposure has started (*Frame* Event will follow).

enumerator **APS_EXPOSURE_END**

An APS frame exposure has completed (*Frame* Event will follow).

enumerator **MIN**

enumerator **MAX**

enum class **Constants** : int32_t

Values:

enumerator **AEDAT_VERSION_LENGTH**

enumerator **MIN**

enumerator **MAX**

enum class **CompressionType** : int32_t

Values:

enumerator **NONE**

enumerator **LZ4**

enumerator **LZ4_HIGH**

enumerator **ZSTD**

enumerator **ZSTD_HIGH**

enumerator **MIN**

enumerator **MAX**

Functions

template<typename **EXPR**, typename **MSG**>

void **runtime_assert** (*EXPR* &&expression, *MSG* &&message, const *std::source_location* &location = *std::source_location::current()*)

```
inline uint32_t coordinateHash (const int16_t x, const int16_t y)
```

Function that creates perfect hash for 2d coordinates.

Parameters

- **x** – x coordinate
- **y** – y coordinate

Returns

a 64 bit hash that uniquely identifies the coordinates

```
template<class EventStoreType>
```

```
inline void roiFilter (const EventStoreType &in, EventStoreType &out, const cv::Rect &roi)
```

Extracts only the events that are within the defined region of interest. This function copies the events from the in EventStore into the given out EventStore, if they intersect with the given region of interest rectangle.

Parameters

- **in** – The EventStore to operate on. Won't be modified.
- **out** – The EventStore to put the ROI events into. Will get modified.
- **roi** – The rectangle with the region of interest.

```
template<class EventStoreType>
```

```
inline void polarityFilter (const EventStoreType &in, EventStoreType &out, bool polarity)
```

Filters events by polarity. Only events that exhibit the same polarity as given in polarity are kept.

Parameters

- **in** – Incoming EventStore to operate on. Won't get modified.
- **out** – The outgoing EventStore to store the kept events on
- **polarity** – The polarity of the events that should be kept

```
template<class EventStoreType>
```

```
inline void maskFilter (const EventStoreType &in, EventStoreType &out, const cv::Mat &mask)
```

Filter event with a coordinate mask. Discards any events that happen on coordinates where mask has a zero value and retains all events with coordinates where mask has a non-zero value.

Template Parameters

EventStoreType – Class for the event store container.

Parameters

- **in** – Incoming EventStore to operate on. Won't get modified.
- **out** – The outgoing EventStore to store the kept events on
- **mask** – The mask to be applied (requires CV_8UC1 type).

```
template<class EventStoreType>
```

```
inline void scale (const EventStoreType &in, EventStoreType &out, double xDivision, double yDivision)
```

Projects the event coordinates onto a smaller range. The x- and y-coordinates the divided by xFactor and yFactor respectively and floored to the next integer. This forms the new coordinates of the event. Due to the nature of this, it can happen that multiple events end up happening simultaneously at the same location. This is still a valid event stream, as time keeps monotonically increasing, but is something that is unlikely to be generated by an event camera.

Parameters

- **in** – The EventStore to operate on. Won't get modified

- **out** – The outgoing EventStore to store the projected events on
- **xDivision** – Division factor for the x-coordinate for the events
- **yDivision** – Division factor for the y-coordinate of the events

```
template<class EventStoreType>
```

```
inline cv::Rect boundingRect (const EventStoreType &packet)
```

Computes and returns a rectangle with dimensions such that all the events in the given EventStore fall into the bounding box.

Parameters

packet – The EventStore to work on

Returns

The smallest possible rectangle that contains all the events in packet.

```
inline std::ostream &operator<< (std::ostream &os, const EventColor &var)
```

```
inline std::ostream &operator<< (std::ostream &os, const PixelArrangement &var)
```

```
inline EventColor colorForEvent (const dv::Event &event, const PixelArrangement pixelArrangement =  
PixelArrangement::MONO)
```

Determine the color of the Bayer color filter for a specific event, based on its address. Please take into account that there are usually twice as many green pixels as there are red or blue ones.

Parameters

- **event** – event to determine filter color for.
- **pixelArrangement** – color pixel arrangement for a sensor.

Returns

filter color.

```
inline EventColor colorForPoint (const cv::Point pixelCoordinates, const PixelArrangement pixelArrangement  
= PixelArrangement::MONO)
```

Determine the color of the Bayer color filter for specific pixel coordinates. Please take into account that there are usually twice as many green pixels as there are red or blue ones.

Parameters

- **pixelCoordinates** – position to determine filter color for.
- **pixelArrangement** – color pixel arrangement for a sensor.

Returns

filter color.

```
inline std::ostream &operator<< (std::ostream &os, const TimeSlicingApproach &var)
```

```
inline TimePoint toTimePoint (const int64_t timestamp)
```

Convert a 64-bit integer microsecond timestamp into a chrono time-point.

Parameters

timestamp – 64-bit integer microsecond timestamp

Returns

Chrono time point (microseconds, system clock).

```
inline int64_t fromTimePoint (const TimePoint timepoint)
```

Convert a chrono time-point into a 64-bit integer microsecond timestamp.

Parameters

timestamp – Chrono time point (microseconds, system clock).

Returns

64-bit integer microsecond timestamp

```
inline int64_t now ()
```

Returns

Current system clock timestamp in microseconds as 64-bit integer.

```
template<dv::concepts::Enum Enumeration>
```

```
constexpr std::underlying_type_t<Enumeration> EnumAsInteger (const Enumeration value) noexcept
```

Functions to help handle enumerations and their values.

```
template<dv::concepts::Enum Enumeration, std::integral T>
```

```
constexpr Enumeration IntegerAsEnum (const T value) noexcept
```

```
template<typename T, typename U>
```

```
inline bool vectorContains (const std::vector<T> &vec, const U &item)
```

Functions to help deal with common vector operations: bool vectorContains(vec, item) bool vectorContainsIf(vec, predicate) bool vectorRemove(vec, item) bool vectorRemoveIf(vec, predicate) void vectorSortUnique(vec) void vectorSortUnique(vec, comparator)

```
template<typename T, typename Pred>
```

```
inline bool vectorContainsIf (const std::vector<T> &vec, Pred predicate)
```

```
template<typename T, typename U>
```

```
inline size_t vectorRemove (std::vector<T> &vec, const U &item)
```

```
template<typename T, typename Pred>
```

```
inline size_t vectorRemoveIf (std::vector<T> &vec, Pred predicate)
```

```
template<typename T>
```

```
inline void vectorSortUnique (std::vector<T> &vec)
```

```
template<typename T, typename Compare>
```

```
inline void vectorSortUnique (std::vector<T> &vec, Compare comp)
```

```
inline std::filesystem::path pathResolveNonExisting (const std::filesystem::path &path)
```

Path cleanup functions for existing paths (canonical) and possibly non-existing ones (absolute).

```
inline std::filesystem::path pathResolveExisting (const std::filesystem::path &path)
```

```
template<typename ObjectT, typename ...Args>
```

```
inline void *mallocConstructorSize (const size_t sizeOfObject, Args&&... args)
```

```
template<typename ObjectT, typename ...Args>
```

```
inline void *mallocConstructor (Args&&... args)
```

```
template<typename ObjectT>
```

```
inline void mallocDestructor (void *object) noexcept
```

```
inline std::string errnoToString (int errorNumber)
```

```
template<concepts::Coordinate2D Input>
```

```
inline bool isWithinDimensions (const Input &point, const cv::Size &resolution)
```

Check whether given point is non-negative and within dimensions of given resolution. The following check is performed: $X \in [0; (\text{width} - 1)]$ and $Y \in [0; (\text{height} - 1)]$. Function will check floating point coordinate fractional part overflow, it will return false in case even fractional part is beyond the valid range.

Parameters

- **point** – Coordinates to check.
- **resolution** – Pixel space resolution.

Returns

True if coordinates are within valid range, false otherwise.

```
inline bool operator== (const BoundingBox &lhs, const BoundingBox &rhs)
```

```
inline bool operator== (const BoundingBoxPacket &lhs, const BoundingBoxPacket &rhs)
```

```
inline const flatbuffers::TypeTable *BoundingBoxTypeTable ()
```

```
inline const flatbuffers::TypeTable *BoundingBoxPacketTypeTable ()
```

```
inline flatbuffers::Offset<BoundingBoxFlatbuffer> CreateBoundingBox (flatbuffers::FlatBufferBuilder &_fbb,  
                                                                    int64_t timestamp = 0, float topLeftX  
                                                                    = 0.0f, float topLeftY = 0.0f, float  
                                                                    bottomRightX = 0.0f, float  
                                                                    bottomRightY = 0.0f, float confidence  
                                                                    = 0.0f,  
                                                                    flatbuffers::Offset<flatbuffers::String>  
                                                                    label = 0)
```

```
inline flatbuffers::Offset<BoundingBoxFlatbuffer> CreateBoundingBoxDirect (flatbuffers::FlatBufferBuilder  
                                                                    &_fbb, int64_t timestamp =  
                                                                    0, float topLeftX = 0.0f, float  
                                                                    topLeftY = 0.0f, float  
                                                                    bottomRightX = 0.0f, float  
                                                                    bottomRightY = 0.0f, float  
                                                                    confidence = 0.0f, const char  
                                                                    *label = nullptr)
```

```
inline flatbuffers::Offset<BoundingBoxFlatbuffer> CreateBoundingBox (flatbuffers::FlatBufferBuilder &_fbb,  
                                                                    const BoundingBox *_o, const  
                                                                    flatbuffers::rehasher_function_t  
                                                                    *_rehasher = nullptr)
```

```
inline flatbuffers::Offset<BoundingBoxPacketFlatbuffer> CreateBoundingBoxPacket (flat-  
                                                                    buffers::FlatBufferBuilder  
                                                                    &_fbb, flat-  
                                                                    buffers::Offset<flatbuffers::Vector<flatbuff  
                                                                    elements = 0)
```

```
inline flatbuffers::Offset<BoundingBoxPacketFlatbuffer> CreateBoundingBoxPacketDirect (flat-  
                                                                    buffers::FlatBufferBuilder  
                                                                    &_fbb, const  
                                                                    std::vector<flatbuffers::Offset<Bou  
                                                                    *elements =  
                                                                    nullptr)
```

```

inline flatbuffers::Offset<BoundingBoxPacketFlatbuffer> CreateBoundingBoxPacket (flat-
                                                                    buffers::FlatBufferBuilder
                                                                    &_fbb, const
                                                                    BoundingBoxPacket
                                                                    *_o, const flat-
                                                                    buffers::rehasher_function_t
                                                                    *_rehasher = nullptr)

inline const dv::BoundingBoxPacketFlatbuffer *GetBoundingBoxPacket (const void *buf)

inline const dv::BoundingBoxPacketFlatbuffer *GetSizePrefixedBoundingBoxPacket (const void *buf)

inline const char *BoundingBoxPacketIdentifier ()

inline bool BoundingBoxPacketBufferHasIdentifier (const void *buf)

inline bool VerifyBoundingBoxPacketBuffer (flatbuffers::Verifier &verifier)

inline bool VerifySizePrefixedBoundingBoxPacketBuffer (flatbuffers::Verifier &verifier)

inline void FinishBoundingBoxPacketBuffer (flatbuffers::FlatBufferBuilder &fbb,
                                           flatbuffers::Offset<dv::BoundingBoxPacketFlatbuffer> root)

inline void FinishSizePrefixedBoundingBoxPacketBuffer (flatbuffers::FlatBufferBuilder &fbb, flat-
                                                                    buffers::Offset<dv::BoundingBoxPacketFlatbuffer>
                                                                    root)

inline std::unique_ptr<BoundingBoxPacket> UnPackBoundingBoxPacket (const void *buf, const
                                                                    flatbuffers::resolver_function_t *res
                                                                    = nullptr)

inline bool operator== (const DepthEvent &lhs, const DepthEvent &rhs)

inline bool operator== (const DepthEventPacket &lhs, const DepthEventPacket &rhs)

inline const flatbuffers::TypeTable *DepthEventTypeTable ()

inline const flatbuffers::TypeTable *DepthEventPacketTypeTable ()

FLATBUFFERS_MANUALLY_ALIGNED_STRUCT (8) DepthEvent final

FLATBUFFERS_STRUCT_END (DepthEvent, 16)

inline flatbuffers::Offset<DepthEventPacketFlatbuffer> CreateDepthEventPacket (flat-
                                                                    buffers::FlatBufferBuilder
                                                                    &_fbb, flat-
                                                                    buffers::Offset<flatbuffers::Vector<const
                                                                    DepthEvent*>> elements
                                                                    = 0)

inline flatbuffers::Offset<DepthEventPacketFlatbuffer> CreateDepthEventPacketDirect (flat-
                                                                    buffers::FlatBufferBuilder
                                                                    &_fbb, const
                                                                    std::vector<DepthEvent>
                                                                    *elements =
                                                                    nullptr)

```

```
inline flatbuffers::Offset<DepthEventPacketFlatbuffer> CreateDepthEventPacket (flat-  
buffers::FlatBufferBuilder  
&_fbb, const  
DepthEventPacket *_o,  
const flat-  
buffers::rehasher_function_t  
*_rehasher = nullptr)  
  
inline const dv::DepthEventPacketFlatbuffer *GetDepthEventPacket (const void *buf)  
  
inline const dv::DepthEventPacketFlatbuffer *GetSizePrefixedDepthEventPacket (const void *buf)  
  
inline const char *DepthEventPacketIdentifier ()  
  
inline bool DepthEventPacketBufferHasIdentifier (const void *buf)  
  
inline bool VerifyDepthEventPacketBuffer (flatbuffers::Verifier &verifier)  
  
inline bool VerifySizePrefixedDepthEventPacketBuffer (flatbuffers::Verifier &verifier)  
  
inline void FinishDepthEventPacketBuffer (flatbuffers::FlatBufferBuilder &fbb,  
flatbuffers::Offset<dv::DepthEventPacketFlatbuffer> root)  
  
inline void FinishSizePrefixedDepthEventPacketBuffer (flatbuffers::FlatBufferBuilder &fbb, flat-  
buffers::Offset<dv::DepthEventPacketFlatbuffer>  
root)  
  
inline std::unique_ptr<DepthEventPacket> UnPackDepthEventPacket (const void *buf, const  
flatbuffers::resolver_function_t *res =  
nullptr)  
  
inline bool operator== (const DepthFrame &lhs, const DepthFrame &rhs)  
  
inline const flatbuffers::TypeTable *DepthFrameTypeTable ()  
  
inline flatbuffers::Offset<DepthFrameFlatbuffer> CreateDepthFrame (flatbuffers::FlatBufferBuilder &_fbb,  
int64_t timestamp = 0, int16_t sizeX =  
0, int16_t sizeY = 0, uint16_t minDepth =  
0, uint16_t maxDepth = 65535,  
uint16_t step = 1, flat-  
buffers::Offset<flatbuffers::Vector<uint16_t>>  
depth = 0)  
  
inline flatbuffers::Offset<DepthFrameFlatbuffer> CreateDepthFrameDirect (flatbuffers::FlatBufferBuilder  
&_fbb, int64_t timestamp = 0,  
int16_t sizeX = 0, int16_t sizeY =  
0, uint16_t minDepth = 0,  
uint16_t maxDepth = 65535,  
uint16_t step = 1, const  
std::vector<uint16_t> *depth =  
nullptr)  
  
inline flatbuffers::Offset<DepthFrameFlatbuffer> CreateDepthFrame (flatbuffers::FlatBufferBuilder &_fbb,  
const DepthFrame *_o, const  
flatbuffers::rehasher_function_t  
*_rehasher = nullptr)
```

```

inline const dv::DepthFrameFlatbuffer *GetDepthFrame (const void *buf)

inline const dv::DepthFrameFlatbuffer *GetSizePrefixedDepthFrame (const void *buf)

inline const char *DepthFrameIdentifier ()

inline bool DepthFrameBufferHasIdentifier (const void *buf)

inline bool VerifyDepthFrameBuffer (flatbuffers::Verifier &verifier)

inline bool VerifySizePrefixedDepthFrameBuffer (flatbuffers::Verifier &verifier)

inline void FinishDepthFrameBuffer (flatbuffers::FlatBufferBuilder &fbb,
                                     flatbuffers::Offset<dv::DepthFrameFlatbuffer> root)

inline void FinishSizePrefixedDepthFrameBuffer (flatbuffers::FlatBufferBuilder &fbb,
                                                 flatbuffers::Offset<dv::DepthFrameFlatbuffer> root)

inline std::unique_ptr<DepthFrame> UnPackDepthFrame (const void *buf, const
                                                    flatbuffers::resolver_function_t *res = nullptr)

inline bool operator== (const Event &lhs, const Event &rhs)

inline bool operator== (const EventPacket &lhs, const EventPacket &rhs)

inline const flatbuffers::TypeTable *EventTypeTable ()

inline const flatbuffers::TypeTable *EventPacketTypeTable ()

FLATBUFFERS_STRUCT_END (Event, 16)

inline flatbuffers::Offset<EventPacketFlatbuffer> CreateEventPacket (flatbuffers::FlatBufferBuilder &fbb,
                                                                flat-
                                                                buffers::Offset<flatbuffers::Vector<const
                                                                Event*>> elements = 0)

inline flatbuffers::Offset<EventPacketFlatbuffer> CreateEventPacketDirect (flatbuffers::FlatBufferBuilder
                                                                &fbb, const
                                                                std::vector<Event> *elements
                                                                = nullptr)

inline flatbuffers::Offset<EventPacketFlatbuffer> CreateEventPacket (flatbuffers::FlatBufferBuilder &fbb,
                                                                const EventPacket *_o, const
                                                                flatbuffers::rehasher_function_t
                                                                *_rehasher = nullptr)

inline const dv::EventPacketFlatbuffer *GetEventPacket (const void *buf)

inline const dv::EventPacketFlatbuffer *GetSizePrefixedEventPacket (const void *buf)

inline const char *EventPacketIdentifier ()

inline bool EventPacketBufferHasIdentifier (const void *buf)

inline bool VerifyEventPacketBuffer (flatbuffers::Verifier &verifier)

inline bool VerifySizePrefixedEventPacketBuffer (flatbuffers::Verifier &verifier)

```

```
inline void FinishEventPacketBuffer (flatbuffers::FlatBufferBuilder &fbb,  
                                     flatbuffers::Offset<dv::EventPacketFlatbuffer> root)  
  
inline void FinishSizePrefixedEventPacketBuffer (flatbuffers::FlatBufferBuilder &fbb,  
                                                  flatbuffers::Offset<dv::EventPacketFlatbuffer> root)  
  
inline std::unique_ptr<EventPacket> UnPackEventPacket (const void *buf, const  
                                                     flatbuffers::resolver_function_t *res = nullptr)  
  
inline bool operator== (const Frame &lhs, const Frame &rhs)  
  
inline const flatbuffers::TypeTable *FrameTypeTable ()  
  
inline const FrameFormat (&EnumValuesFrameFormat ()) [32]  
  
inline const char *const *EnumNamesFrameFormat ()  
  
inline const char *EnumNameFrameFormat (FrameFormat e)  
  
inline const FrameSource (&EnumValuesFrameSource ()) [8]  
  
inline const char *const *EnumNamesFrameSource ()  
  
inline const char *EnumNameFrameSource (FrameSource e)  
  
inline flatbuffers::Offset<FrameFlatbuffer> CreateFrame (flatbuffers::FlatBufferBuilder &_fbb, int64_t  
                                                       timestamp = 0, int64_t timestampStartOfFrame = 0,  
                                                       int64_t timestampEndOfFrame = 0, int64_t  
                                                       timestampStartOfExposure = 0, int64_t  
                                                       timestampEndOfExposure = 0, FrameFormat format  
                                                       = FrameFormat::OPENCV_8U_C1, int16_t sizeX =  
                                                       0, int16_t sizeY = 0, int16_t positionX = 0, int16_t  
                                                       positionY = 0,  
                                                       flatbuffers::Offset<flatbuffers::Vector<uint8_t>>  
                                                       pixels = 0, int64_t exposure = 0, FrameSource source  
                                                       = FrameSource::UNDEFINED)  
  
inline flatbuffers::Offset<FrameFlatbuffer> CreateFrameDirect (flatbuffers::FlatBufferBuilder &_fbb, int64_t  
                                                       timestamp = 0, int64_t  
                                                       timestampStartOfFrame = 0, int64_t  
                                                       timestampEndOfFrame = 0, int64_t  
                                                       timestampStartOfExposure = 0, int64_t  
                                                       timestampEndOfExposure = 0,  
                                                       FrameFormat format =  
                                                       FrameFormat::OPENCV_8U_C1, int16_t  
                                                       sizeX = 0, int16_t sizeY = 0, int16_t  
                                                       positionX = 0, int16_t positionY = 0, const  
                                                       std::vector<uint8_t> *pixels = nullptr,  
                                                       int64_t exposure = 0, FrameSource source =  
                                                       FrameSource::UNDEFINED)  
  
inline flatbuffers::Offset<FrameFlatbuffer> CreateFrame (flatbuffers::FlatBufferBuilder &_fbb, const Frame  
                                                       *_o, const flatbuffers::rehasher_function_t *_rehasher  
                                                       = nullptr)  
  
inline const flatbuffers::TypeTable *FrameFormatTypeTable ()
```

```

inline const flatbuffers::TypeTable *FrameSourceTypeTable ()

inline const dv::FrameFlatbuffer *GetFrame (const void *buf)

inline const dv::FrameFlatbuffer *GetSizePrefixedFrame (const void *buf)

inline const char *FrameIdentifier ()

inline bool FrameBufferHasIdentifier (const void *buf)

inline bool VerifyFrameBuffer (flatbuffers::Verifier &verifier)

inline bool VerifySizePrefixedFrameBuffer (flatbuffers::Verifier &verifier)

inline void FinishFrameBuffer (flatbuffers::FlatBufferBuilder &fbb, flatbuffers::Offset<dv::FrameFlatbuffer>
                                root)

inline void FinishSizePrefixedFrameBuffer (flatbuffers::FlatBufferBuilder &fbb,
                                           flatbuffers::Offset<dv::FrameFlatbuffer> root)

inline std::unique_ptr<Frame> UnPackFrame (const void *buf, const flatbuffers::resolver_function_t *res =
                                           nullptr)

inline bool operator== (const Point3f &lhs, const Point3f &rhs)

inline bool operator== (const Point2f &lhs, const Point2f &rhs)

inline bool operator== (const Vec3f &lhs, const Vec3f &rhs)

inline bool operator== (const Vec2f &lhs, const Vec2f &rhs)

inline bool operator== (const Quaternion &lhs, const Quaternion &rhs)

inline const flatbuffers::TypeTable *Point3fTypeTable ()

inline const flatbuffers::TypeTable *Point2fTypeTable ()

inline const flatbuffers::TypeTable *Vec3fTypeTable ()

inline const flatbuffers::TypeTable *Vec2fTypeTable ()

inline const flatbuffers::TypeTable *QuaternionTypeTable ()

```

FLATBUFFERS_MANUALLY_ALIGNED_STRUCT (4) Point3f final

Structure representing absolute position of a 3D point.

Quaternion with Eigen compatible memory layout, should follow the Hamilton convention.

Structure representing a 2D vector.

Structure representing a 3D vector.

Structure representing absolute position of a 2D point.

FLATBUFFERS_STRUCT_END (Point3f, 12)

FLATBUFFERS_STRUCT_END (Point2f, 8)

```
FLATBUFFERS_STRUCT_END (Vec3f, 12)

FLATBUFFERS_STRUCT_END (Vec2f, 8)

FLATBUFFERS_STRUCT_END (Quaternion, 16)

inline bool operator==(const IMU &lhs, const IMU &rhs)

inline bool operator==(const IMUPacket &lhs, const IMUPacket &rhs)

inline const flatbuffers::TypeTable *IMUTypeTable()

inline const flatbuffers::TypeTable *IMUPacketTypeTable()

inline flatbuffers::Offset<IMUFlatbuffer> CreateIMU (flatbuffers::FlatBufferBuilder &fbb, int64_t timestamp =
    0, float temperature = 0.0f, float accelerometerX = 0.0f,
    float accelerometerY = 0.0f, float accelerometerZ = 0.0f,
    float gyroscopeX = 0.0f, float gyroscopeY = 0.0f, float
    gyroscopeZ = 0.0f, float magnetometerX = 0.0f, float
    magnetometerY = 0.0f, float magnetometerZ = 0.0f)

inline flatbuffers::Offset<IMUFlatbuffer> CreateIMU (flatbuffers::FlatBufferBuilder &fbb, const IMU *_o,
    const flatbuffers::rehasher_function_t *_rehasher =
    nullptr)

inline flatbuffers::Offset<IMUPacketFlatbuffer> CreateIMUPacket (flatbuffers::FlatBufferBuilder &fbb, flat-
buffers::Offset<flatbuffers::Vector<flatbuffers::Offset<IMUFlat
    elements = 0)

inline flatbuffers::Offset<IMUPacketFlatbuffer> CreateIMUPacketDirect (flatbuffers::FlatBufferBuilder
    &fbb, const
    std::vector<flatbuffers::Offset<IMUFlatbuffer>>
    *elements = nullptr)

inline flatbuffers::Offset<IMUPacketFlatbuffer> CreateIMUPacket (flatbuffers::FlatBufferBuilder &fbb, const
    IMUPacket *_o, const
    flatbuffers::rehasher_function_t *_rehasher
    = nullptr)

inline const dv::IMUPacketFlatbuffer *GetIMUPacket (const void *buf)

inline const dv::IMUPacketFlatbuffer *GetSizePrefixedIMUPacket (const void *buf)

inline const char *IMUPacketIdentifier()

inline bool IMUPacketBufferHasIdentifier (const void *buf)

inline bool VerifyIMUPacketBuffer (flatbuffers::Verifier &verifier)

inline bool VerifySizePrefixedIMUPacketBuffer (flatbuffers::Verifier &verifier)

inline void FinishIMUPacketBuffer (flatbuffers::FlatBufferBuilder &fbb,
    flatbuffers::Offset<dv::IMUPacketFlatbuffer> root)
```

```

inline void FinishSizePrefixedIMUPacketBuffer (flatbuffers::FlatBufferBuilder &fbb,
                                                flatbuffers::Offset<dv::IMUPacketFlatbuffer> root)

inline std::unique_ptr<IMUPacket> UnPackIMUPacket (const void *buf, const flatbuffers::resolver_function_t
                                                *res = nullptr)

inline bool operator== (const Observation &lhs, const Observation &rhs)

inline bool operator== (const Landmark &lhs, const Landmark &rhs)

inline bool operator== (const LandmarksPacket &lhs, const LandmarksPacket &rhs)

inline const flatbuffers::TypeTable *ObservationTypeTable ()

inline const flatbuffers::TypeTable *LandmarkTypeTable ()

inline const flatbuffers::TypeTable *LandmarksPacketTypeTable ()

inline flatbuffers::Offset<ObservationFlatbuffer> CreateObservation (flatbuffers::FlatBufferBuilder &_fbb,
                                                                    int32_t trackId = 0, int32_t cameraId =
                                                                    0, flatbuffers::Offset<flatbuffers::String>
                                                                    cameraName = 0, int64_t timestamp =
                                                                    0)

inline flatbuffers::Offset<ObservationFlatbuffer> CreateObservationDirect (flatbuffers::FlatBufferBuilder
                                                                    &_fbb, int32_t trackId = 0,
                                                                    int32_t cameraId = 0, const
                                                                    char *cameraName = nullptr,
                                                                    int64_t timestamp = 0)

inline flatbuffers::Offset<ObservationFlatbuffer> CreateObservation (flatbuffers::FlatBufferBuilder &_fbb,
                                                                    const Observation *_o, const
                                                                    flatbuffers::rehasher_function_t
                                                                    *_rehasher = nullptr)

inline flatbuffers::Offset<LandmarkFlatbuffer> CreateLandmark (flatbuffers::FlatBufferBuilder &_fbb, const
Point3f *pt = 0, int64_t id = 0, int64_t
timestamp = 0, flat-
buffers::Offset<flatbuffers::Vector<int8_t>>
descriptor = 0,
flatbuffers::Offset<flatbuffers::String>
descriptorType = 0,
flatbuffers::Offset<flatbuffers::Vector<float>>
covariance = 0, flat-
buffers::Offset<flatbuffers::Vector<flatbuffers::Offset<ObservationFlatbuffer>>
observations = 0)

inline flatbuffers::Offset<LandmarkFlatbuffer> CreateLandmarkDirect (flatbuffers::FlatBufferBuilder &_fbb,
                                                                    const Point3f *pt = 0, int64_t id = 0,
                                                                    int64_t timestamp = 0, const
                                                                    std::vector<int8_t> *descriptor =
                                                                    nullptr, const char *descriptorType =
                                                                    nullptr, const std::vector<float>
                                                                    *covariance = nullptr, const
                                                                    std::vector<flatbuffers::Offset<ObservationFlatbuffer>>
                                                                    *observations = nullptr)

```

```
inline flatbuffers::Offset<LandmarkFlatbuffer> CreateLandmark (flatbuffers::FlatBufferBuilder &_fbb, const
    Landmark *_o, const
    flatbuffers::rehasher_function_t *_rehasher =
    nullptr)

inline flatbuffers::Offset<LandmarksPacketFlatbuffer> CreateLandmarksPacket (flat-
    buffers::FlatBufferBuilder
    &_fbb, flat-
    buffers::Offset<flatbuffers::Vector<flatbuffers::C
    elements = 0, flat-
    buffers::Offset<flatbuffers::String>
    referenceFrame = 0)

inline flatbuffers::Offset<LandmarksPacketFlatbuffer> CreateLandmarksPacketDirect (flat-
    buffers::FlatBufferBuilder
    &_fbb, const
    std::vector<flatbuffers::Offset<Landma
    *elements = nullptr,
    const char
    *referenceFrame =
    nullptr)

inline flatbuffers::Offset<LandmarksPacketFlatbuffer> CreateLandmarksPacket (flat-
    buffers::FlatBufferBuilder
    &_fbb, const
    LandmarksPacket *_o,
    const flat-
    buffers::rehasher_function_t
    *_rehasher = nullptr)

inline const dv::LandmarksPacketFlatbuffer *GetLandmarksPacket (const void *buf)

inline const dv::LandmarksPacketFlatbuffer *GetSizePrefixedLandmarksPacket (const void *buf)

inline const char *LandmarksPacketIdentifier ()

inline bool LandmarksPacketBufferHasIdentifier (const void *buf)

inline bool VerifyLandmarksPacketBuffer (flatbuffers::Verifier &verifier)

inline bool VerifySizePrefixedLandmarksPacketBuffer (flatbuffers::Verifier &verifier)

inline void FinishLandmarksPacketBuffer (flatbuffers::FlatBufferBuilder &fbb,
    flatbuffers::Offset<dv::LandmarksPacketFlatbuffer> root)

inline void FinishSizePrefixedLandmarksPacketBuffer (flatbuffers::FlatBufferBuilder &fbb, flat-
    buffers::Offset<dv::LandmarksPacketFlatbuffer>
    root)

inline std::unique_ptr<LandmarksPacket> UnPackLandmarksPacket (const void *buf, const
    flatbuffers::resolver_function_t *res =
    nullptr)

inline bool operator== (const Pose &lhs, const Pose &rhs)

inline const flatbuffers::TypeTable *PoseTypeTable ()
```

```

inline flatbuffers::Offset<PoseFlatbuffer> CreatePose (flatbuffers::FlatBufferBuilder &_fbb, int64_t timestamp
= 0, const Vec3f *translation = 0, const Quaternion
*rotation = 0, flatbuffers::Offset<flatbuffers::String>
referenceFrame = 0,
flatbuffers::Offset<flatbuffers::String> targetFrame = 0)

inline flatbuffers::Offset<PoseFlatbuffer> CreatePoseDirect (flatbuffers::FlatBufferBuilder &_fbb, int64_t
timestamp = 0, const Vec3f *translation = 0,
const Quaternion *rotation = 0, const char
*referenceFrame = nullptr, const char
*targetFrame = nullptr)

inline flatbuffers::Offset<PoseFlatbuffer> CreatePose (flatbuffers::FlatBufferBuilder &_fbb, const Pose *_o,
const flatbuffers::rehasher_function_t *_rehasher =
nullptr)

inline const dv::PoseFlatbuffer *GetPose (const void *buf)

inline const dv::PoseFlatbuffer *GetSizePrefixedPose (const void *buf)

inline const char *PoseIdentifier ()

inline bool PoseBufferHasIdentifier (const void *buf)

inline bool VerifyPoseBuffer (flatbuffers::Verifier &verifier)

inline bool VerifySizePrefixedPoseBuffer (flatbuffers::Verifier &verifier)

inline void FinishPoseBuffer (flatbuffers::FlatBufferBuilder &fbb, flatbuffers::Offset<dv::PoseFlatbuffer>
root)

inline void FinishSizePrefixedPoseBuffer (flatbuffers::FlatBufferBuilder &fbb,
flatbuffers::Offset<dv::PoseFlatbuffer> root)

inline std::unique_ptr<Pose> UnPackPose (const void *buf, const flatbuffers::resolver_function_t *res = nullptr)

inline bool operator== (const TimedKeyPoint &lhs, const TimedKeyPoint &rhs)

inline bool operator== (const TimedKeyPointPacket &lhs, const TimedKeyPointPacket &rhs)

inline const flatbuffers::TypeTable *TimedKeyPointTypeTable ()

inline const flatbuffers::TypeTable *TimedKeyPointPacketTypeTable ()

inline flatbuffers::Offset<TimedKeyPointFlatbuffer> CreateTimedKeyPoint (flatbuffers::FlatBufferBuilder
&_fbb, const Point2f *pt = 0,
float size = 0.0f, float angle =
0.0f, float response = 0.0f, int32_t
octave = 0, int32_t class_id = 0,
int64_t timestamp = 0)

inline flatbuffers::Offset<TimedKeyPointFlatbuffer> CreateTimedKeyPoint (flatbuffers::FlatBufferBuilder
&_fbb, const TimedKeyPoint *_o,
const
flatbuffers::rehasher_function_t
*_rehasher = nullptr)

```

```
inline flatbuffers::Offset<TimedKeyPointPacketFlatbuffer> CreateTimedKeyPointPacket (flat-  
buffers::FlatBufferBuilder  
&_fbb, flat-  
buffers::Offset<flatbuffers::Vector<flat-  
elements = 0)
```

```
inline flatbuffers::Offset<TimedKeyPointPacketFlatbuffer> CreateTimedKeyPointPacketDirect (flat-  
buffers::FlatBufferBuilder  
&_fbb,  
const  
std::vector<flatbuffers::Offset<  
*elements  
= nullptr)
```

```
inline flatbuffers::Offset<TimedKeyPointPacketFlatbuffer> CreateTimedKeyPointPacket (flat-  
buffers::FlatBufferBuilder  
&_fbb, const  
TimedKeyPoint-  
Packet *_o, const  
flat-  
buffers::rehasher_function_t  
*_rehasher =  
nullptr)
```

```
inline const dv::TimedKeyPointPacketFlatbuffer *GetTimedKeyPointPacket (const void *buf)
```

```
inline const dv::TimedKeyPointPacketFlatbuffer *GetSizePrefixedTimedKeyPointPacket (const void  
*buf)
```

```
inline const char *TimedKeyPointPacketIdentifier ()
```

```
inline bool TimedKeyPointPacketBufferHasIdentifier (const void *buf)
```

```
inline bool VerifyTimedKeyPointPacketBuffer (flatbuffers::Verifier &verifier)
```

```
inline bool VerifySizePrefixedTimedKeyPointPacketBuffer (flatbuffers::Verifier &verifier)
```

```
inline void FinishTimedKeyPointPacketBuffer (flatbuffers::FlatBufferBuilder &fbb,  
flatbuffers::Offset<dv::TimedKeyPointPacketFlatbuffer>  
root)
```

```
inline void FinishSizePrefixedTimedKeyPointPacketBuffer (flatbuffers::FlatBufferBuilder &fbb, flat-  
buffers::Offset<dv::TimedKeyPointPacketFlatbuffer>  
root)
```

```
inline std::unique_ptr<TimedKeyPointPacket> UnPackTimedKeyPointPacket (const void *buf, const  
flatbuffers::resolver_function_t  
*res = nullptr)
```

```
inline bool operator== (const Trigger &lhs, const Trigger &rhs)
```

```
inline bool operator== (const TriggerPacket &lhs, const TriggerPacket &rhs)
```

```
inline const flatbuffers::TypeTable *TriggerTypeTable ()
```

```
inline const flatbuffers::TypeTable *TriggerPacketTypeTable ()
```

```

inline const TriggerType (&EnumValuesTriggerType ()) [10]

inline const char *const *EnumNamesTriggerType ()

inline const char *EnumNameTriggerType (TriggerType e)

inline flatbuffers::Offset<TriggerFlatbuffer> CreateTrigger (flatbuffers::FlatBufferBuilder &_fbb, int64_t
    timestamp = 0, TriggerType type =
    TriggerType::TIMESTAMP_RESET)

inline flatbuffers::Offset<TriggerFlatbuffer> CreateTrigger (flatbuffers::FlatBufferBuilder &_fbb, const
    Trigger *_o, const flatbuffers::rehasher_function_t
    *_rehasher = nullptr)

inline flatbuffers::Offset<TriggerPacketFlatbuffer> CreateTriggerPacket (flatbuffers::FlatBufferBuilder
    &_fbb, flat-
    buffers::Offset<flatbuffers::Vector<flatbuffers::Offset<
    elements = 0)

inline flatbuffers::Offset<TriggerPacketFlatbuffer> CreateTriggerPacketDirect (flat-
    buffers::FlatBufferBuilder
    &_fbb, const
    std::vector<flatbuffers::Offset<TriggerFlatbuff
    *elements = nullptr)

inline flatbuffers::Offset<TriggerPacketFlatbuffer> CreateTriggerPacket (flatbuffers::FlatBufferBuilder
    &_fbb, const TriggerPacket *_o,
    const
    flatbuffers::rehasher_function_t
    *_rehasher = nullptr)

inline const flatbuffers::TypeTable *TriggerTypeTypeTable ()

inline const dv::TriggerPacketFlatbuffer *GetTriggerPacket (const void *buf)

inline const dv::TriggerPacketFlatbuffer *GetSizePrefixedTriggerPacket (const void *buf)

inline const char *TriggerPacketIdentifier ()

inline bool TriggerPacketBufferHasIdentifier (const void *buf)

inline bool VerifyTriggerPacketBuffer (flatbuffers::Verifier &verifier)

inline bool VerifySizePrefixedTriggerPacketBuffer (flatbuffers::Verifier &verifier)

inline void FinishTriggerPacketBuffer (flatbuffers::FlatBufferBuilder &fbb,
    flatbuffers::Offset<dv::TriggerPacketFlatbuffer> root)

inline void FinishSizePrefixedTriggerPacketBuffer (flatbuffers::FlatBufferBuilder &fbb,
    flatbuffers::Offset<dv::TriggerPacketFlatbuffer>
    root)

inline std::unique_ptr<TriggerPacket> UnPackTriggerPacket (const void *buf, const
    flatbuffers::resolver_function_t *res = nullptr)

inline bool operator== (const PacketHeader &lhs, const PacketHeader &rhs)

```

```
inline bool operator==(const FileDataDefinition &lhs, const FileDataDefinition &rhs)

inline bool operator==(const FileDataTable &lhs, const FileDataTable &rhs)

inline const flatbuffers::TypeTable *PacketHeaderTypeTable ()

inline const flatbuffers::TypeTable *FileDataDefinitionTypeTable ()

inline const flatbuffers::TypeTable *FileDataTableTypeTable ()

FLATBUFFERS_STRUCT_END (PacketHeader, 8)

inline flatbuffers::Offset<FileDataDefinitionFlatbuffer> CreateFileDataDefinition (flat-
                                                                    buffers::FlatBufferBuilder
                                                                    &_fbb, int64_t
                                                                    ByteOffset = 0, const
                                                                    PacketHeader
                                                                    *PacketInfo = 0,
                                                                    int64_t NumElements
                                                                    = 0, int64_t
                                                                    TimestampStart = 0,
                                                                    int64_t TimestampEnd
                                                                    = 0)

inline flatbuffers::Offset<FileDataDefinitionFlatbuffer> CreateFileDataDefinition (flat-
                                                                    buffers::FlatBufferBuilder
                                                                    &_fbb, const
                                                                    FileDataDefinition *_o,
                                                                    const flat-
                                                                    buffers::rehasher_function_t
                                                                    *_rehasher = nullptr)

inline flatbuffers::Offset<FileDataTableFlatbuffer> CreateFileDataTable (flatbuffers::FlatBufferBuilder
                                                                    &_fbb, flat-
                                                                    buffers::Offset<flatbuffers::Vector<flatbuffers::Offset<
                                                                    Table = 0)

inline flatbuffers::Offset<FileDataTableFlatbuffer> CreateFileDataTableDirect (flat-
                                                                    buffers::FlatBufferBuilder
                                                                    &_fbb, const
                                                                    std::vector<flatbuffers::Offset<FileDataDefin
                                                                    *Table = nullptr)

inline flatbuffers::Offset<FileDataTableFlatbuffer> CreateFileDataTable (flatbuffers::FlatBufferBuilder
                                                                    &_fbb, const FileDataTable *_o,
                                                                    const
                                                                    flatbuffers::rehasher_function_t
                                                                    *_rehasher = nullptr)

inline const dv::FileDataTableFlatbuffer *GetFileDataTable (const void *buf)

inline const dv::FileDataTableFlatbuffer *GetSizePrefixedFileDataTable (const void *buf)

inline const char *FileDataTableIdentifier ()

inline bool FileDataTableBufferHasIdentifier (const void *buf)
```

```

inline bool VerifyFileDataTableBuffer (flatbuffers::Verifier &verifier)

inline bool VerifySizePrefixedFileDataTableBuffer (flatbuffers::Verifier &verifier)

inline void FinishFileDataTableBuffer (flatbuffers::FlatBufferBuilder &fbb,
                                       flatbuffers::Offset<dv::FileDataTableFlatbuffer> root)

inline void FinishSizePrefixedFileDataTableBuffer (flatbuffers::FlatBufferBuilder &fbb,
                                                  flatbuffers::Offset<dv::FileDataTableFlatbuffer>
                                                  root)

inline std::unique_ptr<FileDataTable> UnPackFileDataTable (const void *buf, const
                                                         flatbuffers::resolver_function_t *res = nullptr)

inline bool operator== (const IOHeader &lhs, const IOHeader &rhs)

inline const flatbuffers::TypeTable *IOHeaderTypeTable ()

inline const Constants (&EnumValuesConstants ()) [1]

inline const char *const *EnumNamesConstants ()

inline const char *EnumNameConstants (Constants e)

inline const CompressionType (&EnumValuesCompressionType ()) [5]

inline const char *const *EnumNamesCompressionType ()

inline const char *EnumNameCompressionType (CompressionType e)

inline flatbuffers::Offset<IOHeaderFlatbuffer> CreateIOHeader (flatbuffers::FlatBufferBuilder &_fbb,
                                                           CompressionType compression =
                                                           CompressionType::NONE, int64_t
                                                           dataTablePosition = -1,
                                                           flatbuffers::Offset<flatbuffers::String>
                                                           infoNode = 0)

inline flatbuffers::Offset<IOHeaderFlatbuffer> CreateIOHeaderDirect (flatbuffers::FlatBufferBuilder &_fbb,
                                                                    CompressionType compression =
                                                                    CompressionType::NONE, int64_t
                                                                    dataTablePosition = -1, const char
                                                                    *infoNode = nullptr)

inline flatbuffers::Offset<IOHeaderFlatbuffer> CreateIOHeader (flatbuffers::FlatBufferBuilder &_fbb, const
                                                             IOHeader *_o, const
                                                             flatbuffers::rehasher_function_t *_rehasher =
                                                             nullptr)

inline const flatbuffers::TypeTable *ConstantsTypeTable ()

inline const flatbuffers::TypeTable *CompressionTypeTypeTable ()

inline const dv::IOHeaderFlatbuffer *GetIOHeader (const void *buf)

inline const dv::IOHeaderFlatbuffer *GetSizePrefixedIOHeader (const void *buf)

inline const char *IOHeaderIdentifier ()

```

```
inline bool IOHeaderBufferHasIdentifier (const void *buf)

inline bool VerifyIOHeaderBuffer (flatbuffers::Verifier &verifier)

inline bool VerifySizePrefixedIOHeaderBuffer (flatbuffers::Verifier &verifier)

inline void FinishIOHeaderBuffer (flatbuffers::FlatBufferBuilder &fbb,
                                flatbuffers::Offset<dv::IOHeaderFlatbuffer> root)

inline void FinishSizePrefixedIOHeaderBuffer (flatbuffers::FlatBufferBuilder &fbb,
                                              flatbuffers::Offset<dv::IOHeaderFlatbuffer> root)

inline std::unique_ptr<IOHeader> UnPackIOHeader (const void *buf, const flatbuffers::resolver_function_t *res
                                              = nullptr)
```

Variables

```
static constexpr bool DEBUG_ENABLED = {true}

static constexpr std::array<std::array<EventColor, 4>, 5> PIXEL_COLOR_KEYS{{{EventColor::WHITE,
EventColor::WHITE, EventColor::WHITE, EventColor::WHITE}, {EventColor::RED, EventColor::GREEN1,
EventColor::GREEN2, EventColor::BLUE}, {EventColor::GREEN1, EventColor::RED, EventColor::BLUE,
EventColor::GREEN2}, {EventColor::GREEN2, EventColor::BLUE, EventColor::RED, EventColor::GREEN1},
{EventColor::BLUE, EventColor::GREEN2, EventColor::GREEN1, EventColor::RED}},}}

    Address to Color mapping for events based on Bayer filter.

static constexpr int VERSION_MAJOR = {2}

static constexpr int VERSION_MINOR = {0}

static constexpr int VERSION_PATCH = {1}

static constexpr int VERSION = {(2 * 10000) + (0 * 100) + 1}

static constexpr std::string_view NAME_STRING = {"dv-processing"}

static constexpr std::string_view VERSION_STRING = {"2.0.1"}
```

namespace dv

namespace camera

Enums

```
enum class DistortionModel

    Values:

    enumerator NONE
```

enumerator **RADIAL_TANGENTIAL**

enumerator **EQUIDISTANT**

Functions

inline *std::ostream* &**operator<<** (*std::ostream* &os, const *DistortionModel* &var)

inline *DistortionModel* **stringToDistortionModel** (const *std::string_view* model)

Convert a string into the Enum of the DistortionModel

Parameters

model –

Returns

the enum corresponding to the string

inline *std::string* **distortionModelToString** (const *DistortionModel* &model)

Convert a DistortionModel Enum into a string

Parameters

model –

Returns

the string that represent the Distortion model

namespace **calibrations**

namespace **internal**

Variables

static constexpr *std::string_view* **NONE_MODEL_STRING** = {"none"}

static constexpr *std::string_view* **RADIAL_TANGENTIAL_MODEL_STRING** = {"radialTangential"}

static constexpr *std::string_view* **EQUIDISTANT_MODEL_STRING** = {"equidistant"}

namespace **cluster**

namespace **mean_shift**

Typedefs

template<typename **TYPE**, int32_t **ROWS** = Eigen::Dynamic, int32_t **COLUMNS** = Eigen::Dynamic>

using **MeanShiftRowMajorMatrixXX** = *MeanShiftEigenMatrixAdaptor*<*TYPE*, *ROWS*, *COLUMNS*, Eigen::RowMajor>

Convenience alias for n-dimensional data in row-major sample order of arbitrary number of samples

template<typename **TYPE**, int32_t **ROWS** = Eigen::Dynamic, int32_t **COLUMNS** = Eigen::Dynamic>

```
using MeanShiftColMajorMatrixXX = MeanShiftEigenMatrixAdaptor<TYPE, ROWS, COLUMNS,  
Eigen::ColMajor>
```

Convenience alias for n-dimensional data in column-major sample order of arbitrary dimensions and number of samples

```
template<typename TYPE, int32_t SAMPLES = Eigen::Dynamic>
```

```
using MeanShiftRowMajorMatrixX1 = MeanShiftRowMajorMatrixXX<TYPE, SAMPLES, 1>
```

Convenience alias for 1-dimensional data in row-major sample order of arbitrary number of samples

```
template<typename TYPE, int32_t SAMPLES = Eigen::Dynamic>
```

```
using MeanShiftRowMajorMatrixX2 = MeanShiftRowMajorMatrixXX<TYPE, SAMPLES, 2>
```

Convenience alias for 2-dimensional data in row-major sample order of arbitrary number of samples

```
template<typename TYPE, int32_t SAMPLES = Eigen::Dynamic>
```

```
using MeanShiftRowMajorMatrixX3 = MeanShiftRowMajorMatrixXX<TYPE, SAMPLES, 3>
```

Convenience alias for 3-dimensional data in row-major sample order of arbitrary number of samples

```
template<typename TYPE, int32_t SAMPLES = Eigen::Dynamic>
```

```
using MeanShiftRowMajorMatrixX4 = MeanShiftRowMajorMatrixXX<TYPE, SAMPLES, 4>
```

Convenience alias for 4-dimensional data in row-major sample order of arbitrary number of samples

```
template<typename TYPE, int32_t SAMPLES = Eigen::Dynamic>
```

```
using MeanShiftColMajorMatrix1X = MeanShiftColMajorMatrixXX<TYPE, 1, SAMPLES>
```

Convenience alias for 1-dimensional data in column-major sample order of arbitrary number of samples

```
template<typename TYPE, int32_t SAMPLES = Eigen::Dynamic>
```

```
using MeanShiftColMajorMatrix2X = MeanShiftColMajorMatrixXX<TYPE, 2, SAMPLES>
```

Convenience alias for 2-dimensional data in column-major sample order of arbitrary number of samples

```
template<typename TYPE, int32_t SAMPLES = Eigen::Dynamic>
```

```
using MeanShiftColMajorMatrix3X = MeanShiftColMajorMatrixXX<TYPE, 3, SAMPLES>
```

Convenience alias for 3-dimensional data in column-major sample order of arbitrary number of samples

```
template<typename TYPE, int32_t SAMPLES = Eigen::Dynamic>
```

```
using MeanShiftColMajorMatrix4X = MeanShiftColMajorMatrixXX<TYPE, 4, SAMPLES>
```

Convenience alias for 4-dimensional data in column-major sample order of arbitrary number of samples

```
namespace kernel
```

```
namespace concepts
```

Typedefs

```
template<class T>
```

```
using iterable_element_type = typename std::remove_reference_t<decltype(*(std::declval<T>().begin()))>
```

Variables

```
template<typename T>

constexpr bool is_eigen_type = internal::is_eigen_impl<T>::value

template<typename Needle, typename ...Haystack>

constexpr bool is_type_one_of = std::disjunction_v<std::is_same<Needle, Haystack>...>
```

namespace **internal**

namespace **containers**

namespace **kd_tree**

Typedefs

```
template<typename TYPE, int32_t ROWS = Eigen::Dynamic, int32_t COLUMNS = Eigen::Dynamic>

using KDTreeRowMajorXX = KDTreeMatrixAdaptor<TYPE, ROWS, COLUMNS, Eigen::RowMajor>
    Convenience alias for n-dimensional data in row-major sample order of arbitrary number of samples

template<typename TYPE, int32_t ROWS = Eigen::Dynamic, int32_t COLUMNS = Eigen::Dynamic>

using KDTreeColMajorXX = KDTreeMatrixAdaptor<TYPE, ROWS, COLUMNS, Eigen::ColMajor>
    Convenience alias for n-dimensional data in column-major sample order of arbitrary dimensions and number of samples

template<typename TYPE, int32_t SAMPLES = Eigen::Dynamic>

using KDTreeRowMajorX1 = KDTreeRowMajorXX<TYPE, SAMPLES, 1>
    Convenience alias for 1-dimensional data in row-major sample order of arbitrary number of samples

template<typename TYPE, int32_t SAMPLES = Eigen::Dynamic>

using KDTreeRowMajorX2 = KDTreeRowMajorXX<TYPE, SAMPLES, 2>
    Convenience alias for 2-dimensional data in row-major sample order of arbitrary number of samples

template<typename TYPE, int32_t SAMPLES = Eigen::Dynamic>

using KDTreeRowMajorX3 = KDTreeRowMajorXX<TYPE, SAMPLES, 3>
    Convenience alias for 3-dimensional data in row-major sample order of arbitrary number of samples

template<typename TYPE, int32_t SAMPLES = Eigen::Dynamic>

using KDTreeRowMajorX4 = KDTreeRowMajorXX<TYPE, SAMPLES, 4>
    Convenience alias for 4-dimensional data in row-major sample order of arbitrary number of samples

template<typename TYPE, int32_t SAMPLES = Eigen::Dynamic>

using KDTreeColMajor1X = KDTreeColMajorXX<TYPE, 1, SAMPLES>
    Convenience alias for 1-dimensional data in column-major sample order of arbitrary number of samples

template<typename TYPE, int32_t SAMPLES = Eigen::Dynamic>
```

```
using KDTreeColMajor2X = KDTreeColMajorXX<TYPE, 2, SAMPLES>
```

Convenience alias for 2-dimensional data in column-major sample order of arbitrary number of samples

```
template<typename TYPE, int32_t SAMPLES = Eigen::Dynamic>
```

```
using KDTreeColMajor3X = KDTreeColMajorXX<TYPE, 3, SAMPLES>
```

Convenience alias for 3-dimensional data in column-major sample order of arbitrary number of samples

```
template<typename TYPE, int32_t SAMPLES = Eigen::Dynamic>
```

```
using KDTreeColMajor4X = KDTreeColMajorXX<TYPE, 4, SAMPLES>
```

Convenience alias for 5-dimensional data in column-major sample order of arbitrary number of samples

namespace **data**

Functions

```
template<dv::concepts::KeyPointIterable KPI>
```

```
inline std::vector<cv::KeyPoint> fromTimedKeyPoints (const KPI &points)
```

Convert *TimedKeyPoint* vector into cv::KeyPoint vector.

Parameters

points – KeyPoints to be converted.

Returns

A vector of cv::KeyPoint.

```
template<dv::concepts::KeyPointIterable KPI>
```

```
inline std::vector<cv::Point2f> convertToCvPoints (const KPI &points)
```

Convert *TimedKeyPoint* vector into cv::Point2f vector.

Parameters

points – KeyPoints to be converted.

Returns

A vector of cv::Point2f.

```
inline std::vector<dv::TimedKeyPoint> fromCvKeypoints (const std::vector<cv::KeyPoint> &points, const  
int64_t defaultTime = 0)
```

Create a vector of cv::KeyPoint from a given vector of *dv::TimedKeyPoint*.

Parameters

- **points** – cv::KeyPoint vector to be converted.
- **defaultTime** – Timestamp in microseconds to be assigned to all new TimedKeyPoints.

Returns

A vector of TimedKeyPoints.

```
inline cv::Mat depthFrameMap (dv::DepthFrame &frame)
```

Map a depth frame into an OpenCV Mat, no data copies are performed. The resulting cv::Mat will point to the same underlying data.

This function does not affect any data underlying, the const qualifier is not set since cv::Mat can't be const.

Parameters

frame – *Frame* to be mapped.

Returns

Mapped depth frame in `cv::Mat` with data type of `CV_16UC1`.

```
inline cv::Mat depthFrameInMeters (dv::DepthFrame &frame)
```

Converts the given depth frame into an OpenCV matrix containing depth values in meters.

Resulting `cv::Mat` will be of floating type and will apply conversion from millimeters to meters. Depth value of 0.0f should be considered invalid.

This function will copy and scale all values into meters.

Parameters

frame – Depth frame to be converted.

Returns

A `cv::Mat` containing scaled depth values in meters.

```
inline dv::DepthFrame depthFrameFromCvMat (const cv::Mat &depthImage)
```

Converts the given OpenCV matrix with depth values to *DepthFrame*.

`cv::Mat` can contain single-channel floating point containing depth values in meters or single-channel 16-bit unsigned integer values in millimeters. Zero should be used for invalid values.

This function will copy and scale all values into millimeter 16-bit integer representation.

Parameters

depthImage – `cv::Mat` containing the depth values.

Returns

Depth frame containing depth values in 16-bit unsigned integer values representing distance in millimeters.

```
template<std::floating_point Scalar = float>
```

```
inline dv::kinematics::Transformation<Scalar> transformFromPose (const dv::Pose &pose)
```

Convert a pose message into a transformation.

Parameters

pose – Input pose to be converted.

Returns

Transformation representing the pose.

```
template<std::floating_point Scalar = float>
```

```
inline dv::Pose poseFromTransformation (const dv::kinematics::Transformation<Scalar> &transform)
```

Convert a transformation into a pose message.

Parameters

transform – Input transform.

Returns

Pose message representing the transform.

namespace **generate**

Functions

```
inline cv::Mat sampleImage (const cv::Size &resolution)
```

Generate a sample image (single channel 8-bit unsigned integer) containing a few gray rectangles in a black background.

Parameters

resolution – Resolution of the output image.

Returns

Generated image.

```
inline dv::EventStore eventLine (const int64_t timestamp, const cv::Point &a, const cv::Point &b, size_t steps = 0)
```

Generate events along a line between two given end-points.

Parameters

- **timestamp** – Fixed timestamp assigned for all events.
- **a** – Starting point.
- **b** – Ending point.
- **steps** – Number of events generated for the line. If zero is provided, the function uses euclidean distance between the points.

Returns

A batch of event along the line.

```
inline dv::EventStore eventRectangle (const int64_t timestamp, const cv::Point &tl, const cv::Point &br)
```

Generate events along a rectangle edges between two given top-left and bottom right points.

Parameters

- **timestamp** – Fixed timestamp assigned for all events.
- **tl** – Top left coordinate of the rectangle.
- **br** – Bottom right coordinate of the rectangle.

Returns

Event batch containing events at the edges of a given rectangle.

```
inline dv::EventStore eventTestSet (const int64_t timestamp, const cv::Size &resolution)
```

Generate an event test set that contains event for a few intersecting rectangle edges.

Parameters

- **timestamp** – Fixed timestamp assigned for all events.
- **resolution** – Expected resolution limits for the events.

Returns

Generated event batch.

```
inline dv::EventStore uniformlyDistributedEvents (const int64_t timestamp, const cv::Size &resolution, const size_t count, const uint64_t seed = 0)
```

Generate a batch of uniformly distributed set of event within the given resolution.

Parameters

- **timestamp** – Fixed timestamp assigned for all events.
- **resolution** – Resolution limits.
- **count** – Number of events.
- **seed** – Seed for the RNG.

Returns

Generated event batch.

```
inline dv::EventStore normallyDistributedEvents (const int64_t timestamp, const dv::Point2f &center,
                                              const dv::Point2f &stddev, const size_t count, const
                                              uint64_t seed = 0)
```

Generate events normally distributed around a given center coordinates with given standard deviation.

Parameters

- **timestamp** – Timestamp to be assigned to the generated events
- **center** – Center coordinates
- **stddev** – Standard deviation for each of the axes
- **count** – Number of events to generate
- **seed** – Seed for the RNG

Returns

Set of normally distributed events

```
inline dv::EventStore uniformEventsWithinTimeRange (const int64_t startTime, const dv::Duration duration,
                                                    const cv::Size &resolution, const int64_t count,
                                                    const uint64_t seed = 0)
```

Generate a batch of uniformly distributed (in pixel-space) randomly generated events. The timestamps are generated by monotonically increasing the timestamp within the time duration.

Parameters

- **startTime** – Start timestamp in microseconds.
- **duration** – Duration of the generated data.
- **resolution** – Pixel space resolution.
- **count** – Number of output events.
- **seed** – Seed for the RNG.

Returns

Generated event batch.

```
inline cv::Mat dvLogo (const cv::Size &size, const bool colored = true, const cv::Scalar &bgColor =
                      dv::visualization::colors::white, const cv::Scalar &pColor =
                      dv::visualization::colors::iniBlue, const cv::Scalar &nColor =
                      dv::visualization::colors::darkGray)
```

Generate a DV logo using simple drawing methods. Generates in color or grayscale.

Parameters

- **size** – Output dimensions of the drawing
- **colored** – Colored output (CV_8UC3) if true, or grayscale (CV_8UC1) otherwise.

Returns

Image containing DV logo.

```
inline dv::EventStore imageToEvents (const int64_t timestamp, const cv::Mat &image, const uint8_t positive,
                                     const uint8_t negative)
```

Convert an image into event by matching pixel intensities. The algorithm will match all pixel values available in the and match against positive and negative pixel intensity values, according events are going to be added into the output event store. Other pixel intensity values are ignored.

Parameters

- **image** – Input image for conversion
- **positive** – Pixel brightness intensity value to consider the pixel to generate a positive polarity event.
- **negative** – Pixel brightness intensity value to consider the pixel to generate a negative polarity event.

Returns

Generated events.

inline *dv::EventStore* **dvLogoAsEvents** (const int64_t timestamp, const cv::Size &resolution)

Generate a DV logo using simple drawing methods. Generates negative polarity events on the pixels where logo has dark pixels and positive polarity events where pixels have brighter events.

Parameters

- **timestamp** – Timestamp assigned to each generated event.
- **resolution** – Resolution of the events.

Returns

Events that can be accumulated / visualized to generate a logo of DV.

inline *dv::IMU levelImuMeasurement* (const int64_t timestamp)

Generate an *IMU* measurement that measures a camera being on a stable and level surface. All measurement values are going to be zero, except for Y axis of accelerometer, it will measure -1.0G.

Parameters

timestamp – Timestamp to be assigned to the measurement.

Returns

Generated *IMU* measurement.

inline *dv::IMU addNoiseToImu* (const *dv::IMU* &measurement, const float accelerometerStddev, const float gyroscopeStddev, const uint64_t seed = 0)

Apply noise to imu measurements (accelerometer and gyroscope). The noise is modelled as a normal distribution with 0 mean and given standard deviation. The modelled noise is added to the given measurement and return a new *dv::IMU* structure with added noise.

Parameters

- **measurement** – *IMU* measurement to add noise to.
- **accelerometerStddev** – Accelerometer noise standard deviation.
- **gyroscopeStddev** – Gyroscope noise standard deviation.
- **seed** – Seed for the RNG.

Returns

Generated measurement with added noise.

inline *dv::IMU levelImuWithNoise* (const int64_t timestamp, const float accelerometerStddev = 0.1f, const float gyroscopeStddev = 0.01f, const uint64_t seed = 0)

Generate an *IMU* measurement that measures a camera being on a stable and level surface with additional measurement noise. The noise is modelled as a normal distribution with 0 mean and given standard deviation.

Parameters

- **timestamp** – Timestamp to be assigned to the measurement.
- **accelerometerStddev** – Accelerometer noise standard deviation.

- **gyroscopeStddev** – Gyroscope noise standard deviation.
- **seed** – Seed for the RNG.

Returns

Generated *IMU* measurement.

namespace **depth**

Functions

inline *std::shared_ptr*<cv::StereoMatcher> **defaultStereoMatcher** ()

Create a reasonable default stereo matcher, tailored for low texture images (that are generated by accumulating events) and for faster execution.

The method creates an instance of cv::StereoSGBM with following parameter values:

- minDisparity = 0
- numDisparities = 48
- blockSize = 11 : highest recommended block size, small block sizes generate noise in low texture)
- P1 = 8 * (blockSize ^ 2)
- P2 = 32 * (blockSize ^ 2) : P1 and P2 are calculated using recommended equations
- disp12MaxDiff = 0 : disparity is also calculated on right-left image pair, filter out any disparities that do not agree. This enables strong noise filtering (there can be a lot of noise due to low texture)
- preFilterCap = cv::StereoBM::PREFILTER_NORMALIZED_RESPONSE : disable Sobel filter pre-processing
- uniquenessRatio = 15 : this is also an aggressive value for a noise filter
- speckleWindowSize = 240 : this is also an aggressive value for a speckle noise filter
- speckleRange = 1 : this is also an aggressive value for a speckle noise filter
- mode = cv::StereoSGBM::MODE_SGBM_3WAY : Fastest disparity calculation mode

Returns

Stereo semi global block matching algorithm with reasonable defaults for low texture images.

namespace **exceptions**

Typedefs

using **DirectoryError** = *internal::Exception_*<*info::DirectoryError*>

using **DirectoryNotFound** = *internal::Exception_*<*info::DirectoryNotFound*, *DirectoryError*>

using **FileError** = *internal::Exception_*<*info::FileError*>

using **FileOpenError** = *internal::Exception_*<*info::FileOpenError*, *FileError*>

```
using FileReadError = internal::Exception_<info::FileReadError, FileError>

using FileWriteError = internal::Exception_<info::FileWriteError, FileError>

using FileNotFound = internal::Exception_<info::FileNotFound, FileError>

using AedatFileError = internal::Exception_<info::AedatFileError, FileError>

using AedatVersionError = internal::Exception_<info::AedatVersionError, AedatFileError>

using AedatFileParseError = internal::Exception_<info::AedatFileParseError, AedatFileError>

using EndOfFile = internal::Exception_<info::EndOfFile>

using RuntimeError = internal::Exception_<info::RuntimeError>

using BadAlloc = internal::Exception_<info::BadAlloc>

using OutOfRange = internal::Exception_<info::OutOfRange>

using LengthError = internal::Exception_<info::LengthError>

template<class TYPE>
using InvalidArgument = internal::Exception_<info::InvalidArgument<TYPE>>

using NullPointer = internal::Exception_<info::NullPointer>

using IOError = internal::Exception_<info::IOError>

using InputError = internal::Exception_<info::InputError, IOError>

using OutputError = internal::Exception_<info::OutputError, IOError>

using TypeError = internal::Exception_<info::TypeError>

namespace info

namespace internal
```

Functions

```
template<HasCustomExceptionFormatter T>
std::string format (const typename T::Info &info)
```

namespace **features**

Typedefs

```
using ImagePyrFeatureDetector = FeatureDetector<dv::ImagePyramid, cv::Feature2D>
```

```
using ImageFeatureDetector = FeatureDetector<dv::Frame, cv::Feature2D>
```

```
using EventFeatureBlobDetector = FeatureDetector<dv::EventStore, EventBlobDetector>
```

namespace **internal**

This class implement the Arc* corner detector presented in the following paper: <https://www.research-collection.ethz.ch/bitstream/handle/20.500.11850/277131/RAL2018-camera-ready.pdf>

Template Parameters

- **radius1** – radius of the first circle on which the timestamps are checked for corner-ness
- **radius2** – radius of the second circle on which the timestamps are checked for corner-ness

namespace **imgproc**

Functions

```
template<typename T>
inline auto cvMatStepToEigenStride (const cv::MatStep &step)
```

Conversion from cv::MatStep to Eigen::Stride

cv::MatStep stores steps in units of bytes, as the underlying matrix is always stored in uint8_t arrays, which are then interpreted at run-time based on the type (e.g. CV_8U). Contrary to this, Eigen stores matrices in arrays of a type that is determined at compile-time based on a template argument, and therefore stores its strides in units of pointer increments. The conversion between the two can be computed by dividing by or multiplying with sizeof(T).

Template Parameters

T – the type of the scalars stored in the matrices

Parameters

step – the step (stride) in the matrix in units of bytes

Returns

the corresponding Eigen::Stride for the cv::MatStep value provided

```
template<typename T>
inline auto cvMatToEigenMap (const cv::Mat &mat)
```

Maps an Eigen::Map onto a cv::Mat object. This provides a view to the internal storage of the cv::Mat, it doesn't copy any data.

Template Parameters

T – the type of the scalars stored in the matrices

Parameters

mat – the cv::Mat onto which an Eigen::Map should be mapped

Returns

the view into the cv::Mat via an Eigen::Map object

```
template<typename T>
```

```
inline auto cvMatToEigenMap (cv::Mat &mat)
```

Maps an Eigen::Map onto a cv::Mat object. This provides a view to the internal storage of the cv::Mat, it doesn't copy any data.

Template Parameters

T – the type of the scalars stored in the matrices

Parameters

mat – the cv::Mat onto which an Eigen::Map should be mapped

Returns

the view into the cv::Mat via an Eigen::Map object

```
template<typename T>
```

```
inline auto L1Distance (const Eigen::Block<T, Eigen::Dynamic, Eigen::Dynamic> &patch1, const  
Eigen::Block<T, Eigen::Dynamic, Eigen::Dynamic> &patch2)
```

Computes the L1 distance between two blocks (patches) of eigen matrices.

Template Parameters

T – The type of the underlying matrix

Parameters

- **patch1** – the first patch
- **patch2** – the second patch

Returns

the L1 distance between the two patches

```
template<typename T, int32_t MAP_OPTIONS, typename STRIDE>
```

```
inline auto L1Distance (const Eigen::Map<T, MAP_OPTIONS, STRIDE> &m1, const Eigen::Map<T,  
MAP_OPTIONS, STRIDE> &m2)
```

Computes the L1 distance between two matrices

 See also

Eigen::Map::MapOptions

Template Parameters

- **T** – The type of the underlying matrix
- **MAP_OPTIONS** – The options for the underlying matrix.
- **STRIDE** – The stride of the underlying matrix

Parameters

- **m1** – the first matrix
- **m2** – the second matrix

Returns

the L1 distance between the two matrices

```
template<typename T>
inline auto L1Distance (const Eigen::Matrix<T, Eigen::Dynamic, Eigen::Dynamic> &m1, const
                        Eigen::Matrix<T, Eigen::Dynamic, Eigen::Dynamic> &m2)
```

Computes the L1 distance between two matrices

Template Parameters

T – The type of the underlying matrix

Parameters

- **m1** – the first matrix
- **m2** – the second matrix

Returns

the L1 distance between the two matrices

```
inline auto L1Distance (const cv::Mat &m1, const cv::Mat &m2)
```

Computes the L1 distance between two matrices

Parameters

- **m1** – the first matrix
- **m2** – the second matrix

Returns

the L1 distance between the two matrices

```
template<typename T>
inline auto pearsonCorrelation (const Eigen::Block<T, Eigen::Dynamic, Eigen::Dynamic> &patch1, const
                                Eigen::Block<T, Eigen::Dynamic, Eigen::Dynamic> &patch2)
```

Computes the Pearson Correlation between two blocks (patches) of eigen matrices.

Template Parameters

T – The type of the underlying matrix

Parameters

- **patch1** – the first patch
- **patch2** – the second patch

Returns

the Pearson Correlation between the two patches

```
template<typename T, int32_t MAP_OPTIONS, typename STRIDE>
inline auto pearsonCorrelation (const Eigen::Map<T, MAP_OPTIONS, STRIDE> &m1, const
                                Eigen::Map<T, MAP_OPTIONS, STRIDE> &m2)
```

Computes the Pearson Correlation between two matrices

➡ See also

Eigen::Map::MapOptions

Template Parameters

- **T** – The type of the underlying matrix
- **MAP_OPTIONS** – The options for the underlying matrix.
- **STRIDE** – The stride of the underlying matrix

Parameters

- **m1** – the first matrix
- **m2** – the second matrix

Returns

the Pearson Correlation between the two matrices

```
template<typename T>
```

```
inline auto pearsonCorrelation (const Eigen::Matrix<T, Eigen::Dynamic, Eigen::Dynamic> &m1, const  
                                Eigen::Matrix<T, Eigen::Dynamic, Eigen::Dynamic> &m2)
```

Computes the Pearson Correlation between two matrices

Template Parameters

T – The type of the underlying matrix

Parameters

- **m1** – the first matrix
- **m2** – the second matrix

Returns

the Pearson Correlation between the two matrices

```
inline auto pearsonCorrelation (const cv::Mat &m1, const cv::Mat &m2)
```

Computes the Pearson Correlation between two matrices

Parameters

- **m1** – the first matrix
- **m2** – the second matrix

Returns

the Pearson Correlation between the two matrices

```
template<typename T>
```

```
inline auto cosineDistance (const Eigen::Block<T, Eigen::Dynamic, Eigen::Dynamic> &patch1, const  
                             Eigen::Block<T, Eigen::Dynamic, Eigen::Dynamic> &patch2)
```

Computes the Cosine Distance between two blocks (patches) of eigen matrices.

Template Parameters

T – The type of the underlying matrix

Parameters

- **patch1** – the first patch
- **patch2** – the second patch

Returns

the Cosine Distance between the two patches

```
template<typename T>
```

```
inline auto cosineDistance (const Eigen::Matrix<T, Eigen::Dynamic, Eigen::Dynamic> &m1, const  
                             Eigen::Matrix<T, Eigen::Dynamic, Eigen::Dynamic> &m2)
```

Computes the Cosine Distance between two matrices

Template Parameters

T – The type of the underlying matrix

Parameters

- **m1** – the first matrix
- **m2** – the second matrix

Returns

the Cosine Distance between the two matrices

```
template<typename T, int32_t MAP_OPTIONS, typename STRIDE>
inline auto cosineDistance (const Eigen::Map<T, MAP_OPTIONS, STRIDE> &m1, const Eigen::Map<T,
MAP_OPTIONS, STRIDE> &m2)
```

Computes the Cosine Distance between two matrices

 See also

Eigen::Map::MapOptions

Template Parameters

- **T** – The type of the underlying matrix
- **MAP_OPTIONS** – The options for the underlying matrix.
- **STRIDE** – The stride of the underlying matrix

Parameters

- **m1** – the first matrix
- **m2** – the second matrix

Returns

the Cosine Distance between the two matrices

```
inline auto cosineDistance (const cv::Mat &m1, const cv::Mat &m2)
```

Computes the Cosine Distance between two matrices

Template Parameters

T – The type of the underlying matrix

Parameters

- **m1** – the first matrix
- **m2** – the second matrix

Returns

the Cosine Distance between the two matrices

namespace **imu**

namespace **io**

Typedefs

```
using DataReadVariant = std::variant<dv::EventStore, dv::Frame, std::vector<dv::IMU>,
std::vector<dv::Trigger>, DataReadHandler::OutputFlag>
```

Enums

```
enum class ModeFlags : uint8_t
```

Values:

```
enumerator READ
```

```
enumerator WRITE
```

```
enum class WriteFlags : uint8_t
```

Values:

```
enumerator NONE
```

```
enumerator TRUNCATE
```

```
enumerator APPEND
```

```
enum class SeekFlags : int
```

Values:

```
enumerator START
```

```
enumerator CURRENT
```

```
enumerator END
```

Functions

```
inline ModeFlags operator| (const ModeFlags lhs, const ModeFlags rhs)
```

```
inline ModeFlags &operator|= (ModeFlags &lhs, const ModeFlags rhs)
```

```
inline bool operator& (const ModeFlags lhs, const ModeFlags rhs)
```

```
inline WriteFlags operator| (const WriteFlags lhs, const WriteFlags rhs)
```

```
inline WriteFlags &operator|= (WriteFlags &lhs, const WriteFlags rhs)
```

```
inline bool operator& (const WriteFlags lhs, const WriteFlags rhs)
```

```
namespace camera
```

Typedefs

```
using CameraPtr = std::unique_ptr<CameraInputBase>
```

```
using SyncCameraPtr = std::unique_ptr<SyncCameraInputBase>
```

Enums

enum class **CameraModel** : uint8_t

Values:

enumerator **DVS128**

enumerator **DAVIS**

enumerator **DVXPLOER**

enumerator **DVXPLOER_M**

enumerator **DVXPLOER_S**

enumerator **DAVIS_GEN2**

enumerator **DVXPLOER_GEN2**

enum class **USBDeviceType** : uint8_t

Values:

enumerator **FX2**

enumerator **FX3_MB**

enumerator **FX3_BLUE**

enumerator **FX3_RED**

enumerator **CX3_MIPI**

enumerator **FX3_GEN2**

Functions

inline *std::ostream* &**operator**<< (*std::ostream* &os, const *CameraModel* &var)

inline *std::ostream* &**operator**<< (*std::ostream* &os, const *USBDeviceType* &var)

static *std::vector*<*USBDevice::DeviceDescriptor*> **discover** (const *USBDevice::loggerCallbackType* &logger = {})

Discover all compatible cameras connected to this system.

Parameters

logger – discovery logging callback, must be thread-safe

Returns

list of device descriptor structures

```
static CameraPtr open (const USBDevice::DeviceDescriptor &descriptor, USBDevice::loggerCallbackType  
    deviceLogger = {}, const USBDevice::LogLevel deviceLogLevel =  
    USBDevice::LogLevel::LVL_WARNING)
```

Open the device with the specified descriptor structure. Throws if device described cannot be opened. The generic pointer returned only supports some functionality that all cameras implement. For more specific functionality, open the cameras directly via their class constructors, or up-cast with `std::dynamic_cast`.

Parameters

- **descriptor** – device descriptor to try opening
- **deviceLogger** – per-device logging callback, must be thread-safe
- **deviceLogLevel** – initial log-level

Returns

generic pointer to camera

```
static CameraPtr open (const std::string_view serialNumber, USBDevice::loggerCallbackType deviceLogger = {},  
    const USBDevice::LogLevel deviceLogLevel = USBDevice::LogLevel::LVL_WARNING)
```

Open the device with the specified serial number. Throws if device described cannot be opened. The generic pointer returned only supports some functionality that all cameras implement. For more specific functionality, open the cameras directly via their class constructors, or up-cast with `std::dynamic_cast`.

Parameters

- **serialNumber** – device serial number to try opening
- **deviceLogger** – per-device logging callback, must be thread-safe
- **deviceLogLevel** – initial log-level

Returns

generic pointer to camera

```
static CameraPtr open (USBDevice::loggerCallbackType deviceLogger = {}, const USBDevice::LogLevel  
    deviceLogLevel = USBDevice::LogLevel::LVL_WARNING)
```

Open the first device that can be found. Throws if device cannot be opened. The generic pointer returned only supports some functionality that all cameras implement. For more specific functionality, open the cameras directly via their class constructors, or up-cast with `std::dynamic_cast`.

Parameters

- **deviceLogger** – per-device logging callback, must be thread-safe
- **deviceLogLevel** – initial log-level

Returns

generic pointer to camera

```
static SyncCameraPtr openSync (const USBDevice::DeviceDescriptor &descriptor,  
    USBDevice::loggerCallbackType deviceLogger = {}, const USBDevice::LogLevel  
    deviceLogLevel = USBDevice::LogLevel::LVL_WARNING)
```

Open the device with the specified descriptor structure. Device must support clock synchronization. Throws if device described cannot be opened. The generic pointer returned only supports some functionality that all cameras implement. For more specific functionality, open the cameras directly via their class constructors, or up-cast with `std::dynamic_cast`.

Parameters

- **descriptor** – device descriptor to try opening
- **deviceLogger** – per-device logging callback, must be thread-safe

- **deviceLogLevel** – initial log-level

Returns

generic pointer to camera

```
static SyncCameraPtr openSync (const std::string_view serialNumber, USBDevice::loggerCallbackType
                                deviceLogger = {}, const USBDevice::LogLevel deviceLogLevel =
                                USBDevice::LogLevel::LVL_WARNING)
```

Open the device with the specified serial number. Device must support clock synchronization. Throws if device described cannot be opened. The generic pointer returned only supports some functionality that all cameras implement. For more specific functionality, open the cameras directly via their class constructors, or up-cast with `std::dynamic_cast`.

Parameters

- **serialNumber** – device serial number to try opening
- **deviceLogger** – per-device logging callback, must be thread-safe
- **deviceLogLevel** – initial log-level

Returns

generic pointer to camera

```
inline void synchronizeAny (const std::span<SyncCameraInputBase*> cameras)
```

Synchronize any number of cameras with each-other. Only one can be a master clock camera.

Parameters

cameras – cameras to synchronize.

```
inline void synchronizeAnyTwo (SyncCameraInputBase *first, SyncCameraInputBase *second)
```

Synchronize two cameras with each-other. Only one can be a master clock camera.

Parameters

- **first** – camera to synchronize.
- **second** – camera to synchronize.

```
inline void synchronizeAnyTwo (SyncCameraInputBase &first, SyncCameraInputBase &second)
```

Synchronize two cameras with each-other. Only one can be a master clock camera.

Parameters

- **first** – camera to synchronize.
- **second** – camera to synchronize.

```
inline void synchronizeAnyTwo (const SyncCameraPtr &first, const SyncCameraPtr &second)
```

Synchronize two cameras with each-other. Only one can be a master clock camera.

Parameters

- **first** – camera to synchronize.
- **second** – camera to synchronize.

namespace **imu**

Enums

enum class **ImuModel1** : uint8_t

List of supported *IMU* models.

Values:

enumerator **IMU_NONE**

enumerator **IMU_INVENSENSE_6050_6150**

enumerator **IMU_INVENSENSE_6500_9250**

enumerator **IMU_BOSCH_BMI_160**

enumerator **IMU_BOSCH_BMI_270**

enum class **InvensenseAccelRange** : uint8_t

List of accelerometer scale settings for InvenSense IMUs.

Values:

enumerator **RANGE_2G**

enumerator **RANGE_4G**

enumerator **RANGE_8G**

enumerator **RANGE_16G**

enum class **InvensenseGyroRange** : uint8_t

List of gyroscope scale settings for InvenSense IMUs.

Values:

enumerator **RANGE_250DPS**

enumerator **RANGE_500DPS**

enumerator **RANGE_1000DPS**

enumerator **RANGE_2000DPS**

enum class **BoschBMI160AccelRange** : uint8_t

List of accelerometer scale settings for Bosch *IMU*.

Values:

enumerator **RANGE_2G**

enumerator **RANGE_4G**

enumerator **RANGE_8G**

enumerator **RANGE_16G**

enum class **BoschBMI160AccelDataRate** : uint8_t

List of accelerometer data rate settings for Bosch *IMU*.

Values:

enumerator **RATE_12_5HZ**

enumerator **RATE_25HZ**

enumerator **RATE_50HZ**

enumerator **RATE_100HZ**

enumerator **RATE_200HZ**

enumerator **RATE_400HZ**

enumerator **RATE_800HZ**

enumerator **RATE_1600HZ**

enum class **BoschBMI160AccelFilter** : uint8_t

List of accelerometer filter settings for Bosch *IMU*.

Values:

enumerator **FILTER_OSR4**

enumerator **FILTER_OSR2**

enumerator **FILTER_NORMAL**

enum class **BoschBMI160GyroRange** : uint8_t

List of gyroscope scale settings for Bosch *IMU*.

Values:

enumerator **RANGE_2000DPS**

enumerator **RANGE_1000DPS**

enumerator **RANGE_500DPS**

enumerator **RANGE_250DPS**

enumerator **RANGE_125DPS**

enum class **BoschBMI160GyroDataRate** : uint8_t
List of gyroscope data rate settings for Bosch *IMU*.
Values:

enumerator **RATE_25HZ**

enumerator **RATE_50HZ**

enumerator **RATE_100HZ**

enumerator **RATE_200HZ**

enumerator **RATE_400HZ**

enumerator **RATE_800HZ**

enumerator **RATE_1600HZ**

enumerator **RATE_3200HZ**

enum class **BoschBMI160GyroFilter** : uint8_t
List of gyroscope filter settings for Bosch *IMU*.
Values:

enumerator **FILTER_OSR4**

enumerator **FILTER_OSR2**

enumerator **FILTER_NORMAL**

Functions

```
inline std::ostream &operator<< (std::ostream &os, const ImuModel &var)

static float boschBMI160CalculateIMUAccelScale (const uint8_t imuAccelScale)

static float boschBMI160CalculateIMUGyroScale (const uint8_t imuGyroScale)

static int16_t safeFlip16 (const int16_t value)
```

```
namespace parser
```

Typedefs

```
using ParserLoggerCallback = std::function<void(bool debug, std::string_view message)>

using ParserTimeInitCallback = std::function<void()>

using ParserDataCommitCallback = std::function<void(ParsedData buffers)>
```

```
namespace DAVIS
```

Enums

```
enum class SensorModel
    Values:
```

```
    enumerator DAVIS240A
```

```
    enumerator DAVIS240B
```

```
    enumerator DAVIS240C
```

```
    enumerator DAVIS346
```

```
    enumerator DAVIS640
```

```
    enumerator CDAVIS
```

```
enum class ColorMode
    Values:
```

```
    enumerator DEFAULT
```

```
    enumerator GRAYSCALE
```

enumerator **ORIGINAL**

enumerator **ORIGINAL_SPLIT**

Functions

inline *std::ostream* &**operator**<< (*std::ostream* &os, const *SensorModel* &var)

inline *std::ostream* &**operator**<< (*std::ostream* &os, const *ColorMode* &var)

namespace **DVS128**

Variables

static constexpr int16_t **WIDTH** = {128}

static constexpr int16_t **HEIGHT** = {128}

namespace **DVXplorer**

namespace **S5K231Y**

Variables

static constexpr int16_t **WIDTH** = {640}

static constexpr int16_t **HEIGHT** = {480}

namespace **S5KRC1S**

Variables

static constexpr int16_t **WIDTH** = {960}

static constexpr int16_t **HEIGHT** = {720}

namespace **compression**

Functions

static *std::unique_ptr*<*CompressionSupport*> **createCompressionSupport** (const *CompressionType* type)

static *std::unique_ptr*<*DecompressionSupport*> **createDecompressionSupport** (const *CompressionType*
type)

namespace **network**

Typedefs

```
using asioUNIX = asioLocal::stream_protocol
```

```
using asioTCP = asioIP::tcp
```

```
namespace encrypt
```

Functions

```
inline asioSSL::context createEncryptionContext (asioSSL::context::method method, const
std::filesystem::path &certificateChain, const
std::filesystem::path &privateKey, const
std::optional<std::filesystem::path> &CAFile =
std::nullopt)
```

Create an encryption context.

Parameters

- **method** – Encryption mode.
- **certificateChain** – Path to certificate chain.
- **privateKey** – Path to a private key.
- **CAFile** – Path to CAFile, if a std::nullopt is provided, peer verification is disabled. Can be an empty path, in that case the context will use CA from default locations, the peers will be verified.

Returns

Encryption context.

```
inline asioSSL::context defaultEncryptionServer (const std::filesystem::path &certificateChain, const
std::filesystem::path &privateKey, const
std::filesystem::path &CAFile)
```

Create an encryption server context with default configuration: TLSv1.2 encryption algorithm, provided certificate chain, server private key and certificate authority CAFile which is used to verify client certificate.

Parameters

- **certificateChain** – Server certificate chain.
- **privateKey** – Server private key.
- **CAFile** – CAFile for client verification.

Returns

SSL context that can be used for encrypted network connections.

```
inline asioSSL::context defaultEncryptionClient (const std::filesystem::path &certificateChain, const
std::filesystem::path &privateKey)
```

Create an encrypted client context with default configuration: TLSv1.2 encryption algorithm, provided client certificate chain and client private key. Server is always considered trusted and server certificate is not verified, the server will verify the client and can reject the connection during handshake if certificate verification fails.

Parameters

- **certificateChain** – Client certificate chain.
- **privateKey** – Client private key.

Returns

SSL context that can be used with encrypted network connections.

namespace **support**

Typedefs

using **TypeResolver** = *dv::std_function_exact*<const *dv::types::Type**(const uint32_t)>

using **VariantValueOwning** = *std::variant*<bool, int32_t, int64_t, float, double, *std::string*>

Functions

static inline const *dv::types::Type* ***defaultTypeResolver** (const uint32_t typeId)

template<class **PacketType**>

inline *std::shared_ptr*<*dv::types::TypedObject*> **packetToObject** (*PacketType* &&packet, const *TypeResolver* &resolver = *defaultTypeResolver*)

Variables

static constexpr *std::string_view* **AEDAT4_FILE_EXTENSION** = {"aodat4"}

static constexpr *std::string_view* **AEDAT4_HEADER_VERSION** = {"#!AER-DAT4.0\r\n"}

namespace **kinematics**

Typedefs

typedef *LinearTransformer*<float> **LinearTransformerf**
LinearTransformer using single precision float operations

typedef *LinearTransformer*<double> **LinearTransformerd**
LinearTransformer using double precision float operations

typedef *Transformation*<float> **Transformationf**
Transformation using single precision float operations

typedef *Transformation*<double> **Transformationd**
Transformation using double precision float operations

namespace **measurements**

namespace **noise**

Enums

enum class **FrequencyFilterType**

Values:

enumerator **PASS**

enumerator **CUT**

Functions

inline *std::ostream* &**operator<<** (*std::ostream* &os, const *FrequencyFilterType* &var)

namespace **optimization**

namespace **packets**

Enums

enum class **Timestamp**

Values:

enumerator **START**

enumerator **END**

Functions

template<class **ElementType**>

inline int64_t **getTimestamp** (const *ElementType* &element)

Template method that retrieves timestamp from a Timestamped structure.

Template Parameters

ElementType – Type of the element

Parameters

element – Instance of the element

Returns

Timestamp of this element

template<class **PacketType**>

inline bool **isPacketEmpty** (const *PacketType* &packet)

Check if a packet is empty.

Template Parameters

PacketType –

Parameters

packet –

Returns

True if the given packet is empty, false otherwise.

```
template<class PacketType>
inline size_t getPacketSize (const PacketType &packet)
```

Get packet size. This utility template method can be used to generically get size of a EventStore, data packet or any container satisfying the iterable concept.

Template Parameters

PacketType –

Parameters

packet –

Returns

Size of the given packet

```
template<class PacketType>
inline auto getPacketBegin (const PacketType &packet)
```

Generic getter of a begin iterator of a packet.

Template Parameters

PacketType –

Parameters

packet –

Returns

```
template<class PacketType>
inline auto getPacketEnd (const PacketType &packet)
```

Generic getter of an end iterator of a packet.

Template Parameters

PacketType –

Parameters

packet –

Returns

```
template<Timestamp startTime, class PacketType>
inline int64_t getPacketTimestamp (const PacketType &packet)
```

Retrieve packet start or end timestamp using template generation.

Template Parameters

- **startTime** – Use enum to select whether you want start or end timestamp.
- **PacketType** – Packet type, inferred from argument type.

Parameters

packet – Non-empty data packet.

Throws

`InvalidArgument` – exception is thrown if the packet is empty.

Returns

Timestamp of the first or last element in the packet.

```
template<class PacketType>
inline dv::TimeWindow getPacketTimeWindow (const PacketType &packet)
```

Get time window for a given packet.

Template Parameters

PacketType –

Parameters

packet – Non-empty data packet.

Throws

InvalidArgument – exception is thrown if the packet is empty.

Returns

Time window with start and end timestamps of this packet.

namespace **types**

Typedefs

using **PackFuncPtr** = *std::add_pointer_t*<uint32_t(void *toFlatBufferBuilder, const void *fromObject)>

using **UnpackFuncPtr** = *std::add_pointer_t*<void(void *toObject, const void *fromFlatBuffer)>

using **ConstructPtr** = *std::add_pointer_t*<void*(const size_t sizeOfObject)>

using **DestructPtr** = *std::add_pointer_t*<void(void *object)>

using **TimeElementExtractorPtr** = *std::add_pointer_t*<void(const void *object, *TimeElementExtractor* *rangeOut)>

using **TimeRangeExtractorPtr** = *std::add_pointer_t*<void(void *toObject, const void *fromObject, const *TimeElementExtractor* *rangeIn, uint32_t *commitNowOut, uint32_t *exceedsTimeRangeOut)>

Functions

constexpr uint32_t **IdentifierStringToId** (const *std::string_view* id) noexcept

constexpr *std::array*<char, 5> **IdToIdentifierString** (const uint32_t id) noexcept

template<typename **ObjectAPIType**>
inline uint32_t **Packer** (void *toFlatBufferBuilder, const void *fromObject)

template<typename **ObjectAPIType**>
inline void **Unpacker** (void *toObject, const void *fromFlatBuffer)

template<typename **ObjectAPIType**, typename **SubObjectAPIType**>
inline void **TimeElementExtractorDefault** (const void *object, *TimeElementExtractor* *rangeOut) noexcept

template<typename **ObjectAPIType**, typename **SubObjectAPIType**>
inline void **TimeRangeExtractorDefault** (void *toObject, const void *fromObject, const *TimeElementExtractor* *rangeIn, uint32_t *commitNowOut, uint32_t *exceedsTimeRangeAndKeepPacketOut)

template<typename **ObjectAPIType**, typename **SubObjectAPIType**>
constexpr *Type* **makeTypeDefinition** ()

namespace **visualization**

namespace **colors**

Functions

```
inline cv::Scalar someNeonColor (const int32_t someNumber)
```

Variables

```
static const cv::Scalar black = cv::Scalar(0, 0, 0)
```

```
static const cv::Scalar darkGray = cv::Scalar(43, 43, 43)
```

```
static const cv::Scalar gray = cv::Scalar(128, 128, 128)
```

```
static const cv::Scalar silver = cv::Scalar(192, 192, 192)
```

```
static const cv::Scalar white = cv::Scalar(255, 255, 255)
```

```
static const cv::Scalar red = cv::Scalar(0, 0, 255)
```

```
static const cv::Scalar lime = cv::Scalar(0, 255, 0)
```

```
static const cv::Scalar blue = cv::Scalar(255, 0, 0)
```

```
static const cv::Scalar yellow = cv::Scalar(0, 255, 255)
```

```
static const cv::Scalar cyan = cv::Scalar(255, 255, 0)
```

```
static const cv::Scalar magenta = cv::Scalar(255, 0, 255)
```

```
static const cv::Scalar maroon = cv::Scalar(0, 0, 128)
```

```
static const cv::Scalar green = cv::Scalar(0, 128, 0)
```

```
static const cv::Scalar navy = cv::Scalar(128, 0, 0)
```

```
static const cv::Scalar iniBlue = cv::Scalar(183, 93, 0)
```

```
static const std::vector<cv::Scalar> neonPalette = {cv::Scalar(255, 111, 0), cv::Scalar(239, 244, 19),  
cv::Scalar(0, 255, 104), cv::Scalar(0, 255, 250), cv::Scalar(0, 191, 255), cv::Scalar(0, 191, 255), cv::Scalar(92, 0,  
255)}
```

```
namespace dv_capture_node
```

```
namespace dv_runtime
```

```
namespace flatbuffers
```

```
namespace fmt
```

```
namespace std
```

```
file calibration_set.hpp
```

```
#include    "../core/utils.hpp"#include    "../exception/exception.hpp"#include    "calibrations/camera_calibration.hpp"#include    "calibrations/imu_calibration.hpp"#include    "calibrations/stereo_calibration.hpp"#include    <boost/algorithm/string.hpp>#include    <boost/property_tree/json_parser.hpp>#include    <boost/property_tree/ptree.hpp>#include    <opencv2/core.hpp>#include    <iostream>#include    <map>#include    <regex>#include    <vector>
```

```
file camera_calibration.hpp
```

```
#include    "../camera_geometry.hpp"#include    <Eigen/Core>#include    <boost/property_tree/ptree.hpp>#include    <opencv2/core.hpp>#include    <optional>#include    <span>
```

```
file imu_calibration.hpp
```

```
#include    "camera_calibration.hpp"
```

```
file stereo_calibration.hpp
```

```
#include    "camera_calibration.hpp"
```

```
file camera_geometry.hpp
```

```
#include    "../core/core.hpp"#include    "../data/timed_keypoint_base.hpp"#include    "../exception/exception.hpp"#include    "../kinematics/linear_transformer.hpp"#include    <Eigen/Core>#include    <opencv2/calib3d.hpp>#include    <opencv2/core.hpp>#include    <opencv2/core/eigen.hpp>#include    <cmath>#include    <span>#include    <vector>
```

```
file stereo_geometry.hpp
```

```
#include    "../core/utils.hpp"#include    "../data/depth_frame_base.hpp"#include    "calibrations/camera_calibration.hpp"#include    "camera_geometry.hpp"#include    <opencv2/imgproc.hpp>
```

```
file mean_shift.hpp
```

```
#include    "mean_shift/eigen_matrix_adaptor.hpp"#include    "mean_shift/event_store_adaptor.hpp"
```

```
file eigen_matrix_adaptor.hpp
```

```
#include    "../containers/kd_tree.hpp"#include    "kernel.hpp"#include    <Eigen/Dense>#include    <optional>#include    <random>#include    <vector>
```

```
file eigen_matrix_adaptor.hpp
```

```
#include    "../external/nanoflann/nanoflann.hpp"#include    <Eigen/Dense>#include    <memory>
```

```
file event_store_adaptor.hpp
```

```
#include    "../containers/kd_tree.hpp"#include    "kernel.hpp"#include    <optional>#include    <random>#include    <vector>
```

file **event_store_adaptor.hpp**

```
#include "../external/nanoflann/nanoflann.hpp" #include "../core/core.hpp" #include "../data/timed_keypoint_base.hpp" #include <opencv2/core.hpp> #include <memory>
```

file **kernel.hpp**

```
#include <cmath> #include <concepts>
```

file **kd_tree.hpp**

```
#include "kd_tree/eigen_matrix_adaptor.hpp" #include "kd_tree/event_store_adaptor.hpp"
```

file **concepts.hpp**

```
#include "../data/bounding_box_base.hpp" #include "../data/depth_event_base.hpp" #include "../data/depth_frame_base.hpp" #include "../data/event_base.hpp" #include "../data/frame_base.hpp" #include "../data/imu_base.hpp" #include "../data/landmark_base.hpp" #include "../data/pose_base.hpp" #include "../data/timed_keypoint_base.hpp" #include "../data/trigger_base.hpp" #include <Eigen/Core> #include <opencv2/core.hpp> #include <concepts> #include <iterator> #include <type_traits>
```

file **core.hpp**

```
#include "../data/depth_event_base.hpp" #include "../data/event_base.hpp" #include "../data/frame_base.hpp" #include "stream_slicer.hpp" #include "utils.hpp" #include <Eigen/Dense> #include <opencv2/core.hpp> #include <opencv2/core/eigen.hpp> #include <iostream> #include <map> #include <numeric> #include <optional>
```

Functions

```
inline std::ostream &operator<< (std::ostream &os, const cv::Size &var)
```

```
inline std::ostream &operator<< (std::ostream &os, const cv::Point &var)
```

```
inline std::ostream &operator<< (std::ostream &os, const cv::Rect &var)
```

file **dvassert.hpp**

```
#include <boost/stacktrace.hpp> #include "concepts.hpp" #include <fmt/ranges.h> #include <fmt/std.h> #include <cstdlib> #include <filesystem> #include <source_location> #include <string_view>
```

file **event.hpp**

```
#include "core.hpp" #include "filters.hpp"
```

file **event_color.hpp**

```
#include "../core/utils.hpp" #include "../data/event_base.hpp"
```

file **filters.hpp**

```
#include "../core/frame.hpp" #include "../exception/exceptions/generic_exceptions.hpp"
```

file **frame.hpp**

```
#include "frame/accumulator.hpp" #include "frame/accumulator_base.hpp" #include "frame/edge_map_accumulator.hpp"
```

```

file accumulator.hpp
    #include "accumulator_base.hpp"

file accumulator_base.hpp
    #include "../core.hpp"

file edge_map_accumulator.hpp
    #include "accumulator_base.hpp"

file multi_stream_slicer.hpp
    #include "../data/frame_base.hpp" #include "../data/imu_base.hpp" #include "../data/trigger_base.hpp" #include
    "../exception/exceptions/generic_exceptions.hpp" #include "core.hpp" #include "stream_slicer.hpp" #include
    <unordered_map> #include <variant>

file stereo_event_stream_slicer.hpp
    #include "core.hpp"

file stream_slicer.hpp
    #include "../exception/exceptions/generic_exceptions.hpp" #include "concepts.hpp" #include "time_win-
    dow.hpp" #include "utils.hpp" #include <functional> #include <map>

file time.hpp
    #include <chrono>

file time_window.hpp
    #include "time.hpp"

file utils.hpp
    #include "../exception/exceptions/generic_exceptions.hpp" #include "concepts.hpp" #include "dvassert.hpp" #include
    "time.hpp" #include "time_window.hpp" #include <fmt/chrono.h> #include <fmt/format.h> #include <fmt/os-
    tream.h> #include <fmt/ranges.h> #include <fmt/std.h> #include <algorithm> #include <array> #include
    <cerrno> #include <cintrypes> #include <compare> #include <cstdint> #include <cstdddef> #include <cstddint> #include <cst-
    dlib> #include <cstring> #include <filesystem> #include <functional> #include <memory> #include <string> #include
    <string_view> #include <utility> #include <vector>

file utils.hpp
    #include <opencv2/calib3d.hpp>

file utils.hpp
    #include "../core/utils.hpp" #include "../data/bounding_box_base.hpp" #include
    "../data/depth_event_base.hpp" #include "../data/depth_frame_base.hpp" #include
    "../data/event_base.hpp" #include "../data/frame_base.hpp" #include "../data/imu_base.hpp" #include
    "../data/landmark_base.hpp" #include "../data/pose_base.hpp" #include "../data/timed_key-
    point_base.hpp" #include "../data/trigger_base.hpp" #include "../data/types.hpp" #include "../data/FileDataT-
    able.hpp" #include "../data/IOHeader.hpp" #include "io_data_buffer.hpp" #include "io_statistics.hpp" #include
    <string_view>

```

file **boost_geometry_interop.hpp**

```
#include "bounding_box_base.hpp" #include "event_base.hpp" #include "timed_keypoint_base.hpp" #include
<boost/geometry/core/cs.hpp> #include <boost/geometry/geometries/register/box.hpp> #include <boost/geometry/ge-
ometries/register/point.hpp> #include <boost/geometry/geometry.hpp> #include <opencv2/core.hpp>
```

file **bounding_box_base.hpp**

```
#include "../external/flatbuffers/flatbuffers.h" #include <fmt/chrono.h> #include <fmt/format.h> #include <fmt/os-
tream.h> #include <fmt/ranges.h>
```

Variables

VT_TIMESTAMP = 4

VT_TOPLEFTX = 6

VT_TOPLEFTY = 8

VT_BOTTOMRIGHTX = 10

VT_BOTTOMRIGHTY = 12

VT_CONFIDENCE = 14

file **depth_event_base.hpp**

```
#include "../external/flatbuffers/flatbuffers.h" #include <fmt/chrono.h> #include <fmt/format.h> #include <fmt/os-
tream.h> #include <fmt/ranges.h>
```

file **depth_frame_base.hpp**

```
#include "../external/flatbuffers/flatbuffers.h" #include <fmt/chrono.h> #include <fmt/format.h> #include <fmt/os-
tream.h> #include <fmt/ranges.h>
```

Variables

VT_TIMESTAMP = 4

VT_SIZEX = 6

VT_SIZEY = 8

VT_MINDEPTH = 10

VT_MAXDEPTH = 12

VT_STEP = 14

file **event_base.hpp**

```
#include "../external/flatbuffers/flatbuffers.h"#include <fmt/chrono.h>#include <fmt/format.h>#include <fmt/ostream.h>#include <fmt/ranges.h>
```

file **frame_base.hpp**

```
#include "../external/flatbuffers/flatbuffers.h"#include "../core/time.hpp"#include <fmt/chrono.h>#include <fmt/format.h>#include <fmt/ostream.h>#include <fmt/ranges.h>#include <opencv2/core/mat.hpp>#include <ostream>
```

Variables

```
VT_TIMESTAMP      = 4
```

```
VT_TIMESTAMPSTARTOFFFRAME  = 6
```

```
VT_TIMESTAMPENDOFFFRAME   = 8
```

```
VT_TIMESTAMPSTARTOFEXPOSURE = 10
```

```
VT_TIMESTAMPENDOFEXPOSURE  = 12
```

```
VT_FORMAT      = 14
```

```
VT_SIZEX      = 16
```

```
VT_SIZEY      = 18
```

```
VT_POSITIONX   = 20
```

```
VT_POSITIONY   = 22
```

```
VT_PIXELS      = 24
```

```
VT_EXPOSURE     = 26
```

file **generate.hpp**

```
#include "../core/core.hpp"#include "../exception/exceptions/generic_exceptions.hpp"#include "../visualization/colors.hpp"#include <opencv2/imgproc.hpp>#include <random>
```

file **geometry_types_base.hpp**

```
#include "../external/flatbuffers/flatbuffers.h"
```

file **imu_base.hpp**

```
#include "../external/flatbuffers/flatbuffers.h"#include <Eigen/Core>#include <fmt/chrono.h>#include <fmt/format.h>#include <fmt/ostream.h>#include <fmt/ranges.h>#include <numbers>#include <ostream>
```

Variables

VT_TIMESTAMP = 4

VT_TEMPERATURE = 6

VT_ACCELEROMETERX = 8

VT_ACCELEROMETERY = 10

VT_ACCELEROMETERZ = 12

VT_GYROSCOPEX = 14

VT_GYROSCOPEY = 16

VT_GYROSCOPEZ = 18

VT_MAGNETOMETERX = 20

VT_MAGNETOMETERY = 22

file landmark_base.hpp

```
#include        "../external/flatbuffers/flatbuffers.h"#include        "geometry_types_base.hpp"#include  
<fmt/chrono.h>#include <fmt/format.h>#include <fmt/ostream.h>#include <fmt/ranges.h>
```

Variables

VT_TRACKID = 4

VT_CAMERAID = 6

VT_CAMERANAME = 8

VT_PT = 4

VT_ID = 6

VT_TIMESTAMP = 8

VT_DESCRIPTOR = 10

VT_DESCRIPTOR_TYPE = 12

```
VT_COVARIANCE    = 14
```

```
VT_ELEMENTS      = 4
```

file pose_base.hpp

```
#include          "../external/flatbuffers/flatbuffers.h" #include          "geometry_types_base.hpp" #include
<fmt/chrono.h> #include <fmt/format.h> #include <fmt/ostream.h> #include <fmt/ranges.h>
```

Variables

```
VT_TIMESTAMP      = 4
```

```
VT_TRANSLATION    = 6
```

```
VT_ROTATION       = 8
```

```
VT_REFERENCEFRAME = 10
```

file timed_keypoint_base.hpp

```
#include          "../external/flatbuffers/flatbuffers.h" #include          "geometry_types_base.hpp" #include
<fmt/chrono.h> #include <fmt/format.h> #include <fmt/ostream.h> #include <fmt/ranges.h>
```

Variables

```
VT_PT             = 4
```

```
VT_SIZE           = 6
```

```
VT_ANGLE          = 8
```

```
VT_RESPONSE       = 10
```

```
VT_OCTAVE         = 12
```

```
VT_CLASS_ID       = 14
```

file trigger_base.hpp

```
#include          "../external/flatbuffers/flatbuffers.h" #include <fmt/chrono.h> #include <fmt/format.h> #include <fmt/os-
tream.h> #include <fmt/ranges.h>
```

Variables

```
VT_TIMESTAMP      = 4
```

file **types.hpp**

```
#include    "../external/flatbuffers/flatbuffers.h"#include    "../core/utils.hpp"#include    "../exception/exceptions/generic_exceptions.hpp"
```

file **utilities.hpp**

```
#include    "../core/core.hpp"#include    "../kinematics/transformation.hpp"#include    "depth_event_base.hpp"#include    "depth_frame_base.hpp"#include    "event_base.hpp"#include    "pose_base.hpp"#include    "timed_keypoint_base.hpp"#include    <opencv2/core.hpp>
```

file **semi_dense_stereo_matcher.hpp**

```
#include    "../camera/stereo_geometry.hpp"#include    "../core/concepts.hpp"#include    "../core/frame.hpp"#include    "utils.hpp"
```

file **sparse_event_block_matcher.hpp**

```
#include    "../camera/stereo_geometry.hpp"#include    "../core/filters.hpp"#include    "../core/frame.hpp"#include    <opencv2/imgproc.hpp>
```

file **exception.hpp**

```
#include    "exceptions/directory_exceptions.hpp"#include    "exceptions/file_exceptions.hpp"#include    "exceptions/generic_exceptions.hpp"#include    "exceptions/io_exceptions.hpp"#include    "exceptions/type_exceptions.hpp"
```

file **exception_base.hpp**

```
#include    <boost/stacktrace.hpp>#include    <boost/core/demangle.hpp>#include    <fmt/chrono.h>#include    <fmt/format.h>#include    <fmt/ostream.h>#include    <fmt/ranges.h>#include    <fmt/std.h>#include    <concepts>#include    <filesystem>#include    <source_location>#include    <stdexcept>#include    <string>
```

file **directory_exceptions.hpp**

```
#include    "../exception_base.hpp"
```

file **file_exceptions.hpp**

```
#include    "../exception_base.hpp"
```

file **generic_exceptions.hpp**

```
#include    "../exception_base.hpp"
```

file **io_exceptions.hpp**

```
#include    "../exception_base.hpp"
```

file **type_exceptions.hpp**

```
#include    "../exception_base.hpp"
```

file **arc_corner_detector.hpp**

```
#include    "../core/concepts.hpp"#include    "../core/core.hpp"#include    "../data/timed_keypoint_base.hpp"#include    <Eigen/Dense>#include    <opencv2/core.hpp>
```

file **event_blob_detector.hpp**

```
#include "../core/event.hpp"#include "../core/frame/accumulator.hpp"#include "../data/utilities.hpp"#include
<opencv2/opencv.hpp>#include <atomic>#include <utility>
```

```
file event_combined_lk_tracker.hpp
```

```
#include "../core/core.hpp"#include "../core/frame.hpp"#include "../data/utilities.hpp"#include "image_fea-
ture_lk_tracker.hpp"
```

```
file event_feature_lk_tracker.hpp
```

```
#include "../core/frame.hpp"#include "image_feature_lk_tracker.hpp"
```

```
file feature_detector.hpp
```

```
#include "../core/concepts.hpp"#include "../core/core.hpp"#include "../data/timed_keypoint_base.hpp"#include
../data/utilities.hpp"#include "event_blob_detector.hpp"#include "image_pyramid.hpp"#include "keypoint_re-
sampler.hpp"#include <opencv2/core.hpp>#include <opencv2/features2d.hpp>
```

```
file feature_tracks.hpp
```

```
#include "../core/utls.hpp"#include "../exception/exceptions/generic_exceptions.hpp"#include "../visualization/col-
ors.hpp"#include "tracker_base.hpp"
```

```
file image_feature_lk_tracker.hpp
```

```
#include "../data/utilities.hpp"#include "../exception/exceptions/generic_exceptions.hpp"#include "../kine-
matics/motion_compensator.hpp"#include "image_pyramid.hpp"#include "redetection_strategy.hpp"#include
"tracker_base.hpp"#include <utility>
```

```
file image_pyramid.hpp
```

```
#include "../data/frame_base.hpp"#include <opencv2/core.hpp>#include <opencv2/video.hpp>#include <memory>
```

```
file keypoint_resampler.hpp
```

```
#include "../core/concepts.hpp"#include "../data/boost_geometry_interop.hpp"#include <boost/geometry/geome-
try.hpp>#include <boost/geometry/index/rtree.hpp>
```

```
file mean_shift_tracker.hpp
```

```
#include "../core/core.hpp"#include "../exception/exceptions/generic_exceptions.hpp"#include "feature_detec-
tor.hpp"#include "redetection_strategy.hpp"#include "tracker_base.hpp"
```

```
file redetection_strategy.hpp
```

```
#include "tracker_base.hpp"
```

```
file tracker_base.hpp
```

```
#include "feature_detector.hpp"
```

```
file imgproc.hpp
```

```
#include "../core/core.hpp"#include <Eigen/Dense>#include <opencv2/core.hpp>#include
<opencv2/opencv.hpp>#include <optional>
```

```
file rotation-integrator.hpp
```

```
#include    "../core/concepts.hpp"#include    "../data/imu_base.hpp"#include    "../kinematics/transformation.hpp"#include <Eigen/Geometry>#include <numbers>
```

file camera_input_base.hpp

```
#include    "../data_read_handler.hpp"#include    "../input_base.hpp"#include    "imu_support.hpp"#include    "parsers/parser.hpp"#include <boost/circular_buffer.hpp>
```

file camera_model.hpp

```
#include "../core/utils.hpp"
```

file davis.hpp

```
#include "imu_support.hpp"#include "parsers/davis_parser.hpp"#include "sync_camera_input_base.hpp"#include "usb_device.hpp"
```

file discovery.hpp

```
#include "../exception/exceptions/generic_exceptions.hpp"#include "davis.hpp"#include "dvs128.hpp"#include "dvxplorer.hpp"#include "dvxplorer_m.hpp"
```

file dvs128.hpp

```
#include "parsers/dvs128_parser.hpp"#include "sync_camera_input_base.hpp"#include "usb_device.hpp"
```

file dvxplorer.hpp

```
#include    "imu_support.hpp"#include    "parsers/dvxplorer_parser.hpp"#include    "sync_camera_input_base.hpp"#include "usb_device.hpp"
```

file dvxplorer_m.hpp

```
#include    "camera_input_base.hpp"#include    "dvxplorer.hpp"#include    "imu_support.hpp"#include    "parsers/s5k231y_parser.hpp"#include "parsers/s5krc1s_parser.hpp"#include "usb_device_nextgen.hpp"
```

file imu_support.hpp

```
#include "../core/utils.hpp"
```

file davis_parser.hpp

```
#include "../core/event_color.hpp"#include "../imu_support.hpp"#include "parser.hpp"#include <boost/endian.hpp>#include <opencv2/imgproc.hpp>
```

file dvs128_parser.hpp

```
#include "../core/dvassert.hpp"#include "parser.hpp"#include <boost/endian.hpp>
```

file dvxplorer_parser.hpp

```
#include "../core/dvassert.hpp"#include "../imu_support.hpp"#include "parser.hpp"#include <boost/endian.hpp>
```

file parser.hpp

```
#include "../core/utils.hpp"#include "../data/event_base.hpp"#include "../data/frame_base.hpp"#include "../data/imu_base.hpp"#include "../data/trigger_base.hpp"#include <functional>#include <span>
```

```
file s5k231y_parser.hpp
    #include "../core/dvassert.hpp" #include "parser.hpp" #include <boost/endian.hpp>
```

```
file s5krc1s_parser.hpp
    #include "../core/dvassert.hpp" #include "parser.hpp" #include <boost/endian.hpp>
```

```
file sync_camera_input_base.hpp
    #include "camera_input_base.hpp"
```

```
file usb_device.hpp
    #include "../core/utls.hpp" #include "../support/thread_extra.hpp" #include "camera_model.hpp" #include
    <libusb.h> #include <boost/endian.hpp> #include <atomic> #include <shared_mutex> #include <thread> #include
    <unordered_map>
```

Defines

DV_LIBUSB_VERSION_1_0_23

DV_LIBUSB_VERSION_1_0_24

DV_LIBUSB_VERSION_1_0_25_26

DV_LIBUSB_VERSION_1_0_27_28

DV_LIBUSB_VERSION_1_0_29

```
file usb_device_nextgen.hpp
    #include "usb_device.hpp"
```

```
file compression_support.hpp
    #include "../core/utls.hpp" #include "../data/IOHeader.hpp" #include "../support/io_data_buffer.hpp" #include
    <lz4.h> #include <lz4frame.h> #include <lz4hc.h> #include <memory> #include <vector> #include <zstd.h>
```

Defines

LZ4F_HEADER_SIZE_MAX

ZSTD_CLEVEL_DEFAULT

```
file decompression_support.hpp
    #include "../core/utls.hpp" #include "../data/IOHeader.hpp" #include <lz4.h> #include <lz4frame.h> #include
    <lz4hc.h> #include <memory> #include <vector> #include <zstd.h>
```

Defines

LZ4F_HEADER_SIZE_MAX

ZSTD_CLEVEL_DEFAULT

file **FileDataTable.hpp**

#include "../external/flatbuffers/flatbuffers.h"

Variables

VT_BYTEOFFSET = 4

VT_PACKETINFO = 6

VT_NUMELEMENTS = 8

VT_TIMESTAMPSTART = 10

file **IOHeader.hpp**

#include "../external/flatbuffers/flatbuffers.h"

Variables

VT_COMPRESSION = 4

VT_DATATABLEPOSITION = 6

file **data_read_handler.hpp**

#include "../core/core.hpp"#include "../core/frame.hpp"#include "../data/imu_base.hpp"#include "../data/trigger_base.hpp"#include <functional>#include <optional>#include <variant>

file **input_base.hpp**

#include "../core/core.hpp"#include "../data/event_base.hpp"#include "../data/frame_base.hpp"#include "../data/imu_base.hpp"#include "../data/trigger_base.hpp"#include <optional>#include <string>

file **mono_camera_recording.hpp**

#include "../core/frame.hpp"#include "../exception/exceptions/generic_exceptions.hpp"#include "data_read_handler.hpp"#include "input_base.hpp"#include "read_only_file.hpp"#include <functional>#include <optional>

file **mono_camera_writer.hpp**

#include "../core/core.hpp"#include "../core/frame.hpp"#include "../exception/exceptions/generic_exceptions.hpp"#include "camera/camera_input_base.hpp"#include "output_base.hpp"#include "reader.hpp"#include "support/utils.hpp"#include "support/xml_config_io.hpp"#include "write_only_file.hpp"

```
file encrypt.hpp
    #include <boost/asio/ssl.hpp>#include <filesystem>#include <optional>

file socket_base.hpp
    #include <boost/asio.hpp>

file tcp_tls_socket.hpp
    #include "encrypt.hpp"#include "socket_base.hpp"#include <deque>#include <mutex>#include <utility>

file unix_socket.hpp
    #include "socket_base.hpp"#include <deque>#include <mutex>#include <utility>

file write_ordered_socket.hpp
    #include "socket_base.hpp"#include <deque>#include <functional>#include <utility>

file network_reader.hpp
    #include "input_base.hpp"#include "network/encrypt.hpp"#include "network/tcp_tls_socket.hpp"#include "net-
work/unix_socket.hpp"#include "reader.hpp"#include <boost/lockfree/spsc_queue.hpp>

file network_writer.hpp
    #include "network/socket_base.hpp"#include "network/tcp_tls_socket.hpp"#include "net-
work/unix_socket.hpp"#include "network/write_ordered_socket.hpp"#include "output_base.hpp"#include
"stream.hpp"#include "support/utls.hpp"#include "writer.hpp"#include <boost/lockfree/spsc_queue.hpp>#include
<utility>

file output_base.hpp
    #include "../core/core.hpp"

file read_only_file.hpp
    #include "reader.hpp"#include "simplefile.hpp"

file reader.hpp
    #include "compression/decompression_support.hpp"#include "stream.hpp"#include <boost/endian.hpp>#include
<optional>#include <unordered_map>#include <utility>

file simplefile.hpp
    #include "../core/utls.hpp"#include "../exception/exceptions/file_exceptions.hpp"#include "../exception/excep-
tions/generic_exceptions.hpp"#include <boost/nowide/cstdio.hpp>#include <algorithm>#include <cstdio>#include
<filesystem>#include <limits>#include <span>

file stereo_camera_recording.hpp
    #include "mono_camera_recording.hpp"

file stereo_camera_writer.hpp
    #include "camera/sync_camera_input_base.hpp"#include "mono_camera_writer.hpp"
```

file **stream.hpp**

```
#include "support/utils.hpp"#include "support/xml_config_io.hpp"#include <opencv2/core.hpp>#include <optional>
```

file **io_data_buffer.hpp**

```
#include "../data/FileDataTable.hpp"#include <vector>
```

file **io_statistics.hpp**

```
#include <cstdint>
```

file **thread_extra.hpp**

```
#include <cstring>#include <string>
```

Defines

```
PACKED_STRUCT (STRUCT_DECLARATION)
```

file **xml_config_io.hpp**

```
#include "../core/utils.hpp"#include <boost/property_tree/ptree.hpp>#include <boost/property_tree/xml_parser.hpp>#include <map>#include <sstream>#include <variant>
```

file **write_only_file.hpp**

```
#include "simplefile.hpp"#include "writer.hpp"#include <atomic>#include <mutex>#include <queue>#include <thread>
```

file **writer.hpp**

```
#include "compression/compression_support.hpp"#include "support/utils.hpp"#include <iostream>#include <memory>
```

file **linear_transformer.hpp**

```
#include "../core/dvassert.hpp"#include "transformation.hpp"#include <Eigen/Dense>#include <Eigen/StdVector>#include <boost/circular_buffer.hpp>#include <optional>
```

file **motion_compensator.hpp**

```
#include "../core/concepts.hpp"#include "../core/frame.hpp"#include "../exception/exceptions/generic_exceptions.hpp"#include "../measurements/depth.hpp"#include "linear_transformer.hpp"#include "pixel_motion_predictor.hpp"
```

file **pixel_motion_predictor.hpp**

```
#include "../camera/camera_geometry.hpp"#include <utility>
```

file **transformation.hpp**

```
#include "../core/concepts.hpp"#include <Eigen/Core>#include <opencv2/core/eigen.hpp>
```

file **depth.hpp**

```
#include <cstdint>
```

```
file background_activity_noise_filter.hpp
    #include "../core/filters.hpp"
```

```
file fast_decay_noise_filter.hpp
    #include "../core/filters.hpp"
```

```
file frequency_filters.hpp
    #include "../core/filters.hpp" #include <optional>
```

```
file k_noise_filter.hpp
    #include "../core/filters.hpp" #include <optional>
```

```
file contrast_maximization_rotation.hpp
    #include          "../camera/camera_geometry.hpp" #include          "../core/core.hpp" #include
    "../imu/rotation-integrator.hpp" #include  "../kinematics/motion_compensator.hpp" #include  "../optimization/opti-
    mization_functor.hpp"
```

```
file contrast_maximization_translation_and_depth.hpp
    #include          "../camera/camera_geometry.hpp" #include          "../core/core.hpp" #include
    "../imu/rotation-integrator.hpp" #include  "../kinematics/motion_compensator.hpp" #include  "../optimization/opti-
    mization_functor.hpp"
```

```
file contrast_maximization_wrapper.hpp
    #include  "../core/concepts.hpp" #include  "../optimization/optimization_functor.hpp" #include  <memory> #include
    <unsupported/Eigen/NonLinearOptimization> #include  <unsupported/Eigen/NumericalDiff>
```

```
file optimization_functor.hpp
    #include <Eigen/Dense>
```

```
file processing.hpp
    #include          "camera/calibration_set.hpp" #include          "camera/calibrations/camera_calibration.hpp" #include
    "camera/calibrations/imu_calibration.hpp" #include          "camera/calibrations/stereo_calibra-
    tion.hpp" #include          "camera/camera_geometry.hpp" #include          "camera/stereo_geometry.hpp" #include
    "cluster/mean_shift.hpp" #include          "containers/kd_tree.hpp" #include          "core/core.hpp" #include
    "core/event.hpp" #include  "core/event_color.hpp" #include  "core/filters.hpp" #include  "core/frame.hpp" #include
    "core/multi_stream_slicer.hpp" #include          "core/stereo_event_stream_slicer.hpp" #include
    "core/stream_slicer.hpp" #include  "core/time.hpp" #include  "core/utils.hpp" #include  "data/boost_geom-
    etry_interop.hpp" #include  "data/bounding_box_base.hpp" #include  "data/depth_event_base.hpp" #include
    "data/depth_frame_base.hpp" #include  "data/event_base.hpp" #include  "data/frame_base.hpp" #include
    "data/generate.hpp" #include  "data/geometry_types_base.hpp" #include  "data/imu_base.hpp" #include
    "data/landmark_base.hpp" #include  "data/pose_base.hpp" #include  "data/timed_keypoint_base.hpp" #include
    "data/trigger_base.hpp" #include          "data/types.hpp" #include          "data/utilities.hpp" #include
    "depth/semi_dense_stereo_matcher.hpp" #include  "depth/sparse_event_block_matcher.hpp" #include  "ex-
    ception/exception.hpp" #include  "features/arc_corner_detector.hpp" #include  "features/event_blob_de-
    tector.hpp" #include  "features/event_combined_lk_tracker.hpp" #include  "features/event_fea-
    ture_lk_tracker.hpp" #include  "features/feature_detector.hpp" #include  "features/feature_tracks.hpp" #include  "fea-
    tures/image_feature_lk_tracker.hpp" #include  "features/image_pyramid.hpp" #include  "features/keypoint_resam-
    pler.hpp" #include  "features/mean_shift_tracker.hpp" #include  "features/redetection_strategy.hpp" #include  "fea-
    tures/tracker_base.hpp" #include  "imgproc/imgproc.hpp" #include  "imu/rotation-integrator.hpp" #include  "io/cam-
```

```
era/davis.hpp"#include "io/camera/discovery.hpp"#include "io/camera/dvs128.hpp"#include "io/camera/dvxp-  
plorer.hpp"#include "io/camera/dvxpexplorer_m.hpp"#include "io/data_read_handler.hpp"#include "io/mono_cam-  
era_recording.hpp"#include "io/mono_camera_writer.hpp"#include "io/network/tcp_tls_socket.hpp"#include  
"io/network_reader.hpp"#include "io/network_writer.hpp"#include "io/read_only_file.hpp"#include  
"io/reader.hpp"#include "io/simplefile.hpp"#include "io/stereo_camera_recording.hpp"#include "io/stereo_cam-  
era_writer.hpp"#include "io/write_only_file.hpp"#include "io/writer.hpp"#include "kinematics/linear_trans-  
former.hpp"#include "kinematics/motion_compensator.hpp"#include "kinematics/pixel_motion_predic-  
tor.hpp"#include "kinematics/transformation.hpp"#include "measurements/depth.hpp"#include "noise/back-  
ground_activity_noise_filter.hpp"#include "noise/fast_decay_noise_filter.hpp"#include "noise/frequency_fil-  
ters.hpp"#include "noise/k_noise_filter.hpp"#include "optimization/contrast_maximization_rotation.hpp"#include  
"optimization/contrast_maximization_translation_and_depth.hpp"#include "optimization/contrast_maximiza-  
tion_wrapper.hpp"#include "version.hpp"#include "visualization/colors.hpp"#include "visualization/event_visual-  
izer.hpp"#include "visualization/pose_visualizer.hpp"
```

file **version.hpp**

```
#include <string_view>
```

Defines

DV_PROCESSING_VERSION_MAJOR

dv-processing version (MAJOR * 10000 + MINOR * 100 + PATCH).

DV_PROCESSING_VERSION_MINOR

DV_PROCESSING_VERSION_PATCH

DV_PROCESSING_VERSION

DV_PROCESSING_NAME_STRING

dv-processing name string.

DV_PROCESSING_VERSION_STRING

dv-processing version string.

file **colors.hpp**

```
#include <opencv2/core.hpp>
```

file **event_visualizer.hpp**

```
#include "core/core.hpp"#include "core/Utils.hpp"#include "exception/exceptions/generic_excep-  
tions.hpp"#include "colors.hpp"
```

file **pose_visualizer.hpp**

```
#include "camera/camera_geometry.hpp"#include "data/frame_base.hpp"#include "data/land-  
mark_base.hpp"#include "data/utilities.hpp"#include "exception/exception.hpp"#include "kinemat-  
ics/linear_transformer.hpp"#include "visualization/colors.hpp"#include <opencv2/imgproc.hpp>#include  
<iostream>#include <map>
```

dir /builds/inivation/dv/dv-processing/include/dv-processing/camera/calibrations

dir /builds/inivation/dv/dv-processing/include/dv-processing/camera

dir /builds/inivation/dv/dv-processing/include/dv-processing/io/camera

dir /builds/inivation/dv/dv-processing/include/dv-processing/cluster

dir /builds/inivation/dv/dv-processing/include/dv-processing/io/compression

dir /builds/inivation/dv/dv-processing/include/dv-processing/containers

dir /builds/inivation/dv/dv-processing/include/dv-processing/core

dir /builds/inivation/dv/dv-processing/include/dv-processing/data

dir /builds/inivation/dv/dv-processing/include/dv-processing/io/data

dir /builds/inivation/dv/dv-processing/include/dv-processing/depth

dir /builds/inivation/dv/dv-processing/include/dv-processing

dir /builds/inivation/dv/dv-processing/include/dv-processing/exception

dir /builds/inivation/dv/dv-processing/include/dv-processing/exception/exceptions

dir /builds/inivation/dv/dv-processing/include/dv-processing/features

dir /builds/inivation/dv/dv-processing/include/dv-processing/core/frame

dir /builds/inivation/dv/dv-processing/include/dv-processing/imgproc

dir /builds/inivation/dv/dv-processing/include/dv-processing/imu

dir /builds/inivation/dv/dv-processing/include

dir /builds/inivation/dv/dv-processing/include/dv-processing/io

dir /builds/inivation/dv/dv-processing/include/dv-processing/containers/kd_tree

dir /builds/inivation/dv/dv-processing/include/dv-processing/kinematics

dir /builds/inivation/dv/dv-processing/include/dv-processing/cluster/mean_shift

dir /builds/inivation/dv/dv-processing/include/dv-processing/measurements

dir /builds/inivation/dv/dv-processing/include/dv-processing/io/network

dir /builds/inivation/dv/dv-processing/include/dv-processing/noise

dir /builds/inivation/dv/dv-processing/include/dv-processing/optimization

dir /builds/inivation/dv/dv-processing/include/dv-processing/io/camera/parsers

dir /builds/inivation/dv/dv-processing/include/dv-processing/io/support

dir /builds/inivation/dv/dv-processing/include/dv-processing/visualization

HELP

In case of technical issues or any problems, please visit the [support page](https://docs.inivation.com/help/support.html)²⁷. Any issues in the documentation or the code can be reported to our [gitlab issue tracker](https://gitlab.com/inivation/dv/dv-processing/-/issues)²⁸.

²⁷ <https://docs.inivation.com/help/support.html>

²⁸ <https://gitlab.com/inivation/dv/dv-processing/-/issues>

D

- `dv (C++ type)`, 577, 600
- `dv::Accumulator (C++ class)`, 179
- `dv::Accumulator::accumulate (C++ function)`, 180
- `dv::Accumulator::Accumulator (C++ function)`, 180
- `dv::Accumulator::clear (C++ function)`, 181
- `dv::Accumulator::contribute (C++ function)`, 182
- `dv::Accumulator::Decay (C++ enum)`, 179
- `dv::Accumulator::decay (C++ function)`, 182
- `dv::Accumulator::Decay::EXPONENTIAL (C++ enumerator)`, 179
- `dv::Accumulator::Decay::LINEAR (C++ enumerator)`, 179
- `dv::Accumulator::Decay::NONE (C++ enumerator)`, 179
- `dv::Accumulator::Decay::STEP (C++ enumerator)`, 180
- `dv::Accumulator::decayFunction_ (C++ member)`, 183
- `dv::Accumulator::decayParam_ (C++ member)`, 183
- `dv::Accumulator::decayTimeSurface_ (C++ member)`, 183
- `dv::Accumulator::eventContribution_ (C++ member)`, 182
- `dv::Accumulator::generateFrame (C++ function)`, 180
- `dv::Accumulator::getDecayFunction (C++ function)`, 182
- `dv::Accumulator::getDecayParam (C++ function)`, 182
- `dv::Accumulator::getEventContribution (C++ function)`, 181
- `dv::Accumulator::getMaxPotential (C++ function)`, 181
- `dv::Accumulator::getMinPotential (C++ function)`, 182
- `dv::Accumulator::getNeutralPotential (C++ function)`, 182
- `dv::Accumulator::getPotentialSurface (C++ function)`, 182
- `dv::Accumulator::highestTime_ (C++ member)`, 183
- `dv::Accumulator::isIgnorePolarity (C++ function)`, 181
- `dv::Accumulator::lowestTime_ (C++ member)`, 183
- `dv::Accumulator::maxPotential_ (C++ member)`, 182
- `dv::Accumulator::minPotential_ (C++ member)`, 183
- `dv::Accumulator::neutralPotential_ (C++ member)`, 182
- `dv::Accumulator::operator<< (C++ function)`, 182, 183
- `dv::Accumulator::potentialSurface_ (C++ member)`, 183
- `dv::Accumulator::rectifyPolarity_ (C++ member)`, 182
- `dv::Accumulator::resetTimestamp (C++ member)`, 183
- `dv::Accumulator::setDecayFunction (C++ function)`, 181
- `dv::Accumulator::setDecayParam (C++ function)`, 181
- `dv::Accumulator::setEventContribution (C++ function)`, 181
- `dv::Accumulator::setIgnorePolarity (C++ function)`, 181
- `dv::Accumulator::setMaxPotential (C++ function)`, 181
- `dv::Accumulator::setMinPotential (C++ function)`, 181
- `dv::Accumulator::setNeutralPotential (C++ function)`, 181
- `dv::Accumulator::setSynchronousDecay (C++ function)`, 181
- `dv::Accumulator::synchronousDecay_ (C++ member)`, 183
- `dv::AccumulatorBase (C++ class)`, 183
- `dv::AccumulatorBase::~~AccumulatorBase (C++ function)`, 184
- `dv::AccumulatorBase::accept (C++ function)`, 184

dv::AccumulatorBase::accumulate (C++ *function*), 183

dv::AccumulatorBase::AccumulatorBase (C++ *function*), 183

dv::AccumulatorBase::generateFrame (C++ *function*), 184

dv::AccumulatorBase::getResolution (C++ *function*), 184

dv::AccumulatorBase::mResolution (C++ *member*), 184

dv::AccumulatorBase::operator>> (C++ *function*), 184

dv::AccumulatorBase::SharedPtr (C++ *type*), 183

dv::AccumulatorBase::UniquePtr (C++ *type*), 183

dv::AddressableEventStorage (C++ *class*), 184

dv::AddressableEventStorage::_getLastNonFullPartial (C++ *function*), 192

dv::AddressableEventStorage::add (C++ *function*), 185

dv::AddressableEventStorage::AddressableEventStorage (C++ *function*), 185, 186, 192

dv::AddressableEventStorage::at (C++ *function*), 190

dv::AddressableEventStorage::back (C++ *function*), 189

dv::AddressableEventStorage::begin (C++ *function*), 189

dv::AddressableEventStorage::const_iterator (C++ *type*), 185

dv::AddressableEventStorage::const_packet_type (C++ *type*), 185

dv::AddressableEventStorage::const_pointer (C++ *type*), 185

dv::AddressableEventStorage::const_reference (C++ *type*), 185

dv::AddressableEventStorage::const_value_type (C++ *type*), 184

dv::AddressableEventStorage::coordinates (C++ *function*), 185

dv::AddressableEventStorage::dataPartials_ (C++ *member*), 192

dv::AddressableEventStorage::difference_type (C++ *type*), 185

dv::AddressableEventStorage::duration (C++ *function*), 191

dv::AddressableEventStorage::eigen (C++ *function*), 186

dv::AddressableEventStorage::emplace_back (C++ *function*), 186

dv::AddressableEventStorage::end (C++ *function*), 189

dv::AddressableEventStorage::erase (C++ *function*), 190

dv::AddressableEventStorage::eraseTime (C++ *function*), 190

dv::AddressableEventStorage::front (C++ *function*), 189

dv::AddressableEventStorage::getHighestTime (C++ *function*), 190

dv::AddressableEventStorage::getLowestTime (C++ *function*), 189

dv::AddressableEventStorage::getShardCapacity (C++ *function*), 191

dv::AddressableEventStorage::getShardCount (C++ *function*), 191

dv::AddressableEventStorage::isEmpty (C++ *function*), 190

dv::AddressableEventStorage::isWithinStoreTimeRange (C++ *function*), 191

dv::AddressableEventStorage::iterator (C++ *type*), 185

dv::AddressableEventStorage::operator+ (C++ *function*), 186

dv::AddressableEventStorage::operator+= (C++ *function*), 187

dv::AddressableEventStorage::operator<< (C++ *function*), 187, 192

dv::AddressableEventStorage::operator= (C++ *function*), 186

dv::AddressableEventStorage::operator[] (C++ *function*), 190

dv::AddressableEventStorage::packet_type (C++ *type*), 185

dv::AddressableEventStorage::PartialEventDataTypes (C++ *type*), 192

dv::AddressableEventStorage::partialOffsets_ (C++ *member*), 192

dv::AddressableEventStorage::pointer (C++ *type*), 184

dv::AddressableEventStorage::polarities (C++ *function*), 185

dv::AddressableEventStorage::push_back (C++ *function*), 186

dv::AddressableEventStorage::rate (C++ *function*), 191

dv::AddressableEventStorage::reference (C++ *type*), 185

dv::AddressableEventStorage::retainDuration (C++ *function*), 191

dv::AddressableEventStorage::setShardCapacity (C++ *function*), 191

dv::AddressableEventStorage::shardCapacity_ (C++ *member*), 192

dv::AddressableEventStorage::size (C++ *function*), 187

dv::AddressableEventStorage::size_type (C++ type), 185
 dv::AddressableEventStorage::slice (C++ function), 187
 dv::AddressableEventStorage::sliceBack (C++ function), 188
 dv::AddressableEventStorage::sliceRate (C++ function), 189
 dv::AddressableEventStorage::sliceTime (C++ function), 188, 189
 dv::AddressableEventStorage::timestamps (C++ function), 185
 dv::AddressableEventStorage::toPacket (C++ function), 191
 dv::AddressableEventStorage::totalLength_ (C++ member), 192
 dv::AddressableEventStorage::value_type (C++ type), 184
 dv::AddressableEventStorageIterator (C++ class), 192
 dv::AddressableEventStorageIterator::AddressableEventStorageIterator (C++ function), 193
 dv::AddressableEventStorageIterator::dataPartialsPtr_ (C++ member), 195
 dv::AddressableEventStorageIterator::decrement (C++ function), 194
 dv::AddressableEventStorageIterator::difference_type (C++ type), 193
 dv::AddressableEventStorageIterator::increment (C++ function), 194
 dv::AddressableEventStorageIterator::iterator_category (C++ type), 192
 dv::AddressableEventStorageIterator::offset_ (C++ member), 195
 dv::AddressableEventStorageIterator::operator-- (C++ function), 194
 dv::AddressableEventStorageIterator::operator* (C++ function), 193
 dv::AddressableEventStorageIterator::operator++ (C++ function), 193
 dv::AddressableEventStorageIterator::operator+= (C++ function), 194
 dv::AddressableEventStorageIterator::operator!= (C++ function), 194
 dv::AddressableEventStorageIterator::operator-= (C++ function), 194
 dv::AddressableEventStorageIterator::operator== (C++ function), 194
 dv::AddressableEventStorageIterator::operator-> (C++ function), 193
 dv::AddressableEventStorageIterator::partialIndex_ (C++ member), 195
 dv::AddressableEventStorageIterator::pointer (C++ type), 193
 dv::AddressableEventStorageIterator::reference (C++ type), 193
 dv::AddressableEventStorageIterator::size_type (C++ type), 193
 dv::AddressableEventStorageIterator::value_type (C++ type), 192
 dv::AddressableStereoEventStreamSlicer (C++ class), 195
 dv::AddressableStereoEventStreamSlicer::accept (C++ function), 195
 dv::AddressableStereoEventStreamSlicer::clearRightEventsBuffer (C++ function), 196
 dv::AddressableStereoEventStreamSlicer::doEveryNumberOfEvents (C++ function), 195
 dv::AddressableStereoEventStreamSlicer::doEveryTimeInterval (C++ function), 195
 dv::AddressableStereoEventStreamSlicer::hasJob (C++ function), 196
 dv::AddressableStereoEventStreamSlicer::leftEvents (C++ member), 196
 dv::AddressableStereoEventStreamSlicer::minimumEvents (C++ member), 196
 dv::AddressableStereoEventStreamSlicer::minimumTime (C++ member), 196
 dv::AddressableStereoEventStreamSlicer::removeJob (C++ function), 196
 dv::AddressableStereoEventStreamSlicer::rightEvents (C++ member), 196
 dv::AddressableStereoEventStreamSlicer::rightEventSeek (C++ member), 196
 dv::AddressableStereoEventStreamSlicer::slicer (C++ member), 196
 dv::BoundingBox (C++ struct), 203
 dv::BoundingBox::bottomRightX (C++ member), 204
 dv::BoundingBox::bottomRightY (C++ member), 204
 dv::BoundingBox::BoundingBox (C++ function), 204
 dv::BoundingBox::confidence (C++ member), 204
 dv::BoundingBox::GetFullyQualified_name (C++ function), 204
 dv::BoundingBox::label (C++ member), 204
 dv::BoundingBox::TableType (C++ type), 204

`dv::BoundingBox::timestamp` (C++ member), 204
`dv::BoundingBox::topLeftX` (C++ member), 204
`dv::BoundingBox::topLeftY` (C++ member), 204
`dv::BoundingBoxBuilder` (C++ struct), 204
`dv::BoundingBoxBuilder::add_bottomRightX` (C++ function), 204
`dv::BoundingBoxBuilder::add_bottomRightY` (C++ function), 204
`dv::BoundingBoxBuilder::add_confidence` (C++ function), 204
`dv::BoundingBoxBuilder::add_label` (C++ function), 204
`dv::BoundingBoxBuilder::add_timestamp` (C++ function), 204
`dv::BoundingBoxBuilder::add_topLeftX` (C++ function), 204
`dv::BoundingBoxBuilder::add_topLeftY` (C++ function), 204
`dv::BoundingBoxBuilder::BoundingBoxBuilder` (C++ function), 204
`dv::BoundingBoxBuilder::fbb_` (C++ member), 205
`dv::BoundingBoxBuilder::Finish` (C++ function), 205
`dv::BoundingBoxBuilder::operator=` (C++ function), 204
`dv::BoundingBoxBuilder::start_` (C++ member), 205
`dv::BoundingBoxFlatbuffer` (C++ struct), 205
`dv::BoundingBoxFlatbuffer::bottomRightX` (C++ function), 205
`dv::BoundingBoxFlatbuffer::bottomRightY` (C++ function), 205
`dv::BoundingBoxFlatbuffer::confidence` (C++ function), 205
`dv::BoundingBoxFlatbuffer::GetFullyQualifiedName` (C++ function), 205
`dv::BoundingBoxFlatbuffer::label` (C++ function), 205
`dv::BoundingBoxFlatbuffer::MiniReflectTypeTable` (C++ function), 205
`dv::BoundingBoxFlatbuffer::NativeTableType` (C++ type), 205
`dv::BoundingBoxFlatbuffer::Pack` (C++ function), 205
`dv::BoundingBoxFlatbuffer::timestamp` (C++ function), 205
`dv::BoundingBoxFlatbuffer::topLeftX` (C++ function), 205
`dv::BoundingBoxFlatbuffer::topLeftY` (C++ function), 205
`dv::BoundingBoxFlatbuffer::UnPack` (C++ function), 205
`dv::BoundingBoxFlatbuffer::UnPackTo` (C++ function), 205
`dv::BoundingBoxFlatbuffer::UnPackToFrom` (C++ function), 205
`dv::BoundingBoxFlatbuffer::Verify` (C++ function), 205
`dv::BoundingBoxPacket` (C++ struct), 206
`dv::BoundingBoxPacket::BoundingBoxPacket` (C++ function), 206
`dv::BoundingBoxPacket::elements` (C++ member), 206
`dv::BoundingBoxPacket::GetFullyQualifiedName` (C++ function), 206
`dv::BoundingBoxPacket::operator<<` (C++ function), 206
`dv::BoundingBoxPacket::TableType` (C++ type), 206
`dv::BoundingBoxPacketBufferHasIdentifier` (C++ function), 587
`dv::BoundingBoxPacketBuilder` (C++ struct), 206
`dv::BoundingBoxPacketBuilder::add_elements` (C++ function), 206
`dv::BoundingBoxPacketBuilder::BoundingBoxPacketBuilder` (C++ function), 206
`dv::BoundingBoxPacketBuilder::fbb_` (C++ member), 206
`dv::BoundingBoxPacketBuilder::Finish` (C++ function), 206
`dv::BoundingBoxPacketBuilder::operator=` (C++ function), 206
`dv::BoundingBoxPacketBuilder::start_` (C++ member), 206
`dv::BoundingBoxPacketFlatbuffer` (C++ struct), 206
`dv::BoundingBoxPacketFlatbuffer::elements` (C++ function), 207
`dv::BoundingBoxPacketFlatbuffer::GetFullyQualifiedName` (C++ function), 207
`dv::BoundingBoxPacketFlatbuffer::identifier` (C++ member), 207
`dv::BoundingBoxPacketFlatbuffer::MiniReflectTypeTable` (C++ function), 207
`dv::BoundingBoxPacketFlatbuffer::NativeTableType` (C++ type), 207
`dv::BoundingBoxPacketFlatbuffer::Pack` (C++ function), 207
`dv::BoundingBoxPacketFlatbuffer::UnPack` (C++ function), 207
`dv::BoundingBoxPacketFlatbuffer::UnPackTo` (C++ function), 207
`dv::BoundingBoxPacketFlatbuffer::UnPackToFrom` (C++ function), 207
`dv::BoundingBoxPacketFlatbuffer::Verify` (C++ function), 207
`dv::BoundingBoxPacketIdentifier` (C++ function), 205

- tion*), 587
- dv::BoundingBoxPacketTypeTable (C++ *function*), 586
- dv::BoundingBoxTypeTable (C++ *function*), 586
- dv::boundingRect (C++ *function*), 584
- dv::camera (C++ *type*), 600
- dv::camera::calibrations (C++ *type*), 601
- dv::camera::calibrations::CameraCalibration (C++ *struct*), 212
- dv::camera::calibrations::CameraCalibration::CameraCalibration (C++ *function*), 212
- dv::camera::calibrations::CameraCalibration::distortion (C++ *member*), 213
- dv::camera::calibrations::CameraCalibration::distortionModel (C++ *member*), 213
- dv::camera::calibrations::CameraCalibration::focalLength (C++ *member*), 213
- dv::camera::calibrations::CameraCalibration::getCameraGeometry (C++ *function*), 213
- dv::camera::calibrations::CameraCalibration::getCameraMatrix (C++ *function*), 213
- dv::camera::calibrations::CameraCalibration::getDistortionModelString (C++ *function*), 213
- dv::camera::calibrations::CameraCalibration::getOptionalMetadata (C++ *function*), 215
- dv::camera::calibrations::CameraCalibration::getTransformFromTree (C++ *function*), 214
- dv::camera::calibrations::CameraCalibration::getVectorFromTree (C++ *function*), 214
- dv::camera::calibrations::CameraCalibration::master (C++ *member*), 213
- dv::camera::calibrations::CameraCalibration::metadata (C++ *member*), 214
- dv::camera::calibrations::CameraCalibration::Metadata (C++ *struct*), 414
- dv::camera::calibrations::CameraCalibration::Metadata::calibrationError (C++ *member*), 415
- dv::camera::calibrations::CameraCalibration::Metadata::calibrationTime (C++ *member*), 415
- dv::camera::calibrations::CameraCalibration::Metadata::comment (C++ *member*), 415
- dv::camera::calibrations::CameraCalibration::Metadata::internalPattern-
- Shape (C++ *member*), 415
- dv::camera::calibrations::CameraCalibration::Metadata::Metadata (C++ *function*), 414
- dv::camera::calibrations::CameraCalibration::Metadata::operator== (C++ *function*), 415
- dv::camera::calibrations::CameraCalibration::Metadata::patternShape (C++ *member*), 415
- dv::camera::calibrations::CameraCalibration::Metadata::patternSize (C++ *member*), 415
- dv::camera::calibrations::CameraCalibration::Metadata::patternSpacing (C++ *member*), 415
- dv::camera::calibrations::CameraCalibration::Metadata::patternType (C++ *member*), 415
- dv::camera::calibrations::CameraCalibration::Metadata::pixelPitch (C++ *member*), 415
- dv::camera::calibrations::CameraCalibration::Metadata::quality (C++ *member*), 415
- dv::camera::calibrations::CameraCalibration::Metadata::toPropertyTree (C++ *function*), 414
- dv::camera::calibrations::CameraCalibration::name (C++ *member*), 213
- dv::camera::calibrations::CameraCalibration::operator<< (C++ *function*), 215
- dv::camera::calibrations::CameraCalibration::operator== (C++ *function*), 212
- dv::camera::calibrations::CameraCalibration::parsePair (C++ *function*), 215
- dv::camera::calibrations::CameraCalibration::parseTripple (C++ *function*), 215
- dv::camera::calibrations::CameraCalibration::position (C++ *member*), 213
- dv::camera::calibrations::CameraCalibration::principalPoint (C++ *member*), 213
- dv::camera::calibrations::CameraCalibration::pushTransformToTree (C++ *function*), 214
- dv::camera::calibrations::CameraCalibration::pushVectorToTree (C++ *function*), 214
- dv::camera::calibrations::CameraCalibration::resolution (C++ *member*), 213
- dv::camera::calibrations::CameraCalibration::toPropertyTree (C++ *function*),

- 212
- dv::camera::calibrations::CameraCalibration::transformationToC0 (C++ member), 214
- dv::camera::calibrations::IMUCalibration (C++ struct), 368
- dv::camera::calibrations::IMUCalibration::accMax (C++ member), 369
- dv::camera::calibrations::IMUCalibration::accNoiseDensity (C++ member), 369
- dv::camera::calibrations::IMUCalibration::accNoiseRandomWalk (C++ member), 369
- dv::camera::calibrations::IMUCalibration::accOffsetAvg (C++ member), 369
- dv::camera::calibrations::IMUCalibration::accOffsetVar (C++ member), 369
- dv::camera::calibrations::IMUCalibration::IMUCalibration (C++ function), 368
- dv::camera::calibrations::IMUCalibration::metadata (C++ member), 369
- dv::camera::calibrations::IMUCalibration::Metadata (C++ struct), 415
- dv::camera::calibrations::IMUCalibration::Metadata::calibrationTime (C++ member), 416
- dv::camera::calibrations::IMUCalibration::Metadata::comment (C++ member), 416
- dv::camera::calibrations::IMUCalibration::Metadata::Metadata (C++ function), 416
- dv::camera::calibrations::IMUCalibration::Metadata::operator== (C++ function), 416
- dv::camera::calibrations::IMUCalibration::Metadata::toPropertyTree (C++ function), 416
- dv::camera::calibrations::IMUCalibration::name (C++ member), 369
- dv::camera::calibrations::IMUCalibration::omegaMax (C++ member), 369
- dv::camera::calibrations::IMUCalibration::omegaNoiseDensity (C++ member), 369
- dv::camera::calibrations::IMUCalibration::omegaNoiseRandomWalk (C++ member), 369
- dv::camera::calibrations::IMUCalibration::omegaOffsetAvg (C++ member), 369
- dv::camera::calibrations::IMUCalibration::omegaOffsetVar (C++ member), 369
- dv::camera::calibrations::IMUCalibration::operator<< (C++ function), 370
- dv::camera::calibrations::IMUCalibration::operator== (C++ function), 368
- dv::camera::calibrations::IMUCalibration::timeOffsetMicros (C++ member), 369
- dv::camera::calibrations::IMUCalibration::toPropertyTree (C++ function), 368
- dv::camera::calibrations::IMUCalibration::transformationToC0 (C++ member), 369
- dv::camera::calibrations::StereoCalibration (C++ struct), 516
- dv::camera::calibrations::StereoCalibration::essentialMatrix (C++ member), 517
- dv::camera::calibrations::StereoCalibration::fundamentalMatrix (C++ member), 517
- dv::camera::calibrations::StereoCalibration::getEssentialMatrix (C++ function), 517
- dv::camera::calibrations::StereoCalibration::getFundamentalMatrix (C++ function), 517
- dv::camera::calibrations::StereoCalibration::leftCameraName (C++ member), 517
- dv::camera::calibrations::StereoCalibration::metadata (C++ member), 517
- dv::camera::calibrations::StereoCalibration::Metadata (C++ struct), 416
- dv::camera::calibrations::StereoCalibration::Metadata::comment (C++ member), 416
- dv::camera::calibrations::StereoCalibration::Metadata::epipolarError (C++ member), 416
- dv::camera::calibrations::StereoCalibration::Metadata::Metadata (C++ function), 416
- dv::camera::calibrations::StereoCalibration::Metadata::operator== (C++ function), 416
- dv::camera::calibrations::StereoCalibration::Metadata::toPropertyTree (C++ function), 416
- dv::camera::calibrations::StereoCalibration::

tion::operator<< (C++ function), 517
 dv::camera::calibrations::StereoCalibration::operator== (C++ function), 517
 dv::camera::calibrations::StereoCalibration::rightCameraName (C++ member), 517
 dv::camera::calibrations::StereoCalibration::StereoCalibration (C++ function), 517
 dv::camera::calibrations::StereoCalibration::toPropertyTree (C++ function), 517
 dv::camera::CalibrationSet (C++ class), 207
 dv::camera::CalibrationSet::addCameraCalibration (C++ function), 210
 dv::camera::CalibrationSet::addImuCalibration (C++ function), 210
 dv::camera::CalibrationSet::addStereoCalibration (C++ function), 210
 dv::camera::CalibrationSet::CalibrationSet (C++ function), 208, 211
 dv::camera::CalibrationSet::CameraCalibrationMap (C++ type), 207
 dv::camera::CalibrationSet::cameraIndex (C++ member), 211
 dv::camera::CalibrationSet::cameraRigCalibrationFromJsonFile (C++ function), 212
 dv::camera::CalibrationSet::cameraRigCalibrationFromXmlFile (C++ function), 212
 dv::camera::CalibrationSet::cameras (C++ member), 211
 dv::camera::CalibrationSet::getCameraCalibration (C++ function), 208
 dv::camera::CalibrationSet::getCameraCalibrationByName (C++ function), 209
 dv::camera::CalibrationSet::getCameraCalibrations (C++ function), 210
 dv::camera::CalibrationSet::getCameraList (C++ function), 208
 dv::camera::CalibrationSet::getImuCalibration (C++ function), 208
 dv::camera::CalibrationSet::getImuCalibrationByName (C++ function), 209
 dv::camera::CalibrationSet::getImuCalibrations (C++ function), 211
 dv::camera::CalibrationSet::getImuList (C++ function), 208
 dv::camera::CalibrationSet::getStereoCalibration (C++ function), 208
 dv::camera::CalibrationSet::getStereoCalibrationByLeftCameraName (C++ function), 209
 dv::camera::CalibrationSet::getStereoCalibrationByRightCameraName (C++ function), 209
 dv::camera::CalibrationSet::getStereoCalibrations (C++ function), 211
 dv::camera::CalibrationSet::getStereoList (C++ function), 208
 dv::camera::CalibrationSet::IMUCalibrationMap (C++ type), 207
 dv::camera::CalibrationSet::imuIndex (C++ member), 211
 dv::camera::CalibrationSet::imus (C++ member), 211
 dv::camera::CalibrationSet::LoadFromFile (C++ function), 211
 dv::camera::CalibrationSet::oneCameraCalibrationFromXML (C++ function), 212
 dv::camera::CalibrationSet::stereo (C++ member), 211
 dv::camera::CalibrationSet::StereoCalibrationMap (C++ type), 207
 dv::camera::CalibrationSet::toPropertyTree (C++ function), 208
 dv::camera::CalibrationSet::updateCameraCalibration (C++ function), 210
 dv::camera::CalibrationSet::updateImuCalibration (C++ function), 210
 dv::camera::CalibrationSet::updateStereoCameraCalibration (C++ function), 210
 dv::camera::CalibrationSet::writeToFile (C++ function), 211
 dv::camera::CameraGeometry (C++ class), 215
 dv::camera::CameraGeometry::backProject (C++ function), 217
 dv::camera::CameraGeometry::backProjectSequence (C++ function), 217
 dv::camera::CameraGeometry::backProjectUndistort (C++ function), 217
 dv::camera::CameraGeometry::backProjectUndistortSequence (C++ function), 218
 dv::camera::CameraGeometry::CameraGeometry (C++ function), 216
 dv::camera::CameraGeometry::distort (C++ function), 217
 dv::camera::CameraGeometry::distortEquidistant (C++ function), 219
 dv::camera::CameraGeometry::distortRadialTangential (C++ function), 219
 dv::camera::CameraGeometry::distortSequence (C++ function), 217
 dv::camera::CameraGeometry::FunctionImplementation (C++ enum), 215

dv::camera::CameraGeometry::FunctionImplementation::LUT (C++ *enumerator*), 215
 dv::camera::CameraGeometry::FunctionImplementation::SUB_PIXEL (C++ *enumerator*), 215
 dv::camera::CameraGeometry::generateLUTs (C++ *function*), 219
 dv::camera::CameraGeometry::getCameraMatrix (C++ *function*), 219
 dv::camera::CameraGeometry::getCentralPoint (C++ *function*), 219
 dv::camera::CameraGeometry::getDistortion (C++ *function*), 219
 dv::camera::CameraGeometry::getDistortionModel (C++ *function*), 219
 dv::camera::CameraGeometry::getFocalLength (C++ *function*), 219
 dv::camera::CameraGeometry::getResolution (C++ *function*), 219
 dv::camera::CameraGeometry::isUndistortionAvailable (C++ *function*), 218
 dv::camera::CameraGeometry::isWithinDimensions (C++ *function*), 218
 dv::camera::CameraGeometry::mBackProjectLUT (C++ *member*), 220
 dv::camera::CameraGeometry::mCx (C++ *member*), 220
 dv::camera::CameraGeometry::mCy (C++ *member*), 220
 dv::camera::CameraGeometry::mDistortion (C++ *member*), 220
 dv::camera::CameraGeometry::mDistortionLUT (C++ *member*), 220
 dv::camera::CameraGeometry::mDistortionModel (C++ *member*), 221
 dv::camera::CameraGeometry::mDistortionPixelLUT (C++ *member*), 220
 dv::camera::CameraGeometry::mFx (C++ *member*), 220
 dv::camera::CameraGeometry::mFy (C++ *member*), 220
 dv::camera::CameraGeometry::mMaxX (C++ *member*), 220
 dv::camera::CameraGeometry::mMaxY (C++ *member*), 220
 dv::camera::CameraGeometry::mResolution (C++ *member*), 220
 dv::camera::CameraGeometry::operator<< (C++ *function*), 221
 dv::camera::CameraGeometry::project (C++ *function*), 218
 dv::camera::CameraGeometry::projectSequence (C++ *function*), 218
 dv::camera::CameraGeometry::SharedPtr (C++ *type*), 215
 dv::camera::CameraGeometry::undistort (C++ *function*), 216
 dv::camera::CameraGeometry::undistortEvents (C++ *function*), 216
 dv::camera::CameraGeometry::undistortSequence (C++ *function*), 216
 dv::camera::CameraGeometry::UniquePtr (C++ *type*), 215
 dv::camera::DistortionModel (C++ *enum*), 600
 dv::camera::DistortionModel::EQUIDISTANT (C++ *enumerator*), 601
 dv::camera::DistortionModel::NONE (C++ *enumerator*), 600
 dv::camera::DistortionModel::RADIAL_TANGENTIAL (C++ *enumerator*), 600
 dv::camera::distortionModelToString (C++ *function*), 601
 dv::camera::internal (C++ *type*), 601
 dv::camera::internal::EQUIDISTANT_MODEL_STRING (C++ *member*), 601
 dv::camera::internal::NONE_MODEL_STRING (C++ *member*), 601
 dv::camera::internal::RADIAL_TANGENTIAL_MODEL_STRING (C++ *member*), 601
 dv::camera::operator<< (C++ *function*), 601
 dv::camera::StereoGeometry (C++ *class*), 520
 dv::camera::StereoGeometry::backProject (C++ *function*), 523
 dv::camera::StereoGeometry::CameraPosition (C++ *enum*), 520
 dv::camera::StereoGeometry::CameraPosition::LEFT (C++ *enumerator*), 520
 dv::camera::StereoGeometry::CameraPosition::RIGHT (C++ *enumerator*), 520
 dv::camera::StereoGeometry::computeTransformBetween (C++ *function*), 523
 dv::camera::StereoGeometry::convertDisparityToDepth (C++ *function*), 522
 dv::camera::StereoGeometry::createLUTs (C++ *function*), 523
 dv::camera::StereoGeometry::estimateDepth (C++ *function*), 522
 dv::camera::StereoGeometry::FunctionImplementation (C++ *enum*), 520
 dv::camera::StereoGeometry::FunctionImplementation::LUT (C++ *enumerator*), 520
 dv::camera::StereoGeometry::FunctionImplementation::SUB_PIXEL (C++ *enumerator*), 520
 dv::camera::StereoGeometry::getLeftCameraGeometry (C++ *function*), 522

dv::camera::StereoGeometry::getRightCameraGeometry (C++ function), 522
 dv::camera::StereoGeometry::initCoordinateList (C++ function), 525
 dv::camera::StereoGeometry::mBaselineFocal (C++ member), 525
 dv::camera::StereoGeometry::mDistLeft (C++ member), 524
 dv::camera::StereoGeometry::mDistRight (C++ member), 524
 dv::camera::StereoGeometry::mLeftDistModel (C++ member), 524
 dv::camera::StereoGeometry::mLeftProjection (C++ member), 524
 dv::camera::StereoGeometry::mLeftRectifierInverse (C++ member), 524
 dv::camera::StereoGeometry::mLeftRemap1 (C++ member), 523
 dv::camera::StereoGeometry::mLeftRemap2 (C++ member), 523
 dv::camera::StereoGeometry::mLeftRemapLUT (C++ member), 524
 dv::camera::StereoGeometry::mLeftResolution (C++ member), 524
 dv::camera::StereoGeometry::mLeftUnmapLUT (C++ member), 524
 dv::camera::StereoGeometry::mLeftValidMask (C++ member), 524
 dv::camera::StereoGeometry::mOriginalLeft (C++ member), 524
 dv::camera::StereoGeometry::mOriginalRight (C++ member), 524
 dv::camera::StereoGeometry::mRightDistModel (C++ member), 524
 dv::camera::StereoGeometry::mRightProjection (C++ member), 524
 dv::camera::StereoGeometry::mRightRectifierInverse (C++ member), 524
 dv::camera::StereoGeometry::mRightRemap1 (C++ member), 523
 dv::camera::StereoGeometry::mRightRemap2 (C++ member), 524
 dv::camera::StereoGeometry::mRightRemapLUT (C++ member), 524
 dv::camera::StereoGeometry::mRightResolution (C++ member), 524
 dv::camera::StereoGeometry::mRightUnmapLUT (C++ member), 524
 dv::camera::StereoGeometry::mRightValidMask (C++ member), 524
 dv::camera::StereoGeometry::operator<< (C++ function), 525
 dv::camera::StereoGeometry::Q (C++ member), 524
 dv::camera::StereoGeometry::remapEvents (C++ function), 521
 dv::camera::StereoGeometry::remapEventsInternal (C++ function), 525
 dv::camera::StereoGeometry::remapImage (C++ function), 521
 dv::camera::StereoGeometry::remapPoint (C++ function), 521
 dv::camera::StereoGeometry::RN (C++ member), 524
 dv::camera::StereoGeometry::SharedPtr (C++ type), 520
 dv::camera::StereoGeometry::StereoGeometry (C++ function), 520, 521
 dv::camera::StereoGeometry::toDepthFrame (C++ function), 523
 dv::camera::StereoGeometry::UniquePtr (C++ type), 520
 dv::camera::StereoGeometry::unmapPoint (C++ function), 521
 dv::camera::stringToDistortionModel (C++ function), 601
 dv::cluster (C++ type), 601
 dv::cluster::mean_shift (C++ type), 601
 dv::cluster::mean_shift::kernel (C++ type), 602
 dv::cluster::mean_shift::kernel::Epanechnikov (C++ struct), 319
 dv::cluster::mean_shift::kernel::Epanechnikov::apply (C++ function), 319
 dv::cluster::mean_shift::kernel::Epanechnikov::getSearchRadius (C++ function), 319
 dv::cluster::mean_shift::kernel::Gaussian (C++ struct), 360
 dv::cluster::mean_shift::kernel::Gaussian::apply (C++ function), 360
 dv::cluster::mean_shift::kernel::Gaussian::getSearchRadius (C++ function), 360
 dv::cluster::mean_shift::kernel::MeanShiftKernel (C++ concept), 573
 dv::cluster::mean_shift::MeanShiftColMajorMatrix1X (C++ type), 602
 dv::cluster::mean_shift::MeanShiftColMajorMatrix2X (C++ type), 602
 dv::cluster::mean_shift::MeanShiftColMajorMatrix3X (C++ type), 602
 dv::cluster::mean_shift::MeanShiftColMajorMatrix4X (C++ type), 602
 dv::cluster::mean_shift::MeanShiftColMajorMatrixXX (C++ type), 601
 dv::cluster::mean_shift::MeanShiftEigenMatrixAdaptor (C++ class), 396

`dv::cluster::mean_shift::MeanShiftEigenMatrixAdaptor::~~MeanShiftEigenMatrixAdaptor (C++ function), 400`

`dv::cluster::mean_shift::MeanShiftEigenMatrixAdaptor::applyKernel (C++ function), 402`

`dv::cluster::mean_shift::MeanShiftEigenMatrixAdaptor::assignClusters (C++ function), 401`

`dv::cluster::mean_shift::MeanShiftEigenMatrixAdaptor::DIMS (C++ member), 404`

`dv::cluster::mean_shift::MeanShiftEigenMatrixAdaptor::extractSample (C++ function), 403`

`dv::cluster::mean_shift::MeanShiftEigenMatrixAdaptor::findClusterCentres (C++ function), 401`

`dv::cluster::mean_shift::MeanShiftEigenMatrixAdaptor::fit (C++ function), 400`

`dv::cluster::mean_shift::MeanShiftEigenMatrixAdaptor::generateStartingPointsFromData (C++ function), 400`

`dv::cluster::mean_shift::MeanShiftEigenMatrixAdaptor::generateStartingPointsFromRange (C++ function), 400`

`dv::cluster::mean_shift::MeanShiftEigenMatrixAdaptor::getNeighbours (C++ function), 402`

`dv::cluster::mean_shift::MeanShiftEigenMatrixAdaptor::getSample (C++ function), 402`

`dv::cluster::mean_shift::MeanShiftEigenMatrixAdaptor::getZeroVector (C++ function), 403, 404`

`dv::cluster::mean_shift::MeanShiftEigenMatrixAdaptor::KDTree (C++ type), 401`

`dv::cluster::mean_shift::MeanShiftEigenMatrixAdaptor::Matrix (C++ type), 397`

`dv::cluster::mean_shift::MeanShiftEigenMatrixAdaptor::mBandwidth (C++ member), 403`

`dv::cluster::mean_shift::MeanShiftEigenMatrixAdaptor::mConvergence (C++ member), 403`

`dv::cluster::mean_shift::MeanShiftEigenMatrixAdaptor::mData (C++ member), 403`

`dv::cluster::mean_shift::MeanShiftEigenMatrixAdaptor::MeanShiftEigenMatrixAdaptor (C++ function), 398, 400`

`dv::cluster::mean_shift::MeanShiftEigenMatrixAdaptor::mMaxIter (C++ member), 403`

`dv::cluster::mean_shift::MeanShiftEigenMatrixAdaptor::mNumDimensions (C++ member), 403`

`dv::cluster::mean_shift::MeanShiftEigenMatrixAdaptor::mNumSamples (C++ member), 403`

`dv::cluster::mean_shift::MeanShiftEigenMatrixAdaptor::mStartingPoints (C++ member), 403`

`dv::cluster::mean_shift::MeanShiftEigenMatrixAdaptor::NOT_SAMPLE_ORDER (C++ member), 404`

`dv::cluster::mean_shift::MeanShiftEigenMatrixAdaptor::operator= (C++ function), 400`

`dv::cluster::mean_shift::MeanShiftEigenMatrixAdaptor::performShift (C++ function), 401`

`dv::cluster::mean_shift::MeanShiftEigenMatrixAdaptor::randomArrayBetween (C++ function), 403`

`dv::cluster::mean_shift::MeanShiftEigenMatrixAdaptor::STORAGE_ORDER (C++ member), 404`

`dv::cluster::mean_shift::MeanShiftEigenMatrixAdaptor::ThisType (C++ type), 401`

`dv::cluster::mean_shift::MeanShiftEigenMatrixAdaptor::Vector (C++ type), 397`

`dv::cluster::mean_shift::MeanShiftEigenMatrixAdaptor::VectorOfVectors (C++ type), 397`

`dv::cluster::mean_shift::MeanShiftEventStoreAdaptor (C++ class), 404`

`dv::cluster::mean_shift::MeanShiftEventStoreAdaptor::~~MeanShiftEventStoreAdaptor (C++ function), 406`

`dv::cluster::mean_shift::MeanShiftEventStoreAdaptor::applyKernel (C++ function), 408`

`dv::cluster::mean_shift::MeanShiftEventStoreAdaptor::assignClusters (C++ function), 407`

`dv::cluster::mean_shift::MeanShiftEventStoreAdaptor::findClusterCentres (C++ function), 406`

`dv::cluster::mean_shift::MeanShiftEventStoreAdaptor::fit (C++`

- function*), 406
- dv::cluster::mean_shift::Mean-ShiftEventStoreAdaptor::generateStartingPointsFromData (C++ *function*), 407
- dv::cluster::mean_shift::Mean-ShiftEventStoreAdaptor::generateStartingPointsFromRange (C++ *function*), 407
- dv::cluster::mean_shift::Mean-ShiftEventStoreAdaptor::getNeighbours (C++ *function*), 408
- dv::cluster::mean_shift::Mean-ShiftEventStoreAdaptor::getZeroVector (C++ *function*), 409
- dv::cluster::mean_shift::Mean-ShiftEventStoreAdaptor::KDTree (C++ *type*), 408
- dv::cluster::mean_shift::Mean-ShiftEventStoreAdaptor::mBandwidth (C++ *member*), 409
- dv::cluster::mean_shift::Mean-ShiftEventStoreAdaptor::mConvergence (C++ *member*), 409
- dv::cluster::mean_shift::Mean-ShiftEventStoreAdaptor::mData (C++ *member*), 409
- dv::cluster::mean_shift::Mean-ShiftEventStoreAdaptor::Mean-ShiftEventStoreAdaptor (C++ *function*), 404406
- dv::cluster::mean_shift::Mean-ShiftEventStoreAdaptor::mMaxIter (C++ *member*), 409
- dv::cluster::mean_shift::Mean-ShiftEventStoreAdaptor::mNumSamples (C++ *member*), 409
- dv::cluster::mean_shift::Mean-ShiftEventStoreAdaptor::mStartingPoints (C++ *member*), 409
- dv::cluster::mean_shift::Mean-ShiftEventStoreAdaptor::operator= (C++ *function*), 406
- dv::cluster::mean_shift::Mean-ShiftEventStoreAdaptor::performShift (C++ *function*), 408
- dv::cluster::mean_shift::Mean-ShiftEventStoreAdaptor::pow2 (C++ *function*), 409
- dv::cluster::mean_shift::Mean-ShiftEventStoreAdaptor::squaredDistance (C++ *function*), 409
- dv::cluster::mean_shift::Mean-ShiftEventStoreAdaptor::Vector (C++ *type*), 404
- dv::cluster::mean_shift::Mean-ShiftEventStoreAdaptor::VectorOfVectors (C++ *type*), 404
- dv::cluster::mean_shift::MeanShiftRowMajorMatrixX1 (C++ *type*), 602
- dv::cluster::mean_shift::MeanShiftRowMajorMatrixX2 (C++ *type*), 602
- dv::cluster::mean_shift::MeanShiftRowMajorMatrixX3 (C++ *type*), 602
- dv::cluster::mean_shift::MeanShiftRowMajorMatrixX4 (C++ *type*), 602
- dv::cluster::mean_shift::MeanShiftRowMajorMatrixXX (C++ *type*), 601
- dv::colorForEvent (C++ *function*), 584
- dv::colorForPoint (C++ *function*), 584
- dv::CompressionType (C++ *enum*), 582
- dv::CompressionType::LZ4 (C++ *enumerator*), 582
- dv::CompressionType::LZ4_HIGH (C++ *enumerator*), 582
- dv::CompressionType::MAX (C++ *enumerator*), 582
- dv::CompressionType::MIN (C++ *enumerator*), 582
- dv::CompressionType::NONE (C++ *enumerator*), 582
- dv::CompressionType::ZSTD (C++ *enumerator*), 582
- dv::CompressionType::ZSTD_HIGH (C++ *enumerator*), 582
- dv::CompressionTypeTypeTable (C++ *function*), 599
- dv::concepts (C++ *type*), 602
- dv::concepts::Accepts (C++ *concept*), 573
- dv::concepts::AddressableEvent (C++ *concept*), 573
- dv::concepts::BlockAccessible (C++ *concept*), 573
- dv::concepts::CompatibleWithSlicer (C++ *concept*), 573
- dv::concepts::Coordinate2D (C++ *concept*), 573
- dv::concepts::Coordinate2DAccessors (C++ *concept*), 574
- dv::concepts::Coordinate2DConstructible (C++ *concept*), 574
- dv::concepts::Coordinate2DIterable (C++ *concept*), 574
- dv::concepts::Coordinate2DMembers (C++ *concept*), 574
- dv::concepts::Coordinate2DMutableIterable (C++ *concept*), 574
- dv::concepts::Coordinate3D (C++ *concept*), 574
- dv::concepts::Coordinate3DAccessors (C++ *concept*), 574
- dv::concepts::Coordinate3DConstructible (C++ *concept*), 574
- dv::concepts::Coordinate3DIterable (C++ *concept*), 574

dv::concepts::Coordinate3DMembers (C++ *concept*), 574
 dv::concepts::Coordinate3DMutableIterable (C++ *concept*), 574
 dv::concepts::DataPacket (C++ *concept*), 574
 dv::concepts::DVFeatureDetectorAlgorithm (C++ *concept*), 574
 dv::concepts::EigenType (C++ *concept*), 574
 dv::concepts::Enum (C++ *concept*), 574
 dv::concepts::EventFilter (C++ *concept*), 574
 dv::concepts::EventOutputGenerator (C++ *concept*), 574
 dv::concepts::EventStorage (C++ *concept*), 574
 dv::concepts::EventToEventConverter (C++ *concept*), 574
 dv::concepts::EventToFrameConverter (C++ *concept*), 575
 dv::concepts::FeatureDetectorAlgorithm (C++ *concept*), 575
 dv::concepts::FlatbufferPacket (C++ *concept*), 575
 dv::concepts::FrameOutputGenerator (C++ *concept*), 575
 dv::concepts::FrameToEventConverter (C++ *concept*), 575
 dv::concepts::FrameToFrameConverter (C++ *concept*), 575
 dv::concepts::HasElementsVector (C++ *concept*), 575
 dv::concepts::HasTimestampedElementsVector (C++ *concept*), 575
 dv::concepts::HasTimestampedElementsVectorByAccessor (C++ *concept*), 575
 dv::concepts::HasTimestampedElementsVectorByMember (C++ *concept*), 575
 dv::concepts::InputStreamableFrom (C++ *concept*), 575
 dv::concepts::InputStreamableTo (C++ *concept*), 575
 dv::concepts::internal (C++ *type*), 603
 dv::concepts::internal::is_eigen_impl (C++ *struct*), 378
 dv::concepts::inter-
 nal::is_eigen_impl<Eigen::Matrix<T,
 Is...>> (C++ *struct*), 378
 dv::concepts::InvocableReturnArgumentsStrong (C++ *concept*), 575
 dv::concepts::InvocableReturnArgumentsWeak (C++ *concept*), 575
 dv::concepts::IOStreamableFrom (C++ *concept*), 576
 dv::concepts::IOStreamableTo (C++ *concept*), 576
 dv::concepts::is_eigen_type (C++ *member*), 603
 dv::concepts::is_type_one_of (C++ *member*), 603
 dv::concepts::Iterable (C++ *concept*), 576
 dv::concepts::iterable_element_type (C++ *type*), 602
 dv::concepts::KeyPoint (C++ *concept*), 576
 dv::concepts::KeyPointConstructible (C++ *concept*), 576
 dv::concepts::KeyPointIterable (C++ *concept*), 576
 dv::concepts::KeyPointMutableIterable (C++ *concept*), 576
 dv::concepts::MutableIterable (C++ *concept*), 576
 dv::concepts::number (C++ *concept*), 576
 dv::concepts::OpenCVFeatureDetectorAlgorithm (C++ *concept*), 576
 dv::concepts::OutputStreamableFrom (C++ *concept*), 576
 dv::concepts::OutputStreamableTo (C++ *concept*), 576
 dv::concepts::SupportsConstantDepth (C++ *concept*), 576
 dv::concepts::TimedImageContainer (C++ *concept*), 576
 dv::concepts::Timestamped (C++ *concept*), 576
 dv::concepts::TimestampedByAccessor (C++ *concept*), 576
 dv::concepts::TimestampedByMember (C++ *concept*), 576
 dv::concepts::TimestampedIterable (C++ *concept*), 576
 dv::concepts::TimestampMatrixContainer (C++ *concept*), 577
 dv::concepts::TimeSurface (C++ *concept*), 577
 dv::Constants (C++ *enum*), 582
 dv::Constants::AEDAT_VERSION_LENGTH (C++ *enumerator*), 582
 dv::Constants::MAX (C++ *enumerator*), 582
 dv::Constants::MIN (C++ *enumerator*), 582
 dv::ConstantsTypeTable (C++ *function*), 599
 dv::containers (C++ *type*), 603
 dv::containers::kd_tree (C++ *type*), 603
 dv::containers::kd_tree::KDTreeColMajor1X (C++ *type*), 603
 dv::containers::kd_tree::KDTreeColMajor2X (C++ *type*), 603
 dv::containers::kd_tree::KDTreeColMajor3X (C++ *type*), 604
 dv::containers::kd_tree::KDTreeColMajor4X (C++ *type*), 604
 dv::containers::kd_tree::KDTreeColMajorXX (C++ *type*), 603
 dv::containers::kd_tree::KDTreeEventStore-
 Adaptor (C++ *class*), 378

dv::containers::kd_tree::KDTreeEventStore-
 Adaptor::~~KDTreeEventStoreAdaptor
 (C++ function), 379
 dv::containers::kd_tree::KDTreeEventStore-
 Adaptor::begin (C++ function), 380
 dv::containers::kd_tree::KDTreeEventStore-
 Adaptor::derived (C++ function), 380
 dv::containers::kd_tree::KDTreeEventStore-
 Adaptor::end (C++ function), 380
 dv::containers::kd_tree::KDTreeEventStore-
 Adaptor::Index (C++ type), 380
 dv::containers::kd_tree::KDTreeEventStore-
 Adaptor::kdtree_get_bbox (C++ func-
 tion), 380
 dv::containers::kd_tree::KDTreeEventStore-
 Adaptor::kdtree_get_point_count
 (C++ function), 380
 dv::containers::kd_tree::KDTreeEventStore-
 Adaptor::kdtree_get_pt (C++ function),
 380
 dv::containers::kd_tree::KDTreeEventStore-
 Adaptor::KDTreeEventStoreAdaptor
 (C++ function), 378
 dv::containers::kd_tree::KDTreeEventStore-
 Adaptor::knnSearch (C++ function),
 379
 dv::containers::kd_tree::KDTreeEventStore-
 Adaptor::mData (C++ member), 381
 dv::containers::kd_tree::KDTreeEventStore-
 Adaptor::mIndex (C++ member), 381
 dv::containers::kd_tree::KDTreeEventStore-
 Adaptor::operator= (C++ function),
 378
 dv::containers::kd_tree::KDTreeEventStore-
 Adaptor::radiusSearch (C++ function),
 379
 dv::containers::kd_tree::KDTreeMatrixAdap-
 tor (C++ class), 381
 dv::containers::kd_tree::KDTreeMatrix-
 Adaptor::~~KDTreeMatrixAdaptor (C++
 function), 382
 dv::containers::kd_tree::KDTreeMatrixAdap-
 tor::DIMS (C++ member), 383
 dv::containers::kd_tree::KDTreeMatrixAdap-
 tor::getSample (C++ function), 382
 dv::containers::kd_tree::KDTreeMatrix-
 Adaptor::KDTreeMatrixAdaptor (C++
 function), 381, 382
 dv::containers::kd_tree::KDTreeMatrixAdap-
 tor::knnSearch (C++ function), 382
 dv::containers::kd_tree::KDTreeMatrixAdap-
 tor::Matrix (C++ type), 381
 dv::containers::kd_tree::KDTreeMatrixAdap-
 tor::mNumDimensions (C++ member),
 383
 dv::containers::kd_tree::KDTreeMatrix-
 Adaptor::mNumSamples (C++ member),
 383
 dv::containers::kd_tree::KDTreeMatrixAdap-
 tor::mTree (C++ member), 383
 dv::containers::kd_tree::KDTreeMatrixAdap-
 tor::NOT_SAMPLE_ORDER (C++ member),
 383
 dv::containers::kd_tree::KDTreeMatrixAdap-
 tor::operator= (C++ function), 382
 dv::containers::kd_tree::KDTreeMatrix-
 Adaptor::radiusSearch (C++ function),
 382
 dv::containers::kd_tree::KDTreeMatrix-
 Adaptor::STORAGE_ORDER (C++ member),
 383
 dv::containers::kd_tree::KDTreeMatrixAdap-
 tor::ThisType (C++ type), 383
 dv::containers::kd_tree::KDTreeMatrixAdap-
 tor::Tree (C++ type), 383
 dv::containers::kd_tree::KDTreeMatrixAdap-
 tor::Vector (C++ type), 381
 dv::containers::kd_tree::KDTreeRowMajorX1
 (C++ type), 603
 dv::containers::kd_tree::KDTreeRowMajorX2
 (C++ type), 603
 dv::containers::kd_tree::KDTreeRowMajorX3
 (C++ type), 603
 dv::containers::kd_tree::KDTreeRowMajorX4
 (C++ type), 603
 dv::containers::kd_tree::KDTreeRowMajorXX
 (C++ type), 603
 dv::coordinateHash (C++ function), 582
 dv::CreateBoundingBox (C++ function), 586
 dv::CreateBoundingBoxDirect (C++ function), 586
 dv::CreateBoundingBoxPacket (C++ function), 586
 dv::CreateBoundingBoxPacketDirect (C++ func-
 tion), 586
 dv::CreateDepthEventPacket (C++ function), 587
 dv::CreateDepthEventPacketDirect (C++ func-
 tion), 587
 dv::CreateDepthFrame (C++ function), 588
 dv::CreateDepthFrameDirect (C++ function), 588
 dv::CreateEventPacket (C++ function), 589
 dv::CreateEventPacketDirect (C++ function), 589
 dv::CreateFileDataDefinition (C++ function),
 598
 dv::CreateFileDataTable (C++ function), 598
 dv::CreateFileDataTableDirect (C++ function),
 598
 dv::CreateFrame (C++ function), 590
 dv::CreateFrameDirect (C++ function), 590
 dv::CreateIMU (C++ function), 592

`dv::CreateIMUPacket (C++ function), 592`
`dv::CreateIMUPacketDirect (C++ function), 592`
`dv::CreateIOHeader (C++ function), 599`
`dv::CreateIOHeaderDirect (C++ function), 599`
`dv::CreateLandmark (C++ function), 593`
`dv::CreateLandmarkDirect (C++ function), 593`
`dv::CreateLandmarksPacket (C++ function), 594`
`dv::CreateLandmarksPacketDirect (C++ function), 594`
`dv::CreateObservation (C++ function), 593`
`dv::CreateObservationDirect (C++ function), 593`
`dv::CreatePose (C++ function), 594, 595`
`dv::CreatePoseDirect (C++ function), 595`
`dv::CreateTimedKeyPoint (C++ function), 595`
`dv::CreateTimedKeyPointPacket (C++ function), 595, 596`
`dv::CreateTimedKeyPointPacketDirect (C++ function), 596`
`dv::CreateTrigger (C++ function), 597`
`dv::CreateTriggerPacket (C++ function), 597`
`dv::CreateTriggerPacketDirect (C++ function), 597`

`dv::data (C++ type), 604`
`dv::data::convertToCvPoints (C++ function), 604`
`dv::data::depthFrameFromCvMat (C++ function), 605`
`dv::data::depthFrameInMeters (C++ function), 605`
`dv::data::depthFrameMap (C++ function), 604`
`dv::data::fromCvKeypoints (C++ function), 604`
`dv::data::fromTimedKeyPoints (C++ function), 604`

`dv::data::generate (C++ type), 605`
`dv::data::generate::addNoiseToImu (C++ function), 608`
`dv::data::generate::dvLogo (C++ function), 607`
`dv::data::generate::dvLogoAsEvents (C++ function), 608`
`dv::data::generate::eventLine (C++ function), 606`
`dv::data::generate::eventRectangle (C++ function), 606`
`dv::data::generate::eventTestSet (C++ function), 606`
`dv::data::generate::imageToEvents (C++ function), 607`
`dv::data::generate::levelImuMeasurement (C++ function), 608`
`dv::data::generate::levelImuWithNoise (C++ function), 608`
`dv::data::generate::normallyDistributedEvents (C++ function), 606`
`dv::data::generate::sampleImage (C++ function), 605`

`dv::data::generate::uniformEventsWithinTimeRange (C++ function), 607`
`dv::data::generate::uniformlyDistributedEvents (C++ function), 606`
`dv::data::poseFromTransformation (C++ function), 605`
`dv::data::transformFromPose (C++ function), 605`
`dv::DEBUG_ENABLED (C++ member), 600`
`dv::depth (C++ type), 609`
`dv::depth::defaultStereoMatcher (C++ function), 609`
`dv::DepthEventPacket (C++ struct), 271`
`dv::DepthEventPacket::DepthEventPacket (C++ function), 271`
`dv::DepthEventPacket::elements (C++ member), 272`
`dv::DepthEventPacket::GetFullyQualified-Name (C++ function), 272`
`dv::DepthEventPacket::operator<< (C++ function), 272`
`dv::DepthEventPacket::TableType (C++ type), 271`

`dv::DepthEventPacketBufferHasIdentifier (C++ function), 588`
`dv::DepthEventPacketBuilder (C++ struct), 272`
`dv::DepthEventPacketBuilder::add_elements (C++ function), 272`
`dv::DepthEventPacketBuilder::DepthEvent-Builder (C++ function), 272`
`dv::DepthEventPacketBuilder::fbb_ (C++ member), 272`
`dv::DepthEventPacketBuilder::Finish (C++ function), 272`
`dv::DepthEventPacketBuilder::operator= (C++ function), 272`
`dv::DepthEventPacketBuilder::start_ (C++ member), 272`
`dv::DepthEventPacketFlatbuffer (C++ struct), 272`
`dv::DepthEventPacketFlatbuffer::elements (C++ function), 272`
`dv::DepthEventPacketFlatbuffer::GetFullyQualifiedName (C++ function), 273`
`dv::DepthEventPacketFlatbuffer::identifier (C++ member), 273`
`dv::DepthEventPacketFlatbuffer::MiniReflectTypeTable (C++ function), 273`
`dv::DepthEventPacketFlatbuffer::NativeTableType (C++ type), 272`
`dv::DepthEventPacketFlatbuffer::Pack (C++ function), 273`
`dv::DepthEventPacketFlatbuffer::UnPack (C++ function), 272`
`dv::DepthEventPacketFlatbuffer::UnPackTo`

(C++ function), 272
 dv::DepthEventPacketFlatbuffer::UnPack-
 ToFrom (C++ function), 273
 dv::DepthEventPacketFlatbuffer::Verify (C++
 function), 272
 dv::DepthEventPacketIdentifier (C++ function),
 588
 dv::DepthEventPacketTypeTable (C++ function),
 587
 dv::DepthEventStore (C++ type), 577
 dv::DepthEventTypeTable (C++ function), 587
 dv::DepthFrame (C++ struct), 273
 dv::DepthFrame::depth (C++ member), 273
 dv::DepthFrame::DepthFrame (C++ function), 273
 dv::DepthFrame::GetFullyQualified Name (C++
 function), 274
 dv::DepthFrame::maxDepth (C++ member), 273
 dv::DepthFrame::minDepth (C++ member), 273
 dv::DepthFrame::operator<< (C++ function), 274
 dv::DepthFrame::sizeX (C++ member), 273
 dv::DepthFrame::sizeY (C++ member), 273
 dv::DepthFrame::step (C++ member), 273
 dv::DepthFrame::TableType (C++ type), 273
 dv::DepthFrame::timestamp (C++ member), 273
 dv::DepthFrameBufferHasIdentifier (C++ func-
 tion), 589
 dv::DepthFrameBuilder (C++ struct), 274
 dv::DepthFrameBuilder::add_depth (C++ func-
 tion), 274
 dv::DepthFrameBuilder::add_maxDepth (C++
 function), 274
 dv::DepthFrameBuilder::add_minDepth (C++
 function), 274
 dv::DepthFrameBuilder::add_sizeX (C++ func-
 tion), 274
 dv::DepthFrameBuilder::add_sizeY (C++ func-
 tion), 274
 dv::DepthFrameBuilder::add_step (C++ func-
 tion), 274
 dv::DepthFrameBuilder::add_timestamp (C++
 function), 274
 dv::DepthFrameBuilder::DepthFrameBuilder
 (C++ function), 274
 dv::DepthFrameBuilder::fbb_ (C++ member), 274
 dv::DepthFrameBuilder::Finish (C++ function),
 274
 dv::DepthFrameBuilder::operator= (C++ func-
 tion), 274
 dv::DepthFrameBuilder::start_ (C++ member),
 274
 dv::DepthFrameFlatbuffer (C++ struct), 274
 dv::DepthFrameFlatbuffer::depth (C++ func-
 tion), 275
 dv::DepthFrameFlatbuffer::GetFullyQuali-
 fiedName (C++ function), 275
 dv::DepthFrameFlatbuffer::identifier (C++
 member), 275
 dv::DepthFrameFlatbuffer::maxDepth (C++
 function), 275
 dv::DepthFrameFlatbuffer::minDepth (C++
 function), 275
 dv::DepthFrameFlatbuffer::MiniReflectType-
 Table (C++ function), 275
 dv::DepthFrameFlatbuffer::NativeTableType
 (C++ type), 274
 dv::DepthFrameFlatbuffer::Pack (C++ function),
 275
 dv::DepthFrameFlatbuffer::sizeX (C++ func-
 tion), 274
 dv::DepthFrameFlatbuffer::sizeY (C++ func-
 tion), 274
 dv::DepthFrameFlatbuffer::step (C++ function),
 275
 dv::DepthFrameFlatbuffer::timestamp (C++
 function), 274
 dv::DepthFrameFlatbuffer::UnPack (C++ func-
 tion), 275
 dv::DepthFrameFlatbuffer::UnPackTo (C++
 function), 275
 dv::DepthFrameFlatbuffer::UnPackToFrom (C++
 function), 275
 dv::DepthFrameFlatbuffer::Verify (C++ func-
 tion), 275
 dv::DepthFrameIdentifier (C++ function), 589
 dv::DepthFrameTypeTable (C++ function), 588
 dv::Duration (C++ type), 577
 dv::EdgeMapAccumulator (C++ class), 315
 dv::EdgeMapAccumulator::accumulate (C++
 function), 316
 dv::EdgeMapAccumulator::buffer (C++ member),
 318
 dv::EdgeMapAccumulator::contribution (C++
 member), 318
 dv::EdgeMapAccumulator::decay (C++ member),
 318
 dv::EdgeMapAccumulator::DECAY_FULL (C++
 member), 317
 dv::EdgeMapAccumulator::DECAY_NONE (C++
 member), 317
 dv::EdgeMapAccumulator::decayLUT (C++ mem-
 ber), 318
 dv::EdgeMapAccumulator::DecayMode (C++
 enum), 317
 dv::EdgeMapAccumulator::decayMode (C++ mem-
 ber), 318
 dv::EdgeMapAccumulator::DecayMode::DECAY
 (C++ enumerator), 317
 dv::EdgeMapAccumulator::DecayMode::FULL

- (C++ enumerator), 317
- dv::EdgeMapAccumulator::DecayMode::NONE (C++ enumerator), 317
- dv::EdgeMapAccumulator::drawIncrement (C++ member), 318
- dv::EdgeMapAccumulator::EdgeMapAccumulator (C++ function), 315
- dv::EdgeMapAccumulator::generateFrame (C++ function), 316
- dv::EdgeMapAccumulator::getDecay (C++ function), 317
- dv::EdgeMapAccumulator::getEventContribution (C++ function), 316
- dv::EdgeMapAccumulator::getNeutralPotential (C++ function), 317
- dv::EdgeMapAccumulator::ignorePolarity (C++ member), 318
- dv::EdgeMapAccumulator::imageBuffer (C++ member), 318
- dv::EdgeMapAccumulator::incrementLUT (C++ member), 318
- dv::EdgeMapAccumulator::isIgnorePolarity (C++ function), 317
- dv::EdgeMapAccumulator::maxByteValue (C++ member), 318
- dv::EdgeMapAccumulator::neutralByteValue (C++ member), 318
- dv::EdgeMapAccumulator::neutralValue (C++ member), 318
- dv::EdgeMapAccumulator::operator<< (C++ function), 316, 318
- dv::EdgeMapAccumulator::reset (C++ function), 316
- dv::EdgeMapAccumulator::setDecay (C++ function), 317
- dv::EdgeMapAccumulator::setEventContribution (C++ function), 316
- dv::EdgeMapAccumulator::setIgnorePolarity (C++ function), 317
- dv::EdgeMapAccumulator::setNeutralPotential (C++ function), 317
- dv::EigenEvents (C++ struct), 319
- dv::EigenEvents::coordinates (C++ member), 319
- dv::EigenEvents::EigenEvents (C++ function), 319
- dv::EigenEvents::polarities (C++ member), 319
- dv::EigenEvents::timestamps (C++ member), 319
- dv::EnumAsInteger (C++ function), 585
- dv::EnumNameCompressionType (C++ function), 599
- dv::EnumNameConstants (C++ function), 599
- dv::EnumNameFrameFormat (C++ function), 590
- dv::EnumNameFrameSource (C++ function), 590
- dv::EnumNamesCompressionType (C++ function), 599
- dv::EnumNamesConstants (C++ function), 599
- dv::EnumNamesFrameFormat (C++ function), 590
- dv::EnumNamesFrameSource (C++ function), 590
- dv::EnumNamesTriggerType (C++ function), 597
- dv::EnumNameTriggerType (C++ function), 597
- dv::EnumValuesCompressionType (C++ function), 599
- dv::EnumValuesConstants (C++ function), 599
- dv::EnumValuesFrameFormat (C++ function), 590
- dv::EnumValuesFrameSource (C++ function), 590
- dv::EnumValuesTriggerType (C++ function), 596
- dv::errnoToString (C++ function), 585
- dv::EventColor (C++ enum), 578
- dv::EventColor::BLUE (C++ enumerator), 578
- dv::EventColor::GREEN1 (C++ enumerator), 578
- dv::EventColor::GREEN2 (C++ enumerator), 578
- dv::EventColor::RED (C++ enumerator), 578
- dv::EventColor::WHITE (C++ enumerator), 578
- dv::EventFilterBase (C++ class), 332
- dv::EventFilterBase::~EventFilterBase (C++ function), 332
- dv::EventFilterBase::accept (C++ function), 332
- dv::EventFilterBase::generateEvents (C++ function), 332
- dv::EventFilterBase::getNumberOfIncomingEvents (C++ function), 333
- dv::EventFilterBase::getNumberOfOutgoingEvents (C++ function), 333
- dv::EventFilterBase::getReductionFactor (C++ function), 333
- dv::EventFilterBase::mBuffer (C++ member), 333
- dv::EventFilterBase::mHighestProcessedTime (C++ member), 333
- dv::EventFilterBase::mNumberIncomingEvents (C++ member), 333
- dv::EventFilterBase::mNumberOutgoingEvents (C++ member), 333
- dv::EventFilterBase::operator>> (C++ function), 333
- dv::EventFilterBase::retain (C++ function), 332
- dv::EventFilterChain (C++ class), 333
- dv::EventFilterChain::addFilter (C++ function), 333
- dv::EventFilterChain::filters (C++ member), 334
- dv::EventFilterChain::operator<< (C++ function), 334
- dv::EventFilterChain::retain (C++ function), 334
- dv::EventMaskFilter (C++ class), 334
- dv::EventMaskFilter::EventMaskFilter (C++ function), 334

dv::EventMaskFilter::getMask (C++ function), 334
 dv::EventMaskFilter::mMask (C++ member), 335
 dv::EventMaskFilter::operator<< (C++ function), 334
 dv::EventMaskFilter::retain (C++ function), 334
 dv::EventMaskFilter::setMask (C++ function), 334
 dv::EventPacket (C++ struct), 335
 dv::EventPacket::elements (C++ member), 335
 dv::EventPacket::EventPacket (C++ function), 335
 dv::EventPacket::GetFullyQualifiedName (C++ function), 335
 dv::EventPacket::operator<< (C++ function), 335
 dv::EventPacket::TableType (C++ type), 335
 dv::EventPacketBufferHasIdentifier (C++ function), 589
 dv::EventPacketBuilder (C++ struct), 335
 dv::EventPacketBuilder::add_elements (C++ function), 335
 dv::EventPacketBuilder::EventPacketBuilder (C++ function), 335
 dv::EventPacketBuilder::fbb_ (C++ member), 336
 dv::EventPacketBuilder::Finish (C++ function), 335
 dv::EventPacketBuilder::operator= (C++ function), 335
 dv::EventPacketBuilder::start_ (C++ member), 336
 dv::EventPacketFlatbuffer (C++ struct), 336
 dv::EventPacketFlatbuffer::elements (C++ function), 336
 dv::EventPacketFlatbuffer::GetFullyQualifiedName (C++ function), 336
 dv::EventPacketFlatbuffer::identifier (C++ member), 336
 dv::EventPacketFlatbuffer::MiniReflectTypeTable (C++ function), 336
 dv::EventPacketFlatbuffer::NativeTableType (C++ type), 336
 dv::EventPacketFlatbuffer::Pack (C++ function), 336
 dv::EventPacketFlatbuffer::UnPack (C++ function), 336
 dv::EventPacketFlatbuffer::UnPackTo (C++ function), 336
 dv::EventPacketFlatbuffer::UnPackToFrom (C++ function), 336
 dv::EventPacketFlatbuffer::Verify (C++ function), 336
 dv::EventPacketIdentifier (C++ function), 589
 dv::EventPacketTypeTable (C++ function), 589
 dv::EventPolarityFilter (C++ class), 336
 dv::EventPolarityFilter::EventPolarityFilter (C++ function), 337
 dv::EventPolarityFilter::getPolarity (C++ function), 337
 dv::EventPolarityFilter::mPolarity (C++ member), 337
 dv::EventPolarityFilter::operator<< (C++ function), 337
 dv::EventPolarityFilter::retain (C++ function), 337
 dv::EventPolarityFilter::setPolarity (C++ function), 337
 dv::EventRegionFilter (C++ class), 337
 dv::EventRegionFilter::EventRegionFilter (C++ function), 337
 dv::EventRegionFilter::getRegion (C++ function), 337
 dv::EventRegionFilter::mRegion (C++ member), 338
 dv::EventRegionFilter::operator<< (C++ function), 338
 dv::EventRegionFilter::retain (C++ function), 338
 dv::EventRegionFilter::setRegion (C++ function), 338
 dv::EventStore (C++ type), 577
 dv::EventStreamSlicer (C++ type), 577
 dv::EventTimeComparator (C++ class), 338
 dv::EventTimeComparator::operator() (C++ function), 338
 dv::EventTypeTable (C++ function), 589
 dv::exceptions (C++ type), 609
 dv::exceptions::AedatFileError (C++ type), 610
 dv::exceptions::AedatFileParseError (C++ type), 610
 dv::exceptions::AedatVersionError (C++ type), 610
 dv::exceptions::BadAlloc (C++ type), 610
 dv::exceptions::DirectoryError (C++ type), 609
 dv::exceptions::DirectoryNotFound (C++ type), 609
 dv::exceptions::EndOfFile (C++ type), 610
 dv::exceptions::Exception (C++ class), 340
 dv::exceptions::Exception::~~Exception (C++ function), 340
 dv::exceptions::Exception::createInfo (C++ function), 340
 dv::exceptions::Exception::Exception (C++ function), 340
 dv::exceptions::Exception::mInfo (C++ member), 340
 dv::exceptions::Exception::mShortInfo (C++ member), 340

dv::exceptions::Exception::operator<< (C++ function), 340

dv::exceptions::Exception::shortWhat (C++ function), 340

dv::exceptions::Exception::what (C++ function), 340

dv::exceptions::FileError (C++ type), 609

dv::exceptions::FileNotFound (C++ type), 610

dv::exceptions::FileOpenError (C++ type), 609

dv::exceptions::FileReadError (C++ type), 609

dv::exceptions::FileWriteError (C++ type), 610

dv::exceptions::info (C++ type), 610

dv::exceptions::info::AedatFileError (C++ struct), 196

dv::exceptions::info::AedatFileError::Info (C++ type), 197

dv::exceptions::info::AedatFileParseError (C++ struct), 197

dv::exceptions::info::AedatFileParseError::format (C++ function), 197

dv::exceptions::info::AedatFileParseError::Info (C++ type), 197

dv::exceptions::info::AedatVersionError (C++ struct), 197

dv::exceptions::info::AedatVersionError::format (C++ function), 197

dv::exceptions::info::AedatVersionError::Info (C++ type), 197

dv::exceptions::info::BadAlloc (C++ struct), 200

dv::exceptions::info::DirectoryError (C++ struct), 276

dv::exceptions::info::DirectoryError::format (C++ function), 276

dv::exceptions::info::DirectoryError::Info (C++ type), 276

dv::exceptions::info::DirectoryNotFound (C++ struct), 276

dv::exceptions::info::DirectoryNotFound::format (C++ function), 276

dv::exceptions::info::DirectoryNotFound::Info (C++ type), 276

dv::exceptions::info::EmptyException (C++ struct), 319

dv::exceptions::info::EmptyException::Info (C++ type), 319

dv::exceptions::info::EndOfFile (C++ struct), 319

dv::exceptions::info::EndOfFile::format (C++ function), 319

dv::exceptions::info::EndOfFile::Info (C++ type), 319

dv::exceptions::info::FileError (C++ struct), 353

dv::exceptions::info::FileError::format (C++ function), 353

dv::exceptions::info::FileError::Info (C++ type), 353

dv::exceptions::info::FileNotFound (C++ struct), 354

dv::exceptions::info::FileNotFound::format (C++ function), 354

dv::exceptions::info::FileNotFound::Info (C++ type), 354

dv::exceptions::info::FileOpenError (C++ struct), 354

dv::exceptions::info::FileOpenError::format (C++ function), 354

dv::exceptions::info::FileOpenError::Info (C++ type), 354

dv::exceptions::info::FileReadError (C++ struct), 354

dv::exceptions::info::FileReadError::format (C++ function), 354

dv::exceptions::info::FileReadError::Info (C++ type), 354

dv::exceptions::info::FileWriteError (C++ struct), 354

dv::exceptions::info::FileWriteError::format (C++ function), 355

dv::exceptions::info::FileWriteError::Info (C++ type), 354

dv::exceptions::info::InputError (C++ struct), 374

dv::exceptions::info::InputError::ErrorInfo (C++ struct), 319

dv::exceptions::info::InputError::ErrorInfo::mName (C++ member), 320

dv::exceptions::info::InputError::ErrorInfo::mTypeIdentifier (C++ member), 320

dv::exceptions::info::InputError::format (C++ function), 375

dv::exceptions::info::InputError::Info (C++ type), 374

dv::exceptions::info::InvalidArgument (C++ struct), 375

dv::exceptions::info::InvalidArgument::format (C++ function), 375

dv::exceptions::info::InvalidArgument::Info (C++ type), 375

dv::exceptions::info::IOError (C++ struct), 375

dv::exceptions::info::LengthError (C++ struct), 390

dv::exceptions::info::NullPointer (C++ struct), 451

dv::exceptions::info::OutOfRange (C++ struct), 455

dv::exceptions::info::OutputError (C++ struct), 456
 dv::exceptions::info::OutputError::Error-Info (C++ struct), 320
 dv::exceptions::info::OutputError::Error-Info::mName (C++ member), 320
 dv::exceptions::info::OutputError::Error-Info::mTypeIdentifier (C++ member), 320
 dv::exceptions::info::OutputError::format (C++ function), 456
 dv::exceptions::info::OutputError::Info (C++ type), 456
 dv::exceptions::info::RuntimeError (C++ struct), 500
 dv::exceptions::info::TypeError (C++ struct), 556
 dv::exceptions::info::TypeError::format (C++ function), 556
 dv::exceptions::info::TypeError::Info (C++ type), 556
 dv::exceptions::InputError (C++ type), 610
 dv::exceptions::internal (C++ type), 610
 dv::exceptions::internal::Exception_ (C++ class), 340
 dv::exceptions::internal::Exception_::~~Exception_ (C++ function), 341
 dv::exceptions::internal::Exception_::Exception_ (C++ function), 341
 dv::exceptions::internal::Exception_::Info (C++ type), 341
 dv::exceptions::internal::Exception_::operator<< (C++ function), 341
 dv::exceptions::internal::format (C++ function), 611
 dv::exceptions::internal::HasCustomExceptionFormatter (C++ concept), 577
 dv::exceptions::internal::HasExtraExceptionInfo (C++ concept), 577
 dv::exceptions::internal::NoCustomExceptionFormatter (C++ concept), 577
 dv::exceptions::InvalidArgument (C++ type), 610
 dv::exceptions::IOError (C++ type), 610
 dv::exceptions::LengthError (C++ type), 610
 dv::exceptions::NullPointer (C++ type), 610
 dv::exceptions::OutOfRange (C++ type), 610
 dv::exceptions::OutputError (C++ type), 610
 dv::exceptions::RuntimeError (C++ type), 610
 dv::exceptions::TypeError (C++ type), 610
 dv::features (C++ type), 611
 dv::features::ArcCornerDetector (C++ class), 197
 dv::features::ArcCornerDetector::Arc-
 CornerDetector (C++ function), 197
 dv::features::ArcCornerDetector::ArcLimits (C++ class), 199
 dv::features::ArcCornerDetector::Arc-
 cLimits::ArcLimits (C++ function), 199
 dv::features::ArcCornerDetector::Arc-
 cLimits::MAX_ARC_SIZE_FACTOR (C++ member), 199
 dv::features::ArcCornerDetector::Arc-
 cLimits::mCircumference (C++ member), 199
 dv::features::ArcCornerDetector::Arc-
 cLimits::MIN_ARC_SIZE_FACTOR (C++ member), 199
 dv::features::ArcCornerDetector::Arc-
 cLimits::mMaxSize (C++ member), 199
 dv::features::ArcCornerDetector::Arc-
 cLimits::mMinSize (C++ member), 199
 dv::features::ArcCornerDetector::Arc-
 cLimits::satisfied (C++ function), 199
 dv::features::ArcCornerDetector::check-
 SurroundingTimestamps (C++ function), 198
 dv::features::ArcCornerDetector::Circular-
 TimeSurfaceView (C++ class), 226
 dv::features::ArcCornerDetector::Circular-
 TimeSurfaceView::circularDecrement (C++ function), 226
 dv::features::ArcCornerDetector::Circular-
 TimeSurfaceView::circularIncrement (C++ function), 226
 dv::features::ArcCornerDetector::Circular-
 TimeSurfaceView::CircularTimeSur-
 faceView (C++ function), 226
 dv::features::ArcCornerDetector::Circular-
 TimeSurfaceView::CoordVector (C++ type), 226
 dv::features::ArcCornerDetector::Circular-
 TimeSurfaceView::getTimestamp (C++ function), 226
 dv::features::ArcCornerDetector::Circu-
 larTimeSurfaceView::mCoords (C++ member), 226
 dv::features::ArcCornerDetector::detect (C++ function), 198
 dv::features::ArcCornerDetector::expandArc (C++ function), 198
 dv::features::ArcCornerDetector::get-
 TimeSurface (C++ function), 198
 dv::features::ArcCornerDetector::insid-
 eCorner (C++ function), 198
 dv::features::ArcCornerDetector::mAr-

cLimits (C++ member), 199
 dv::features::ArcCornerDetector::mCircles (C++ member), 199
 dv::features::ArcCornerDetector::mCornerRange (C++ member), 199
 dv::features::ArcCornerDetector::mResetTsAfterDetection (C++ member), 199
 dv::features::ArcCornerDetector::mTimeSurfaces (C++ member), 199
 dv::features::ArcCornerDetector::SharedPtr (C++ type), 197
 dv::features::ArcCornerDetector::UniquePtr (C++ type), 197
 dv::features::EventBlobDetector (C++ class), 321
 dv::features::EventBlobDetector::defaultBlobDetector (C++ function), 322
 dv::features::EventBlobDetector::detect (C++ function), 322
 dv::features::EventBlobDetector::EventBlobDetector (C++ function), 322
 dv::features::EventBlobDetector::mAccumulator (C++ member), 323
 dv::features::EventBlobDetector::mBlobDetector (C++ member), 323
 dv::features::EventBlobDetector::mPreprocessFcn (C++ member), 323
 dv::features::EventBlobDetector::mPyramidLevel (C++ member), 323
 dv::features::EventCombinedLKTracker (C++ class), 323
 dv::features::EventCombinedLKTracker::accept (C++ function), 323, 325
 dv::features::EventCombinedLKTracker::EventCombinedLKTracker (C++ function), 327
 dv::features::EventCombinedLKTracker::getAccumulatedFrames (C++ function), 323
 dv::features::EventCombinedLKTracker::getEventTrackPoints (C++ function), 323
 dv::features::EventCombinedLKTracker::getMinRateForIntermediateTracking (C++ function), 325
 dv::features::EventCombinedLKTracker::getNumEvents (C++ function), 324
 dv::features::EventCombinedLKTracker::getNumIntermediateFrames (C++ function), 324
 dv::features::EventCombinedLKTracker::getStoreTimeLimit (C++ function), 324
 dv::features::EventCombinedLKTracker::mAccumulatedFrames (C++ member), 327
 dv::features::EventCombinedLKTracker::mAc-cumulator (C++ member), 327
 dv::features::EventCombinedLKTracker::mEventBuffer (C++ member), 327
 dv::features::EventCombinedLKTracker::mEventTrackPoints (C++ member), 327
 dv::features::EventCombinedLKTracker::mMinRateForIntermediateTracking (C++ member), 327
 dv::features::EventCombinedLKTracker::mNumberOfEvents (C++ member), 327
 dv::features::EventCombinedLKTracker::mNumIntermediateFrames (C++ member), 327
 dv::features::EventCombinedLKTracker::MotionAwareTracker (C++ function), 326
 dv::features::EventCombinedLKTracker::mStoreTimeLimit (C++ member), 327
 dv::features::EventCombinedLKTracker::RegularTracker (C++ function), 325
 dv::features::EventCombinedLKTracker::setAccumulator (C++ function), 324
 dv::features::EventCombinedLKTracker::setConstantDepth (C++ function), 325
 dv::features::EventCombinedLKTracker::setMinRateForIntermediateTracking (C++ function), 325
 dv::features::EventCombinedLKTracker::setNumberOfEvents (C++ function), 324
 dv::features::EventCombinedLKTracker::setNumIntermediateFrames (C++ function), 324
 dv::features::EventCombinedLKTracker::setStoreTimeLimit (C++ function), 324
 dv::features::EventCombinedLKTracker::SharedPtr (C++ type), 323
 dv::features::EventCombinedLKTracker::track (C++ function), 327
 dv::features::EventCombinedLKTracker::trackIntermediateEvents (C++ function), 326
 dv::features::EventCombinedLKTracker::UniquePtr (C++ type), 323
 dv::features::EventFeatureBlobDetector (C++ type), 611
 dv::features::EventFeatureLKTracker (C++ class), 327
 dv::features::EventFeatureLKTracker::accept (C++ function), 328, 329, 332

dv::features::EventFeatureLKTracker::EventFeatureLKTracker (C++ function), 331
 dv::features::EventFeatureLKTracker::getAccumulatedFrame (C++ function), 328
 dv::features::EventFeatureLKTracker::getFramerate (C++ function), 328
 dv::features::EventFeatureLKTracker::getNumberOfEvents (C++ function), 329
 dv::features::EventFeatureLKTracker::getStoreTimeLimit (C++ function), 328
 dv::features::EventFeatureLKTracker::mAccumulatedFrame (C++ member), 332
 dv::features::EventFeatureLKTracker::mAccumulator (C++ member), 331
 dv::features::EventFeatureLKTracker::mEventBuffer (C++ member), 331
 dv::features::EventFeatureLKTracker::mFramerate (C++ member), 331
 dv::features::EventFeatureLKTracker::mLastRunTimestamp (C++ member), 331
 dv::features::EventFeatureLKTracker::mNumberOfEvents (C++ member), 331
 dv::features::EventFeatureLKTracker::MotionAwareTracker (C++ function), 330
 dv::features::EventFeatureLKTracker::mPeriod (C++ member), 331
 dv::features::EventFeatureLKTracker::mStoreTimeLimit (C++ member), 331
 dv::features::EventFeatureLKTracker::RegularTracker (C++ function), 330
 dv::features::EventFeatureLKTracker::setAccumulator (C++ function), 329
 dv::features::EventFeatureLKTracker::setConstantDepth (C++ function), 329
 dv::features::EventFeatureLKTracker::setFramerate (C++ function), 328
 dv::features::EventFeatureLKTracker::setNumberOfEvents (C++ function), 329
 dv::features::EventFeatureLKTracker::setStoreTimeLimit (C++ function), 329
 dv::features::EventFeatureLKTracker::SharedPtr (C++ type), 328
 dv::features::EventFeatureLKTracker::track (C++ function), 331
 dv::features::EventFeatureLKTracker::UniquePtr (C++ type), 328
 dv::features::FeatureCountRedetection (C++ class), 343
 dv::features::FeatureCountRedetection::detectRedetection (C++ function), 343
 dv::features::FeatureCountRedetection::FeatureCountRedetection (C++ function), 343
 dv::features::FeatureCountRedetection::mMinimumProportionOfTracks (C++ member), 343
 dv::features::FeatureDetector (C++ class), 343
 dv::features::FeatureDetector::~~FeatureDetector (C++ function), 345
 dv::features::FeatureDetector::AlgorithmPtr (C++ type), 344
 dv::features::FeatureDetector::classIdCounter (C++ member), 347
 dv::features::FeatureDetector::detect (C++ function), 346
 dv::features::FeatureDetector::detector (C++ member), 347
 dv::features::FeatureDetector::FeatureDetector (C++ function), 344
 dv::features::FeatureDetector::FeaturePostProcessing (C++ enum), 344
 dv::features::FeatureDetector::FeaturePostProcessing::ADAPTIVE_NMS (C++ enumerator), 344
 dv::features::FeatureDetector::FeaturePostProcessing::NONE (C++ enumerator), 344
 dv::features::FeatureDetector::FeaturePostProcessing::TOP_N (C++ enumerator), 344
 dv::features::FeatureDetector::FIND_ALL (C++ member), 346
 dv::features::FeatureDetector::getImageDimensions (C++ function), 346
 dv::features::FeatureDetector::getMargin (C++ function), 346
 dv::features::FeatureDetector::getMarginROI (C++ function), 346
 dv::features::FeatureDetector::getPostProcessing (C++ function), 345
 dv::features::FeatureDetector::imageDimensions (C++ member), 347
 dv::features::FeatureDetector::isWithinROI (C++ function), 346
 dv::features::FeatureDetector::margin (C++ member), 347
 dv::features::FeatureDetector::operator<< (C++ function), 347
 dv::features::FeatureDetector::postProcessing (C++ member), 347
 dv::features::FeatureDetector::resampler (C++ member), 347
 dv::features::FeatureDetector::roiBuffered

(C++ member), 347

dv::features::FeatureDetector::runDetection (C++ function), 345

dv::features::FeatureDetector::runRedetection (C++ function), 345

dv::features::FeatureDetector::setMargin (C++ function), 346

dv::features::FeatureDetector::setPostProcessing (C++ function), 345

dv::features::FeatureDetector::SharedPtr (C++ type), 344

dv::features::FeatureDetector::ThisType (C++ type), 344

dv::features::FeatureDetector::UniquePtr (C++ type), 344

dv::features::FeatureTracks (C++ class), 347

dv::features::FeatureTracks::accept (C++ function), 347, 348

dv::features::FeatureTracks::addKeypoint (C++ function), 349

dv::features::FeatureTracks::clear (C++ function), 349

dv::features::FeatureTracks::eachTrack (C++ function), 348

dv::features::FeatureTracks::getHighestTime (C++ function), 349

dv::features::FeatureTracks::getHistoryDuration (C++ function), 348

dv::features::FeatureTracks::getLatestTrackKeypoints (C++ function), 348

dv::features::FeatureTracks::getTrack (C++ function), 348

dv::features::FeatureTracks::getTrackIds (C++ function), 348

dv::features::FeatureTracks::getTrackTimeout (C++ function), 349

dv::features::FeatureTracks::isEmpty (C++ function), 349

dv::features::FeatureTracks::maintainBufferDuration (C++ function), 349

dv::features::FeatureTracks::mHighestTime (C++ member), 350

dv::features::FeatureTracks::mHistory (C++ member), 349

dv::features::FeatureTracks::mHistoryDuration (C++ member), 349

dv::features::FeatureTracks::mTrackTimeout (C++ member), 349

dv::features::FeatureTracks::setHistoryDuration (C++ function), 348

dv::features::FeatureTracks::setTrackTimeout (C++ function), 349

dv::features::FeatureTracks::visualize (C++ function), 348

dv::features::ImageFeatureDetector (C++ type), 611

dv::features::ImageFeatureLKTracker (C++ class), 361

dv::features::ImageFeatureLKTracker::accept (C++ function), 362

dv::features::ImageFeatureLKTracker::Config (C++ type), 361

dv::features::ImageFeatureLKTracker::constantDepth (C++ member), 365

dv::features::ImageFeatureLKTracker::depthHistoryDuration (C++ member), 365

dv::features::ImageFeatureLKTracker::getConstantDepth (C++ function), 363

dv::features::ImageFeatureLKTracker::getRejectionDistanceThreshold (C++ function), 363

dv::features::ImageFeatureLKTracker::ImageFeatureLKTracker (C++ function), 364

dv::features::ImageFeatureLKTracker::isLookbackRejectionEnabled (C++ function), 362

dv::features::ImageFeatureLKTracker::mCamera (C++ member), 365

dv::features::ImageFeatureLKTracker::mConfig (C++ member), 365

dv::features::ImageFeatureLKTracker::mCurrentFrame (C++ member), 365

dv::features::ImageFeatureLKTracker::mDepthHistory (C++ member), 365

dv::features::ImageFeatureLKTracker::mDetector (C++ member), 365

dv::features::ImageFeatureLKTracker::mLookbackRejection (C++ member), 365

dv::features::ImageFeatureLKTracker::MotionAwareTracker (C++ function), 364

dv::features::ImageFeatureLKTracker::mPredictor (C++ member), 365

dv::features::ImageFeatureLKTracker::mPreviousFrame (C++ member), 365

dv::features::ImageFeatureLKTracker::mRejectionStrategy (C++ member), 365

dv::features::ImageFeatureLKTracker::mRejectionDistanceThreshold (C++ member), 365

dv::features::ImageFeatureLKTracker::mResolution (C++ member), 365

dv::features::ImageFeatureLKTracker::mTracker (C++ member), 365

dv::features::ImageFeatureLK-

Tracker::mTransformer (C++ member), 365

dv::features::ImageFeatureLKTracker::predictNextPoints (C++ function), 364

dv::features::ImageFeatureLKTracker::RegularTracker (C++ function), 364

dv::features::ImageFeatureLKTracker::setConstantDepth (C++ function), 363

dv::features::ImageFeatureLKTracker::setDetector (C++ function), 362

dv::features::ImageFeatureLKTracker::setLookbackRejection (C++ function), 362

dv::features::ImageFeatureLKTracker::setMotionPredictor (C++ function), 362

dv::features::ImageFeatureLKTracker::setRedetectionStrategy (C++ function), 362

dv::features::ImageFeatureLKTracker::setRejectionDistanceThreshold (C++ function), 363

dv::features::ImageFeatureLKTracker::SharedPtr (C++ type), 361

dv::features::ImageFeatureLKTracker::track (C++ function), 364

dv::features::ImageFeatureLKTracker::UniquePtr (C++ type), 361

dv::features::ImagePyramid (C++ class), 365

dv::features::ImagePyramid::ImagePyramid (C++ function), 366

dv::features::ImagePyramid::pyramid (C++ member), 366

dv::features::ImagePyramid::SharedPtr (C++ type), 365

dv::features::ImagePyramid::timestamp (C++ member), 366

dv::features::ImagePyramid::UniquePtr (C++ type), 365

dv::features::ImagePyrFeatureDetector (C++ type), 611

dv::features::internal (C++ type), 611

dv::features::internal::CircleCoordinates (C++ struct), 224

dv::features::internal::CircleCoordinates<3> (C++ struct), 224

dv::features::internal::CircleCoordinates<3>::coords (C++ member), 224

dv::features::internal::CircleCoordinates<4> (C++ struct), 224

dv::features::internal::CircleCoordinates<4>::coords (C++ member), 225

dv::features::internal::CircleCoordinates<5> (C++ struct), 225

dv::features::internal::CircleCoordinates<5>::coords (C++ member), 225

dv::features::internal::CircleCoordinates<6> (C++ struct), 225

dv::features::internal::CircleCoordinates<6>::coords (C++ member), 225

dv::features::internal::CircleCoordinates<7> (C++ struct), 225

dv::features::internal::CircleCoordinates<7>::coords (C++ member), 225

dv::features::KeyPointResampler (C++ class), 383

dv::features::KeyPointResampler::getTolerance (C++ function), 384

dv::features::KeyPointResampler::KeyPointResampler (C++ function), 383

dv::features::KeyPointResampler::mCols (C++ member), 384

dv::features::KeyPointResampler::mPreviousSolution (C++ member), 384

dv::features::KeyPointResampler::mRows (C++ member), 384

dv::features::KeyPointResampler::mTolerance (C++ member), 384

dv::features::KeyPointResampler::RangeValue (C++ type), 384

dv::features::KeyPointResampler::setTolerance (C++ function), 384

dv::features::LucasKanadeConfig (C++ struct), 394

dv::features::LucasKanadeConfig::maskedFeatureDetect (C++ member), 394

dv::features::LucasKanadeConfig::numPyramidLayers (C++ member), 395

dv::features::LucasKanadeConfig::searchWindowSize (C++ member), 395

dv::features::LucasKanadeConfig::terminationEpsilon (C++ member), 394

dv::features::MeanShiftTracker (C++ class), 409

dv::features::MeanShiftTracker::accept (C++ function), 410

dv::features::MeanShiftTracker::computeShift (C++ function), 412

dv::features::MeanShiftTracker::findSpatialWindow (C++ function), 413

dv::features::MeanShiftTracker::getBandwidth (C++ function), 411

dv::features::MeanShiftTracker::getConvergenceNorm (C++ function), 411

dv::features::MeanShiftTracker::getMaxIterations (C++ function), 412

dv::features::MeanShiftTracker::getStepSize (C++ function), 411

dv::features::MeanShiftTracker::getTimeWindow (C++ function), 411

dv::features::MeanShiftTracker::getWeight-

Multiplier (C++ function), 411

dv::features::MeanShiftTracker::kernelEpanechnikovWeights (C++ function), 413

dv::features::MeanShiftTracker::mBandwidth (C++ member), 413

dv::features::MeanShiftTracker::mConvergenceNorm (C++ member), 414

dv::features::MeanShiftTracker::mDetector (C++ member), 414

dv::features::MeanShiftTracker::MeanShiftTracker (C++ function), 410

dv::features::MeanShiftTracker::mEvents (C++ member), 414

dv::features::MeanShiftTracker::mLastFreeClassId (C++ member), 414

dv::features::MeanShiftTracker::mMaxIters (C++ member), 414

dv::features::MeanShiftTracker::mRedetectionStrategy (C++ member), 414

dv::features::MeanShiftTracker::mResolution (C++ member), 414

dv::features::MeanShiftTracker::mStepSize (C++ member), 414

dv::features::MeanShiftTracker::mSurface (C++ member), 413

dv::features::MeanShiftTracker::mTimeWindow (C++ member), 413

dv::features::MeanShiftTracker::mWeightMultiplier (C++ member), 414

dv::features::MeanShiftTracker::runRedetection (C++ function), 413

dv::features::MeanShiftTracker::setBandwidth (C++ function), 411

dv::features::MeanShiftTracker::setConvergenceNorm (C++ function), 412

dv::features::MeanShiftTracker::setDetector (C++ function), 410

dv::features::MeanShiftTracker::setMaxIterations (C++ function), 412

dv::features::MeanShiftTracker::setRedetectionStrategy (C++ function), 410

dv::features::MeanShiftTracker::setStepSize (C++ function), 411

dv::features::MeanShiftTracker::setTimeWindow (C++ function), 411

dv::features::MeanShiftTracker::setWeightMultiplier (C++ function), 411

dv::features::MeanShiftTracker::track (C++ function), 410

dv::features::MeanShiftTracker::updateCenterLocation (C++ function), 412

dv::features::MeanShiftTracker::updateTracks (C++ function), 412

dv::features::NoRedetection (C++ class), 451

dv::features::NoRedetection::decideRedetection (C++ function), 451

dv::features::RedetectionStrategy (C++ class), 496

dv::features::RedetectionStrategy::~~RedetectionStrategy (C++ function), 497

dv::features::RedetectionStrategy::decideRedetection (C++ function), 497

dv::features::RedetectionStrategy::SharedPtr (C++ type), 496

dv::features::RedetectionStrategy::UniquePtr (C++ type), 496

dv::features::TrackerBase (C++ class), 547

dv::features::TrackerBase::~~TrackerBase (C++ function), 547

dv::features::TrackerBase::getLastFrameResults (C++ function), 547

dv::features::TrackerBase::getMaxTracks (C++ function), 547

dv::features::TrackerBase::lastFrameResults (C++ member), 548

dv::features::TrackerBase::maxTracks (C++ member), 548

dv::features::TrackerBase::removeTracks (C++ function), 547

dv::features::TrackerBase::Result (C++ struct), 497

dv::features::TrackerBase::Result::asKeyFrame (C++ member), 497

dv::features::TrackerBase::Result::ConstPtr (C++ type), 497

dv::features::TrackerBase::Result::keypoints (C++ member), 497

dv::features::TrackerBase::Result::Result (C++ function), 497

dv::features::TrackerBase::Result::SharedPtr (C++ type), 497

dv::features::TrackerBase::Result::timestamp (C++ member), 497

dv::features::TrackerBase::runTracking (C++ function), 547

dv::features::TrackerBase::setMaxTracks (C++ function), 547

dv::features::TrackerBase::SharedPtr (C++ type), 547

dv::features::TrackerBase::track (C++ function), 548

dv::features::TrackerBase::UniquePtr (C++ type), 547

dv::features::UpdateIntervalOrFeatureCounterRedetection (C++ class), 557

dv::features::UpdateIntervalOrFeatureCounterRedetection::decideRedetection

- (C++ function), 557
- dv::features::UpdateIntervalOrFeatureCountRedetection::featureCountRedetection (C++ member), 557
- dv::features::UpdateIntervalOrFeatureCountRedetection::UpdateIntervalOrFeatureCountRedetection (C++ function), 557
- dv::features::UpdateIntervalOrFeatureCountRedetection::updateIntervalRedetection (C++ member), 557
- dv::features::UpdateIntervalRedetection (C++ class), 557
- dv::features::UpdateIntervalRedetection::decideRedetection (C++ function), 558
- dv::features::UpdateIntervalRedetection::mLastDetectionTime (C++ member), 558
- dv::features::UpdateIntervalRedetection::mUpdateIntervalTime (C++ member), 558
- dv::features::UpdateIntervalRedetection::UpdateIntervalRedetection (C++ function), 558
- dv::FileDataDefinition (C++ struct), 350
- dv::FileDataDefinition::ByteOffset (C++ member), 350
- dv::FileDataDefinition::FileDataDefinition (C++ function), 350
- dv::FileDataDefinition::GetFullyQualifiedName (C++ function), 350
- dv::FileDataDefinition::NumElements (C++ member), 350
- dv::FileDataDefinition::PacketInfo (C++ member), 350
- dv::FileDataDefinition::TableType (C++ type), 350
- dv::FileDataDefinition::TimestampEnd (C++ member), 350
- dv::FileDataDefinition::TimestampStart (C++ member), 350
- dv::FileDataDefinitionBuilder (C++ struct), 350
- dv::FileDataDefinitionBuilder::add_ByteOffset (C++ function), 350
- dv::FileDataDefinitionBuilder::add_NumElements (C++ function), 350
- dv::FileDataDefinitionBuilder::add_PacketInfo (C++ function), 350
- dv::FileDataDefinitionBuilder::add_TimestampEnd (C++ function), 350
- dv::FileDataDefinitionBuilder::add_TimestampStart (C++ function), 350
- dv::FileDataDefinitionBuilder::fbb_ (C++ member), 351
- dv::FileDataDefinitionBuilder::FileDataDefinitionBuilder (C++ function), 350
- dv::FileDataDefinitionBuilder::Finish (C++ function), 351
- dv::FileDataDefinitionBuilder::operator= (C++ function), 350
- dv::FileDataDefinitionBuilder::start_ (C++ member), 351
- dv::FileDataDefinitionFlatbuffer (C++ struct), 351
- dv::FileDataDefinitionFlatbuffer::ByteOffset (C++ function), 351
- dv::FileDataDefinitionFlatbuffer::GetFullyQualifiedName (C++ function), 351
- dv::FileDataDefinitionFlatbuffer::MiniReflectTypeTable (C++ function), 351
- dv::FileDataDefinitionFlatbuffer::NativeTableType (C++ type), 351
- dv::FileDataDefinitionFlatbuffer::NumElements (C++ function), 351
- dv::FileDataDefinitionFlatbuffer::Pack (C++ function), 351
- dv::FileDataDefinitionFlatbuffer::PacketInfo (C++ function), 351
- dv::FileDataDefinitionFlatbuffer::TimestampEnd (C++ function), 351
- dv::FileDataDefinitionFlatbuffer::TimestampStart (C++ function), 351
- dv::FileDataDefinitionFlatbuffer::UnPack (C++ function), 351
- dv::FileDataDefinitionFlatbuffer::UnPackTo (C++ function), 351
- dv::FileDataDefinitionFlatbuffer::UnPackToFrom (C++ function), 351
- dv::FileDataDefinitionFlatbuffer::Verify (C++ function), 351
- dv::FileDataDefinitionTypeTable (C++ function), 598
- dv::FileDataTable (C++ struct), 351
- dv::FileDataTable::FileDataTable (C++ function), 352
- dv::FileDataTable::GetFullyQualifiedName (C++ function), 352
- dv::FileDataTable::Table (C++ member), 352
- dv::FileDataTable::TableType (C++ type), 352
- dv::FileDataTableBufferHasIdentifier (C++ function), 598
- dv::FileDataTableBuilder (C++ struct), 352
- dv::FileDataTableBuilder::add_Table (C++ function), 352
- dv::FileDataTableBuilder::fbb_ (C++ member), 352

dv::FileDataTableBuilder::FileDataTable-
 Builder (C++ function), 352
 dv::FileDataTableBuilder::Finish (C++ func-
 tion), 352
 dv::FileDataTableBuilder::operator= (C++
 function), 352
 dv::FileDataTableBuilder::start_ (C++ mem-
 ber), 352
 dv::FileDataTableFlatbuffer (C++ struct), 352
 dv::FileDataTableFlatbuffer::GetFullyQual-
 ifiedName (C++ function), 353
 dv::FileDataTableFlatbuffer::identifier
 (C++ member), 353
 dv::FileDataTableFlatbuffer::MiniReflect-
 TypeTable (C++ function), 353
 dv::FileDataTableFlatbuffer::NativeTable-
 Type (C++ type), 352
 dv::FileDataTableFlatbuffer::Pack (C++ func-
 tion), 353
 dv::FileDataTableFlatbuffer::Table (C++
 function), 352
 dv::FileDataTableFlatbuffer::UnPack (C++
 function), 352
 dv::FileDataTableFlatbuffer::UnPackTo (C++
 function), 353
 dv::FileDataTableFlatbuffer::UnPackToFrom
 (C++ function), 353
 dv::FileDataTableFlatbuffer::Verify (C++
 function), 352
 dv::FileDataTableIdentifier (C++ function), 598
 dv::FileDataTableTypeTable (C++ function), 598
 dv::FinishBoundingBoxPacketBuffer (C++ func-
 tion), 587
 dv::FinishDepthEventPacketBuffer (C++ func-
 tion), 588
 dv::FinishDepthFrameBuffer (C++ function), 589
 dv::FinishEventPacketBuffer (C++ function), 589
 dv::FinishFileDataTableBuffer (C++ function),
 599
 dv::FinishFrameBuffer (C++ function), 591
 dv::FinishIMUPacketBuffer (C++ function), 592
 dv::FinishIOHeaderBuffer (C++ function), 600
 dv::FinishLandmarksPacketBuffer (C++ func-
 tion), 594
 dv::FinishPoseBuffer (C++ function), 595
 dv::FinishSizePrefixedBoundingBoxPacket-
 Buffer (C++ function), 587
 dv::FinishSizePrefixedDepthEventPacket-
 Buffer (C++ function), 588
 dv::FinishSizePrefixedDepthFrameBuffer (C++
 function), 589
 dv::FinishSizePrefixedEventPacketBuffer
 (C++ function), 590
 dv::FinishSizePrefixedFileDataTableBuffer
 (C++ function), 599
 dv::FinishSizePrefixedFrameBuffer (C++ func-
 tion), 591
 dv::FinishSizePrefixedIMUPacketBuffer (C++
 function), 592
 dv::FinishSizePrefixedIOHeaderBuffer (C++
 function), 600
 dv::FinishSizePrefixedLandmarksPacket-
 Buffer (C++ function), 594
 dv::FinishSizePrefixedPoseBuffer (C++ func-
 tion), 595
 dv::FinishSizePrefixedTimedKeyPointPacket-
 Buffer (C++ function), 596
 dv::FinishSizePrefixedTriggerPacketBuffer
 (C++ function), 597
 dv::FinishTimedKeyPointPacketBuffer (C++
 function), 596
 dv::FinishTriggerPacketBuffer (C++ function),
 597
 dv::Frame (C++ struct), 357
 dv::Frame::exposure (C++ member), 358
 dv::Frame::Frame (C++ function), 358
 dv::Frame::GetFullyQualifiedName (C++ func-
 tion), 358
 dv::Frame::image (C++ member), 358
 dv::Frame::operator<< (C++ function), 358
 dv::Frame::positionX (C++ member), 358
 dv::Frame::positionY (C++ member), 358
 dv::Frame::source (C++ member), 358
 dv::Frame::TableType (C++ type), 358
 dv::Frame::timestamp (C++ member), 358
 dv::FrameBufferHasIdentifier (C++ function),
 591
 dv::FrameBuilder (C++ struct), 358
 dv::FrameBuilder::add_exposure (C++ function),
 359
 dv::FrameBuilder::add_format (C++ function),
 359
 dv::FrameBuilder::add_pixels (C++ function),
 359
 dv::FrameBuilder::add_positionX (C++ func-
 tion), 359
 dv::FrameBuilder::add_positionY (C++ func-
 tion), 359
 dv::FrameBuilder::add_sizeX (C++ function), 359
 dv::FrameBuilder::add_sizeY (C++ function), 359
 dv::FrameBuilder::add_source (C++ function),
 359
 dv::FrameBuilder::add_timestamp (C++ func-
 tion), 358
 dv::FrameBuilder::add_timestampEndOfExpo-
 sure (C++ function), 359
 dv::FrameBuilder::add_timestampEndOfFrame
 (C++ function), 358

dv::FrameBuilder::add_timestampStartOfExposure (C++ function), 358
 dv::FrameBuilder::add_timestampStartOfFrame (C++ function), 358
 dv::FrameBuilder::fbb_ (C++ member), 359
 dv::FrameBuilder::Finish (C++ function), 359
 dv::FrameBuilder::FrameBuilder (C++ function), 359
 dv::FrameBuilder::operator= (C++ function), 359
 dv::FrameBuilder::start_ (C++ member), 359
 dv::FrameFlatbuffer (C++ struct), 359
 dv::FrameFlatbuffer::exposure (C++ function), 360
 dv::FrameFlatbuffer::format (C++ function), 359
 dv::FrameFlatbuffer::GetFullyQualifiedName (C++ function), 360
 dv::FrameFlatbuffer::identifier (C++ member), 360
 dv::FrameFlatbuffer::MiniReflectTypeTable (C++ function), 360
 dv::FrameFlatbuffer::NativeTableType (C++ type), 359
 dv::FrameFlatbuffer::Pack (C++ function), 360
 dv::FrameFlatbuffer::pixels (C++ function), 360
 dv::FrameFlatbuffer::positionX (C++ function), 360
 dv::FrameFlatbuffer::positionY (C++ function), 360
 dv::FrameFlatbuffer::sizeX (C++ function), 360
 dv::FrameFlatbuffer::sizeY (C++ function), 360
 dv::FrameFlatbuffer::source (C++ function), 360
 dv::FrameFlatbuffer::timestamp (C++ function), 359
 dv::FrameFlatbuffer::timestampEndOfExposure (C++ function), 359
 dv::FrameFlatbuffer::timestampEndOfFrame (C++ function), 359
 dv::FrameFlatbuffer::timestampStartOfExposure (C++ function), 359
 dv::FrameFlatbuffer::timestampStartOfFrame (C++ function), 359
 dv::FrameFlatbuffer::UnPack (C++ function), 360
 dv::FrameFlatbuffer::UnPackTo (C++ function), 360
 dv::FrameFlatbuffer::UnPackToFrom (C++ function), 360
 dv::FrameFlatbuffer::Verify (C++ function), 360
 dv::FrameFormat (C++ enum), 579
 dv::FrameFormat::MAX (C++ enumerator), 580
 dv::FrameFormat::MIN (C++ enumerator), 580
 dv::FrameFormat::OPENCV_8S_C1 (C++ enumerator), 579
 dv::FrameFormat::OPENCV_8S_C2 (C++ enumerator), 579
 dv::FrameFormat::OPENCV_8S_C3 (C++ enumerator), 579
 dv::FrameFormat::OPENCV_8S_C4 (C++ enumerator), 580
 dv::FrameFormat::OPENCV_8U_C1 (C++ enumerator), 579
 dv::FrameFormat::OPENCV_8U_C2 (C++ enumerator), 579
 dv::FrameFormat::OPENCV_8U_C3 (C++ enumerator), 579
 dv::FrameFormat::OPENCV_8U_C4 (C++ enumerator), 580
 dv::FrameFormat::OPENCV_16F_C1 (C++ enumerator), 579
 dv::FrameFormat::OPENCV_16F_C2 (C++ enumerator), 579
 dv::FrameFormat::OPENCV_16F_C3 (C++ enumerator), 580
 dv::FrameFormat::OPENCV_16F_C4 (C++ enumerator), 580
 dv::FrameFormat::OPENCV_16S_C1 (C++ enumerator), 579
 dv::FrameFormat::OPENCV_16S_C2 (C++ enumerator), 579
 dv::FrameFormat::OPENCV_16S_C3 (C++ enumerator), 580
 dv::FrameFormat::OPENCV_16S_C4 (C++ enumerator), 580
 dv::FrameFormat::OPENCV_16U_C1 (C++ enumerator), 579
 dv::FrameFormat::OPENCV_16U_C2 (C++ enumerator), 579
 dv::FrameFormat::OPENCV_16U_C3 (C++ enumerator), 579
 dv::FrameFormat::OPENCV_16U_C4 (C++ enumerator), 580
 dv::FrameFormat::OPENCV_32F_C1 (C++ enumerator), 579
 dv::FrameFormat::OPENCV_32F_C2 (C++ enumerator), 579
 dv::FrameFormat::OPENCV_32F_C3 (C++ enumerator), 580
 dv::FrameFormat::OPENCV_32F_C4 (C++ enumerator), 580
 dv::FrameFormat::OPENCV_32S_C1 (C++ enumerator), 579
 dv::FrameFormat::OPENCV_32S_C2 (C++ enumerator), 579
 dv::FrameFormat::OPENCV_32S_C3 (C++ enumerator), 580
 dv::FrameFormat::OPENCV_32S_C4 (C++ enumerator), 580
 dv::FrameFormat::OPENCV_64F_C1 (C++ enumerator), 579

`dv::FrameFormat::OPENCV_64F_C2` (C++ *enumerator*), 579

`dv::FrameFormat::OPENCV_64F_C3` (C++ *enumerator*), 580

`dv::FrameFormat::OPENCV_64F_C4` (C++ *enumerator*), 580

`dv::FrameFormatTypeTable` (C++ *function*), 590

`dv::FrameIdentifier` (C++ *function*), 591

`dv::FrameSource` (C++ *enum*), 580

`dv::FrameSource::ACCUMULATION` (C++ *enumerator*), 580

`dv::FrameSource::MAX` (C++ *enumerator*), 581

`dv::FrameSource::MIN` (C++ *enumerator*), 581

`dv::FrameSource::MOTION_COMPENSATION` (C++ *enumerator*), 581

`dv::FrameSource::OTHER` (C++ *enumerator*), 581

`dv::FrameSource::RECONSTRUCTION` (C++ *enumerator*), 581

`dv::FrameSource::SENSOR` (C++ *enumerator*), 580

`dv::FrameSource::SYNTHETIC` (C++ *enumerator*), 581

`dv::FrameSource::UNDEFINED` (C++ *enumerator*), 580

`dv::FrameSource::VISUALIZATION` (C++ *enumerator*), 581

`dv::FrameSourceTypeTable` (C++ *function*), 590

`dv::FrameStreamSlicer` (C++ *type*), 577

`dv::FrameTypeTable` (C++ *function*), 590

`dv::fromTimePoint` (C++ *function*), 584

`dv::GetBoundingBoxPacket` (C++ *function*), 587

`dv::GetDepthEventPacket` (C++ *function*), 588

`dv::GetDepthFrame` (C++ *function*), 588

`dv::GetEventPacket` (C++ *function*), 589

`dv::GetFileDataTable` (C++ *function*), 598

`dv::GetFrame` (C++ *function*), 591

`dv::GetIMUPacket` (C++ *function*), 592

`dv::GetIOHeader` (C++ *function*), 599

`dv::GetLandmarksPacket` (C++ *function*), 594

`dv::GetPose` (C++ *function*), 595

`dv::GetSizePrefixedBoundingBoxPacket` (C++ *function*), 587

`dv::GetSizePrefixedDepthEventPacket` (C++ *function*), 588

`dv::GetSizePrefixedDepthFrame` (C++ *function*), 589

`dv::GetSizePrefixedEventPacket` (C++ *function*), 589

`dv::GetSizePrefixedFileDataTable` (C++ *function*), 598

`dv::GetSizePrefixedFrame` (C++ *function*), 591

`dv::GetSizePrefixedIMUPacket` (C++ *function*), 592

`dv::GetSizePrefixedIOHeader` (C++ *function*), 599

`dv::GetSizePrefixedLandmarksPacket` (C++ *function*), 594

`dv::GetSizePrefixedPose` (C++ *function*), 595

`dv::GetSizePrefixedTimedKeyPointPacket` (C++ *function*), 596

`dv::GetSizePrefixedTriggerPacket` (C++ *function*), 597

`dv::GetTimedKeyPointPacket` (C++ *function*), 596

`dv::GetTriggerPacket` (C++ *function*), 597

`dv::imgproc` (C++ *type*), 611

`dv::imgproc::cosineDistance` (C++ *function*), 614, 615

`dv::imgproc::cvMatStepToEigenStride` (C++ *function*), 611

`dv::imgproc::cvMatToEigenMap` (C++ *function*), 611, 612

`dv::imgproc::L1Distance` (C++ *function*), 612, 613

`dv::imgproc::pearsonCorrelation` (C++ *function*), 613, 614

`dv::IMU` (C++ *struct*), 366

`dv::imu` (C++ *type*), 615

`dv::IMU::accelerometerX` (C++ *member*), 367

`dv::IMU::accelerometerY` (C++ *member*), 367

`dv::IMU::accelerometerZ` (C++ *member*), 367

`dv::IMU::getAccelerations` (C++ *function*), 366

`dv::IMU::getAngularVelocities` (C++ *function*), 367

`dv::IMU::GetFullyQualifiedName` (C++ *function*), 367

`dv::IMU::gyroscopeX` (C++ *member*), 367

`dv::IMU::gyroscopeY` (C++ *member*), 367

`dv::IMU::gyroscopeZ` (C++ *member*), 367

`dv::IMU::IMU` (C++ *function*), 366

`dv::IMU::magnetometerX` (C++ *member*), 367

`dv::IMU::magnetometerY` (C++ *member*), 367

`dv::IMU::magnetometerZ` (C++ *member*), 367

`dv::imu::RotationIntegrator` (C++ *class*), 497

`dv::imu::RotationIntegrator::accept` (C++ *function*), 498

`dv::imu::RotationIntegrator::getRotation` (C++ *function*), 498

`dv::imu::RotationIntegrator::getTimestamp` (C++ *function*), 498

`dv::imu::RotationIntegrator::getTransformation` (C++ *function*), 498

`dv::imu::RotationIntegrator::mGyroscopeOffset` (C++ *member*), 499

`dv::imu::RotationIntegrator::mR_S0_S` (C++ *member*), 499

`dv::imu::RotationIntegrator::mSensorToTargetTimeOffset` (C++ *member*), 498

`dv::imu::RotationIntegrator::mT_S0_target` (C++ *member*), 498

`dv::imu::RotationIntegrator::mTimestamp` (C++ *member*), 499

dv::imu::RotationIntegrator::RotationIntegrator (C++ function), 498
 dv::imu::RotationIntegrator::rotationMatrixFromImu (C++ function), 498
 dv::imu::RotationIntegrator::setT_S_target (C++ function), 498
 dv::IMU::TableType (C++ type), 366
 dv::IMU::temperature (C++ member), 367
 dv::IMU::timestamp (C++ member), 367
 dv::IMUBuilder (C++ struct), 367
 dv::IMUBuilder::add_accelerometerX (C++ function), 367
 dv::IMUBuilder::add_accelerometerY (C++ function), 368
 dv::IMUBuilder::add_accelerometerZ (C++ function), 368
 dv::IMUBuilder::add_gyroscopeX (C++ function), 368
 dv::IMUBuilder::add_gyroscopeY (C++ function), 368
 dv::IMUBuilder::add_gyroscopeZ (C++ function), 368
 dv::IMUBuilder::add_magnetometerX (C++ function), 368
 dv::IMUBuilder::add_magnetometerY (C++ function), 368
 dv::IMUBuilder::add_magnetometerZ (C++ function), 368
 dv::IMUBuilder::add_temperature (C++ function), 367
 dv::IMUBuilder::add_timestamp (C++ function), 367
 dv::IMUBuilder::fbb_ (C++ member), 368
 dv::IMUBuilder::Finish (C++ function), 368
 dv::IMUBuilder::IMUBuilder (C++ function), 368
 dv::IMUBuilder::operator= (C++ function), 368
 dv::IMUBuilder::start_ (C++ member), 368
 dv::IMUFlatbuffer (C++ struct), 370
 dv::IMUFlatbuffer::accelerometerX (C++ function), 370
 dv::IMUFlatbuffer::accelerometerY (C++ function), 370
 dv::IMUFlatbuffer::accelerometerZ (C++ function), 370
 dv::IMUFlatbuffer::GetFullyQualifiedName (C++ function), 370
 dv::IMUFlatbuffer::gyroscopeX (C++ function), 370
 dv::IMUFlatbuffer::gyroscopeY (C++ function), 370
 dv::IMUFlatbuffer::gyroscopeZ (C++ function), 370
 dv::IMUFlatbuffer::magnetometerX (C++ function), 370
 dv::IMUFlatbuffer::magnetometerY (C++ function), 370
 dv::IMUFlatbuffer::magnetometerZ (C++ function), 370
 dv::IMUFlatbuffer::MiniReflectTypeTable (C++ function), 370
 dv::IMUFlatbuffer::NativeTableType (C++ type), 370
 dv::IMUFlatbuffer::Pack (C++ function), 371
 dv::IMUFlatbuffer::temperature (C++ function), 370
 dv::IMUFlatbuffer::timestamp (C++ function), 370
 dv::IMUFlatbuffer::UnPack (C++ function), 370
 dv::IMUFlatbuffer::UnPackTo (C++ function), 370
 dv::IMUFlatbuffer::UnPackToFrom (C++ function), 370
 dv::IMUFlatbuffer::Verify (C++ function), 370
 dv::IMUPacket (C++ struct), 371
 dv::IMUPacket::elements (C++ member), 371
 dv::IMUPacket::GetFullyQualifiedName (C++ function), 371
 dv::IMUPacket::IMUPacket (C++ function), 371
 dv::IMUPacket::operator<< (C++ function), 371
 dv::IMUPacket::TableType (C++ type), 371
 dv::IMUPacketBufferHasIdentifier (C++ function), 592
 dv::IMUPacketBuilder (C++ struct), 371
 dv::IMUPacketBuilder::add_elements (C++ function), 371
 dv::IMUPacketBuilder::fbb_ (C++ member), 371
 dv::IMUPacketBuilder::Finish (C++ function), 371
 dv::IMUPacketBuilder::IMUPacketBuilder (C++ function), 371
 dv::IMUPacketBuilder::operator= (C++ function), 371
 dv::IMUPacketBuilder::start_ (C++ member), 371
 dv::IMUPacketFlatbuffer (C++ struct), 371
 dv::IMUPacketFlatbuffer::elements (C++ function), 372
 dv::IMUPacketFlatbuffer::GetFullyQualifiedName (C++ function), 372
 dv::IMUPacketFlatbuffer::identifier (C++ member), 372
 dv::IMUPacketFlatbuffer::MiniReflectTypeTable (C++ function), 372
 dv::IMUPacketFlatbuffer::NativeTableType (C++ type), 372
 dv::IMUPacketFlatbuffer::Pack (C++ function), 372
 dv::IMUPacketFlatbuffer::UnPack (C++ function), 372

dv::IMUPacketFlatbuffer::UnPackTo (C++ *function*), 372
 dv::IMUPacketFlatbuffer::UnPackToFrom (C++ *function*), 372
 dv::IMUPacketFlatbuffer::Verify (C++ *function*), 372
 dv::IMUPacketIdentifier (C++ *function*), 592
 dv::IMUPacketTypeTable (C++ *function*), 592
 dv::IMUStreamSlicer (C++ *type*), 577
 dv::IMUTypeTable (C++ *function*), 592
 dv::IntegerAsEnum (C++ *function*), 585
 dv::io (C++ *type*), 615
 dv::io::camera (C++ *type*), 616
 dv::io::camera::CameraInputBase (C++ *class*), 221
 dv::io::camera::CameraInputBase::Flatten (C++ *enum*), 221
 dv::io::camera::CameraInputBase::Flatten::FLATTEN_ON (C++ *enumerator*), 221
 dv::io::camera::CameraInputBase::Flatten::NONE (C++ *enumerator*), 221
 dv::io::camera::CameraInputBase::Flatten::OFF_ONLY (C++ *enumerator*), 221
 dv::io::camera::CameraInputBase::Flatten::ON_ONLY (C++ *enumerator*), 221
 dv::io::camera::CameraInputBase::getCropArea (C++ *function*), 222
 dv::io::camera::CameraInputBase::getEventSeekTime (C++ *function*), 223
 dv::io::camera::CameraInputBase::getFlatten (C++ *function*), 222
 dv::io::camera::CameraInputBase::getFlipHorizontal (C++ *function*), 221
 dv::io::camera::CameraInputBase::getFlipVertical (C++ *function*), 221
 dv::io::camera::CameraInputBase::getFrameSeekTime (C++ *function*), 223
 dv::io::camera::CameraInputBase::getImuModel (C++ *function*), 222
 dv::io::camera::CameraInputBase::getImuSeekTime (C++ *function*), 223
 dv::io::camera::CameraInputBase::getNextEventBatch (C++ *function*), 223
 dv::io::camera::CameraInputBase::getNextEventPacket (C++ *function*), 222
 dv::io::camera::CameraInputBase::getNextFrame (C++ *function*), 223
 dv::io::camera::CameraInputBase::getNextImuBatch (C++ *function*), 223
 dv::io::camera::CameraInputBase::getNextTriggerBatch (C++ *function*), 223
 dv::io::camera::CameraInputBase::getPixelPitch (C++ *function*), 222
 dv::io::camera::CameraInputBase::getTimeInterval (C++ *function*), 222
 dv::io::camera::CameraInputBase::getTimeStampOffset (C++ *function*), 222
 dv::io::camera::CameraInputBase::getTriggerSeekTime (C++ *function*), 223
 dv::io::camera::CameraInputBase::handleNext (C++ *function*), 224
 dv::io::camera::CameraInputBase::isStreamAvailable (C++ *function*), 223
 dv::io::camera::CameraInputBase::mBuffers (C++ *member*), 224
 dv::io::camera::CameraInputBase::operator<< (C++ *function*), 224
 dv::io::camera::CameraInputBase::readNext (C++ *function*), 223
 dv::io::camera::CameraInputBase::setCropArea (C++ *function*), 222
 dv::io::camera::CameraInputBase::setFlatten (C++ *function*), 222
 dv::io::camera::CameraInputBase::setFlipHorizontal (C++ *function*), 221
 dv::io::camera::CameraInputBase::setFlipVertical (C++ *function*), 221
 dv::io::camera::CameraInputBase::setTimeInterval (C++ *function*), 222
 dv::io::camera::CameraInputBase::SortedPacketBuffers (C++ *class*), 510
 dv::io::camera::CameraInputBase::SortedPacketBuffers::acceptData (C++ *function*), 510
 dv::io::camera::CameraInputBase::SortedPacketBuffers::clearBuffers (C++ *function*), 510
 dv::io::camera::CameraInputBase::SortedPacketBuffers::getEventSeekTime (C++ *function*), 511
 dv::io::camera::CameraInputBase::SortedPacketBuffers::getFrameSeekTime (C++ *function*), 511
 dv::io::camera::CameraInputBase::SortedPacketBuffers::getImuSeekTime (C++ *function*), 511
 dv::io::camera::CameraInputBase::SortedPacketBuffers::getNextTriggerSeekTime (C++ *function*), 511
 dv::io::camera::CameraInputBase::SortedPacketBuffers::mDataLock (C++ *member*), 511
 dv::io::camera::CameraInputBase::SortedPacketBuffers::mEvents (C++ *member*), 511
 dv::io::camera::CameraInputBase::Sorted-

PacketBuffers::mEventStreamSeek
 (C++ member), 511

dv::io::camera::CameraInputBase::Sorted-
 PacketBuffers::mFrames (C++ member),
 511

dv::io::camera::CameraInputBase::Sorted-
 PacketBuffers::mFrameStreamSeek
 (C++ member), 511

dv::io::camera::CameraInputBase::Sorted-
 PacketBuffers::mImu (C++ member),
 511

dv::io::camera::CameraInputBase::Sorted-
 PacketBuffers::mImuStreamSeek (C++
 member), 511

dv::io::camera::CameraInputBase::Sorted-
 PacketBuffers::mPacketCount (C++
 member), 511

dv::io::camera::CameraInputBase::Sort-
 edPacketBuffers::mTriggers (C++
 member), 512

dv::io::camera::CameraInputBase::Sorted-
 PacketBuffers::mTriggerStreamSeek
 (C++ member), 511

dv::io::camera::CameraInputBase::Sorted-
 PacketBuffers::PacketCountType (C++
 type), 511

dv::io::camera::CameraInputBase::Sorted-
 PacketBuffers::popEventPacket (C++
 function), 510

dv::io::camera::CameraInputBase::Sorted-
 PacketBuffers::popFrame (C++ function),
 510

dv::io::camera::CameraInputBase::Sorted-
 PacketBuffers::popImuPacket (C++
 function), 510

dv::io::camera::CameraInputBase::Sorted-
 PacketBuffers::popTriggerPacket
 (C++ function), 510

dv::io::camera::CameraInputBase::Sorted-
 PacketBuffers::readNextPacket (C++
 function), 511

dv::io::camera::CameraModel (C++ enum), 617

dv::io::camera::CameraModel::DAVIS (C++ enu-
 merator), 617

dv::io::camera::CameraModel::DAVIS_GEN2
 (C++ enumerator), 617

dv::io::camera::CameraModel::DVS128 (C++
 enumerator), 617

dv::io::camera::CameraModel::DVXPLOER (C++
 enumerator), 617

dv::io::camera::CameraModel::DVX-
 PLOER_GEN2 (C++ enumerator), 617

dv::io::camera::CameraModel::DVXPLOER_M
 (C++ enumerator), 617

dv::io::camera::CameraModel::DVXPLOER_S
 (C++ enumerator), 617

dv::io::camera::CameraPtr (C++ type), 616

dv::io::camera::DAVIS (C++ class), 232

dv::io::camera::DAVIS::~~DAVIS (C++ function),
 237

dv::io::camera::DAVIS::APS_COLOR_FILTER
 (C++ member), 260

dv::io::camera::DAVIS::APS_END_COLUMN_0
 (C++ member), 261

dv::io::camera::DAVIS::APS_END_ROW_0 (C++
 member), 261

dv::io::camera::DAVIS::APS_EXPOSURE (C++
 member), 261

dv::io::camera::DAVIS::APS_FRAME_INTERVAL
 (C++ member), 262

dv::io::camera::DAVIS::APS_GLOBAL_SHUTTER
 (C++ member), 261

dv::io::camera::DAVIS::APS_GSFDRESET (C++
 member), 262

dv::io::camera::DAVIS::APS_GSPDRESET (C++
 member), 262

dv::io::camera::DAVIS::APS_GSRESETFALL (C++
 member), 262

dv::io::camera::DAVIS::APS_GSTXFALL (C++
 member), 262

dv::io::camera::DAVIS::APS_HAS_GLOBAL_SHUT-
 TER (C++ member), 261

dv::io::camera::DAVIS::APS_ORIENTA-
 TION_INFO (C++ member), 260

dv::io::camera::DAVIS::APS_RSFDSETTLE (C++
 member), 262

dv::io::camera::DAVIS::APS_RUN (C++ member),
 261

dv::io::camera::DAVIS::APS_SIZE_COLUMNS
 (C++ member), 260

dv::io::camera::DAVIS::APS_SIZE_ROWS (C++
 member), 260

dv::io::camera::DAVIS::APS_START_COLUMN_0
 (C++ member), 261

dv::io::camera::DAVIS::APS_START_ROW_0 (C++
 member), 261

dv::io::camera::DAVIS::APS_TRANSFER (C++
 member), 262

dv::io::camera::DAVIS::APS_WAIT_ON_TRANS-
 FER_STALL (C++ member), 261

dv::io::camera::DAVIS::apsHasGlobalShutter
 (C++ member), 249

dv::io::camera::DAVIS::AUTOEXPO-
 SURE_HIGH_BOUNDARY (C++ member),
 270

dv::io::camera::DAVIS::AUTOEXPOSURE_HIS-
 TOGRAM_BINS_HIGH (C++ member), 270

dv::io::camera::DAVIS::AUTOEXPOSURE_HIS-

TOGRAM_BINS_LOW (C++ member), 270

dv::io::camera::DAVIS::AUTOEXPOSURE_HIS-
TOGRAM_MSV_SIZE (C++ member), 270

dv::io::camera::DAVIS::AUTOEXPOSURE_HIS-
TOGRAM_RANGE (C++ member), 270

dv::io::camera::DAVIS::AUTOEXPOSURE_HIS-
TOGRAM_RANGES (C++ member), 270

dv::io::camera::DAVIS::AUTOEXPOSURE_HIS-
TOGRAM_SIZE (C++ member), 270

dv::io::camera::DAVIS::AUTOEXPO-
SURE_LOW_BOUNDARY (C++ member), 270

dv::io::camera::DAVIS::AUTOEXPO-
SURE_MSV_CORRECTION (C++ member), 270

dv::io::camera::DAVIS::AUTOEXPOSURE_UNDER-
OVER_CORRECTION (C++ member), 270

dv::io::camera::DAVIS::AUTOEXPOSURE_UNDER-
OVER_FRAC (C++ member), 270

dv::io::camera::DAVIS::AutoExposureCall-
back (C++ type), 236

dv::io::camera::DAVIS::caerBiasCoarseFine-
Generate (C++ function), 253

dv::io::camera::DAVIS::caerBiasCoarse-
FineParse (C++ function), 253

dv::io::camera::DAVIS::caerBiasShifted-
SourceGenerate (C++ function), 253

dv::io::camera::DAVIS::caerBiasShifted-
SourceParse (C++ function), 254

dv::io::camera::DAVIS::caerBiasVDACGener-
ate (C++ function), 253

dv::io::camera::DAVIS::caerBiasVDACParse
(C++ function), 253

dv::io::camera::DAVIS::CDAVIS_BIAS_IFRE-
FRBN (C++ member), 269

dv::io::camera::DAVIS::CDAVIS_BIAS_IFTHRBN
(C++ member), 269

dv::io::camera::DAVIS::CDAVIS_BIAS_SSN (C++
member), 269

dv::io::camera::DAVIS::CDAVIS_BIAS_SSP (C++
member), 269

dv::io::camera::DAVIS::CDAVIS_CHIP_ADJUS-
TOVG1LO (C++ member), 268

dv::io::camera::DAVIS::CDAVIS_CHIP_ADJUS-
TOVG2LO (C++ member), 268

dv::io::camera::DAVIS::CDAVIS_CHIP_AD-
JUSTTX2OVG2HI (C++ member), 269

dv::io::camera::DAVIS::CDAVIS_CHIP_AERNAROW
(C++ member), 268

dv::io::camera::DAVIS::CDAVIS_CHIP_ANALOG-
MUX0 (C++ member), 268

dv::io::camera::DAVIS::CDAVIS_CHIP_ANALOG-
MUX1 (C++ member), 268

dv::io::camera::DAVIS::CDAVIS_CHIP_ANALOG-
MUX2 (C++ member), 268

dv::io::camera::DAVIS::CDAVIS_CHIP_BIAS-
MUX0 (C++ member), 268

dv::io::camera::DAVIS::CDAVIS_CHIP_DIGI-
TALMUX0 (C++ member), 268

dv::io::camera::DAVIS::CDAVIS_CHIP_DIGI-
TALMUX1 (C++ member), 268

dv::io::camera::DAVIS::CDAVIS_CHIP_DIGI-
TALMUX2 (C++ member), 268

dv::io::camera::DAVIS::CDAVIS_CHIP_DIGI-
TALMUX3 (C++ member), 268

dv::io::camera::DAVIS::CDAVIS_CHIP_RESET-
CALIBNEURON (C++ member), 268

dv::io::camera::DAVIS::CDAVIS_CHIP_RE-
SETTESTPIXEL (C++ member), 268

dv::io::camera::DAVIS::CDAVIS_CHIP_SE-
LECTGRAYCOUNTER (C++ member), 268

dv::io::camera::DAVIS::CDAVIS_CHIP_TESTADC
(C++ member), 268

dv::io::camera::DAVIS::CDAVIS_CHIP_TYPEN-
CALIBNEURON (C++ member), 268

dv::io::camera::DAVIS::CDAVIS_CHIP_USEAOUT
(C++ member), 268

dv::io::camera::DAVIS::CDavisBiasCF (C++
enum), 235

dv::io::camera::DAVIS::CDavisBiasCF::ADC-
Comparator (C++ enumerator), 236

dv::io::camera::DAVIS::CDavisBiasCF::AER-
PullDown (C++ enumerator), 236

dv::io::camera::DAVIS::CDavisBiasCF::AER-
PullUpX (C++ enumerator), 236

dv::io::camera::DAVIS::CDavisBiasCF::AER-
PullUpY (C++ enumerator), 236

dv::io::camera::DAVIS::CDavisBiasCF::Ap-
sReadoutSourceFollower (C++ enumera-
tor), 236

dv::io::camera::DAVIS::CDavisBiasCF::Ar-
rayBiasBuffer (C++ enumerator), 236

dv::io::camera::DAVIS::CDavisBiasCF::Ar-
rayLogicBuffer (C++ enumerator), 236

dv::io::camera::DAVIS::CDavisBiasCF::Bias-
Buffer (C++ enumerator), 236

dv::io::camera::DAVIS::CDavisBi-
asCF::DACBuffer (C++ enumerator), 236

dv::io::camera::DAVIS::CDavisBiasCF::Diff
(C++ enumerator), 235

dv::io::camera::DAVIS::CDavisBiasCF::Fall-
Time (C++ enumerator), 236

dv::io::camera::DAVIS::CDavisBi-
asCF::LCOLTimeout (C++ enumerator), 236

dv::io::camera::DAVIS::CDavisBiasCF::Lo-
calBuffer (C++ enumerator), 235

dv::io::camera::DAVIS::CDavisBiasCF::Off

(C++ enumerator), 235

dv::io::camera::DAVIS::CDavisBiasCF::On (C++ enumerator), 235

dv::io::camera::DAVIS::CDavisBiasCF::PadFollower (C++ enumerator), 235

dv::io::camera::DAVIS::CDavisBiasCF::Photoreceptor (C++ enumerator), 235

dv::io::camera::DAVIS::CDavisBiasCF::PhotoreceptorSourceFollower (C++ enumerator), 236

dv::io::camera::DAVIS::CDavisBiasCF::PixelInverter (C++ enumerator), 235

dv::io::camera::DAVIS::CDavisBiasCF::ReadoutBuffer (C++ enumerator), 236

dv::io::camera::DAVIS::CDavisBiasCF::Refractory (C++ enumerator), 236

dv::io::camera::DAVIS::CDavisBiasCF::RiseTime (C++ enumerator), 236

dv::io::camera::DAVIS::CDavisBiasVDAC (C++ enum), 235

dv::io::camera::DAVIS::CDavisBiasVDAC::ADCReferenceHigh (C++ enumerator), 235

dv::io::camera::DAVIS::CDavisBiasVDAC::ADCReferenceLow (C++ enumerator), 235

dv::io::camera::DAVIS::CDavisBiasVDAC::ADCTestVoltage (C++ enumerator), 235

dv::io::camera::DAVIS::CDavisBiasVDAC::ApsCascode (C++ enumerator), 235

dv::io::camera::DAVIS::CDavisBiasVDAC::Gnd07 (C++ enumerator), 235

dv::io::camera::DAVIS::CDavisBiasVDAC::OVG1Low (C++ enumerator), 235

dv::io::camera::DAVIS::CDavisBiasVDAC::OVG2Low (C++ enumerator), 235

dv::io::camera::DAVIS::CDavisBiasVDAC::TX2OVG2High (C++ enumerator), 235

dv::io::camera::DAVIS::CoarseFineBias (C++ struct), 226

dv::io::camera::DAVIS::CoarseFineBias::CoarseFineBias (C++ function), 226

dv::io::camera::DAVIS::CoarseFineBias::coarseValue (C++ member), 226

dv::io::camera::DAVIS::CoarseFineBias::currentLevel (C++ member), 227

dv::io::camera::DAVIS::CoarseFineBias::enabled (C++ member), 226

dv::io::camera::DAVIS::CoarseFineBias::fineValue (C++ member), 226

dv::io::camera::DAVIS::CoarseFineBias::sex (C++ member), 226

dv::io::camera::DAVIS::CoarseFineBias::CoarseFineBias::type (C++ member), 227

dv::io::camera::DAVIS::CoarseFineBiasCurrentLevel (C++ enum), 250

dv::io::camera::DAVIS::CoarseFineBiasCurrentLevel::LOW (C++ enumerator), 251

dv::io::camera::DAVIS::CoarseFineBiasCurrentLevel::NORMAL (C++ enumerator), 251

dv::io::camera::DAVIS::CoarseFineBiasSex (C++ enum), 250

dv::io::camera::DAVIS::CoarseFineBiasSex::N_TYPE (C++ enumerator), 250

dv::io::camera::DAVIS::CoarseFineBiasSex::P_TYPE (C++ enumerator), 250

dv::io::camera::DAVIS::CoarseFineBiasType (C++ enum), 250

dv::io::camera::DAVIS::CoarseFineBiasType::CASCODE (C++ enumerator), 250

dv::io::camera::DAVIS::CoarseFineBiasType::NORMAL (C++ enumerator), 250

dv::io::camera::DAVIS::COMPATIBLE_CAMERA (C++ member), 270

dv::io::camera::DAVIS::computeAutomaticExposure (C++ function), 252

dv::io::camera::DAVIS::dataParserCallback (C++ function), 251

dv::io::camera::DAVIS::DAVIS (C++ function), 236, 237

dv::io::camera::DAVIS::DAVIS240_BIAS_IFRBN (C++ member), 269

dv::io::camera::DAVIS::DAVIS240_BIAS_IFTHRBN (C++ member), 269

dv::io::camera::DAVIS::DAVIS240_BIAS_SSN (C++ member), 269

dv::io::camera::DAVIS::DAVIS240_BIAS_SSP (C++ member), 269

dv::io::camera::DAVIS::DAVIS240_CHIP_AERNAROW (C++ member), 267

dv::io::camera::DAVIS::DAVIS240_CHIP_ANALOG_MUX0 (C++ member), 266

dv::io::camera::DAVIS::DAVIS240_CHIP_ANALOG_MUX1 (C++ member), 266

dv::io::camera::DAVIS::DAVIS240_CHIP_ANALOG_MUX2 (C++ member), 266

dv::io::camera::DAVIS::DAVIS240_CHIP_BIAS_MUX0 (C++ member), 266

dv::io::camera::DAVIS::DAVIS240_CHIP_DIGITALMUX0 (C++ member), 266

dv::io::camera::DAVIS::DAVIS240_CHIP_DIGITALMUX1 (C++ member), 266

dv::io::camera::DAVIS::DAVIS240_CHIP_DIGI-

TALMUX2 (C++ member), 266
 dv::io::camera::DAVIS::DAVIS240_CHIP_DIGI-
 TALMUX3 (C++ member), 266
 dv::io::camera::DAVIS::DAVIS240_CHIP_GLOBAL_SHUT-
 TER (C++ member), 267
 dv::io::camera::DAVIS::DAVIS240_CHIP_RE-
 SETCALIBNEURON (C++ member), 267
 dv::io::camera::DAVIS::DAVIS240_CHIP_RE-
 SETTESTPIXEL (C++ member), 267
 dv::io::camera::DAVIS::DAVIS240_CHIP_SPE-
 CIALPIXELCONTROL (C++ member), 267
 dv::io::camera::DAVIS::DAVIS240_CHIP_TYPE-
 NCALIBNEURON (C++ member), 267
 dv::io::camera::DAVIS::DAVIS240_CHIP_USE-
 AOUT (C++ member), 267
 dv::io::camera::DAVIS::Davis240BiasCF (C++
 enum), 233
 dv::io::camera::DAVIS::Davis240Bi-
 asCF::AERPulldown (C++ enumerator),
 233
 dv::io::camera::DAVIS::Davis240Bi-
 asCF::AERPulldupX (C++ enumerator),
 233
 dv::io::camera::DAVIS::Davis240Bi-
 asCF::AERPulldupY (C++ enumerator),
 233
 dv::io::camera::DAVIS::Davis240BiasCF::Ap-
 sCascode (C++ enumerator), 233
 dv::io::camera::DAVIS::Davis240Bi-
 asCF::ApsOverflowLevel (C++ enu-
 merator), 233
 dv::io::camera::DAVIS::Davis240BiasCF::Ap-
 sReadoutSourceFollower (C++ enumera-
 tor), 233
 dv::io::camera::DAVIS::Davis240BiasCF::Bi-
 asBuffer (C++ enumerator), 233
 dv::io::camera::DAVIS::Davis240Bi-
 asCF::Diff (C++ enumerator), 233
 dv::io::camera::DAVIS::Davis240Bi-
 asCF::DiffCascode (C++ enumerator),
 233
 dv::io::camera::DAVIS::Davis240Bi-
 asCF::LCOlTimeout (C++ enumerator),
 233
 dv::io::camera::DAVIS::Davis240BiasCF::Lo-
 calBuffer (C++ enumerator), 233
 dv::io::camera::DAVIS::Davis240BiasCF::Off
 (C++ enumerator), 233
 dv::io::camera::DAVIS::Davis240BiasCF::On
 (C++ enumerator), 233
 dv::io::camera::DAVIS::Davis240Bi-
 asCF::PadFollower (C++ enumerator),
 233
 dv::io::camera::DAVIS::Davis240Bi-
 asCF::Photoreceptor (C++ enumerator),
 233
 dv::io::camera::DAVIS::Davis240Bi-
 asCF::PhotoreceptorSourceFollower
 (C++ enumerator), 233
 dv::io::camera::DAVIS::Davis240Bi-
 asCF::PixelInverter (C++ enumerator),
 233
 dv::io::camera::DAVIS::Davis240BiasCF::Re-
 fractory (C++ enumerator), 233
 dv::io::camera::DAVIS::DAVIS346_BIAS_IFRE-
 FRBN (C++ member), 269
 dv::io::camera::DAVIS::DAVIS346_BIAS_IFTHRBN
 (C++ member), 269
 dv::io::camera::DAVIS::DAVIS346_BIAS_SSN
 (C++ member), 269
 dv::io::camera::DAVIS::DAVIS346_BIAS_SSP
 (C++ member), 269
 dv::io::camera::DAVIS::DAVIS346_CHIP_AERNAROW
 (C++ member), 267
 dv::io::camera::DAVIS::DAVIS346_CHIP_ANALOG-
 MUX0 (C++ member), 267
 dv::io::camera::DAVIS::DAVIS346_CHIP_ANALOG-
 MUX1 (C++ member), 267
 dv::io::camera::DAVIS::DAVIS346_CHIP_ANALOG-
 MUX2 (C++ member), 267
 dv::io::camera::DAVIS::DAVIS346_CHIP_BIAS-
 MUX0 (C++ member), 267
 dv::io::camera::DAVIS::DAVIS346_CHIP_DIGI-
 TALMUX0 (C++ member), 267
 dv::io::camera::DAVIS::DAVIS346_CHIP_DIGI-
 TALMUX1 (C++ member), 267
 dv::io::camera::DAVIS::DAVIS346_CHIP_DIGI-
 TALMUX2 (C++ member), 267
 dv::io::camera::DAVIS::DAVIS346_CHIP_DIGI-
 TALMUX3 (C++ member), 267
 dv::io::camera::DAVIS::DAVIS346_CHIP_GLOBAL_SHUT-
 TER (C++ member), 268
 dv::io::camera::DAVIS::DAVIS346_CHIP_RE-
 SETCALIBNEURON (C++ member), 267
 dv::io::camera::DAVIS::DAVIS346_CHIP_RE-
 SETTESTPIXEL (C++ member), 267
 dv::io::camera::DAVIS::DAVIS346_CHIP_SE-
 LECTGRAYCOUNTER (C++ member), 268
 dv::io::camera::DAVIS::DAVIS346_CHIP_TES-
 TADC (C++ member), 268
 dv::io::camera::DAVIS::DAVIS346_CHIP_TYPE-
 NCALIBNEURON (C++ member), 267
 dv::io::camera::DAVIS::DAVIS346_CHIP_USE-
 AOUT (C++ member), 267
 dv::io::camera::DAVIS::Davis346BiasCF (C++
 enum), 234
 dv::io::camera::DAVIS::Davis346BiasCF::AD-
 CComparator (C++ enumerator), 234

dv::io::camera::DAVIS::Davis346Bi-
 asCF::AERPullDown (C++ *enumerator*),
 234
 dv::io::camera::DAVIS::Davis346Bi-
 asCF::AERPullUpX (C++ *enumerator*),
 235
 dv::io::camera::DAVIS::Davis346Bi-
 asCF::AERPullUpY (C++ *enumerator*),
 235
 dv::io::camera::DAVIS::Davis346BiasCF::Ap-
 sReadoutSourceFollower (C++ *enumera-*
 tor), 234
 dv::io::camera::DAVIS::Davis346BiasCF::Bi-
 asBuffer (C++ *enumerator*), 235
 dv::io::camera::DAVIS::Davis346Bi-
 asCF::COLSelectLow (C++ *enumerator*),
 234
 dv::io::camera::DAVIS::Davis346Bi-
 asCF::DACBuffer (C++ *enumerator*),
 234
 dv::io::camera::DAVIS::Davis346Bi-
 asCF::Diff (C++ *enumerator*), 234
 dv::io::camera::DAVIS::Davis346Bi-
 asCF::LCOTimeout (C++ *enumerator*),
 234
 dv::io::camera::DAVIS::Davis346BiasCF::Lo-
 calBuffer (C++ *enumerator*), 234
 dv::io::camera::DAVIS::Davis346BiasCF::Off
 (C++ *enumerator*), 234
 dv::io::camera::DAVIS::Davis346BiasCF::On
 (C++ *enumerator*), 234
 dv::io::camera::DAVIS::Davis346Bi-
 asCF::PadFollower (C++ *enumerator*),
 234
 dv::io::camera::DAVIS::Davis346Bi-
 asCF::Photoreceptor (C++ *enumerator*),
 234
 dv::io::camera::DAVIS::Davis346Bi-
 asCF::PhotoreceptorSourceFollower
 (C++ *enumerator*), 234
 dv::io::camera::DAVIS::Davis346Bi-
 asCF::PixelInverter (C++ *enumerator*),
 234
 dv::io::camera::DAVIS::Davis346Bi-
 asCF::ReadoutBuffer (C++ *enumerator*),
 234
 dv::io::camera::DAVIS::Davis346BiasCF::Re-
 fractory (C++ *enumerator*), 234
 dv::io::camera::DAVIS::Davis346BiasVDAC
 (C++ *enum*), 233
 dv::io::camera::DAVIS::Davis346BiasV-
 DAC::ADCReferenceHigh (C++ *enumera-*
 tor), 234
 dv::io::camera::DAVIS::Davis346BiasV-
 DAC::ADCReferenceLow (C++ *enumerator*),
 234
 dv::io::camera::DAVIS::Davis346BiasV-
 DAC::ADCTestVoltage (C++ *enumerator*),
 234
 dv::io::camera::DAVIS::Davis346BiasV-
 DAC::ApsCascode (C++ *enumerator*),
 234
 dv::io::camera::DAVIS::Davis346BiasV-
 DAC::ApsOverflowLevel (C++ *enumera-*
 tor), 233
 dv::io::camera::DAVIS::DVS_EXTER-
 NAL_AER_CONTROL (C++ *member*), 256
 dv::io::camera::DAVIS::DVS_FILTER_BACK-
 GROUND_ACTIVITY (C++ *member*), 258
 dv::io::camera::DAVIS::DVS_FILTER_BACK-
 GROUND_ACTIVITY_TIME (C++ *member*),
 258
 dv::io::camera::DAVIS::DVS_FIL-
 TER_PIXEL_0_COLUMN (C++ *member*),
 256
 dv::io::camera::DAVIS::DVS_FIL-
 TER_PIXEL_0_ROW (C++ *member*), 256
 dv::io::camera::DAVIS::DVS_FIL-
 TER_PIXEL_1_COLUMN (C++ *member*),
 257
 dv::io::camera::DAVIS::DVS_FIL-
 TER_PIXEL_1_ROW (C++ *member*), 257
 dv::io::camera::DAVIS::DVS_FIL-
 TER_PIXEL_2_COLUMN (C++ *member*),
 257
 dv::io::camera::DAVIS::DVS_FIL-
 TER_PIXEL_2_ROW (C++ *member*), 257
 dv::io::camera::DAVIS::DVS_FIL-
 TER_PIXEL_3_COLUMN (C++ *member*),
 257
 dv::io::camera::DAVIS::DVS_FIL-
 TER_PIXEL_3_ROW (C++ *member*), 257
 dv::io::camera::DAVIS::DVS_FIL-
 TER_PIXEL_4_COLUMN (C++ *member*),
 257
 dv::io::camera::DAVIS::DVS_FIL-
 TER_PIXEL_4_ROW (C++ *member*), 257
 dv::io::camera::DAVIS::DVS_FIL-
 TER_PIXEL_5_COLUMN (C++ *member*),
 257
 dv::io::camera::DAVIS::DVS_FIL-
 TER_PIXEL_5_ROW (C++ *member*), 257
 dv::io::camera::DAVIS::DVS_FIL-
 TER_PIXEL_6_COLUMN (C++ *member*),
 257
 dv::io::camera::DAVIS::DVS_FIL-
 TER_PIXEL_6_ROW (C++ *member*), 257
 dv::io::camera::DAVIS::DVS_FIL-

TER_PIXEL_7_COLUMN (C++ member), 258	TICS_EVENTS_DROPPED (C++ member), 260
dv::io::camera::DAVIS::DVS_FILTER_PIXEL_7_ROW (C++ member), 258	dv::io::camera::DAVIS::DVS_STATISTICS-TICS_EVENTS_ROW (C++ member), 259
dv::io::camera::DAVIS::DVS_FILTER_POLARITY_FLATTEN (C++ member), 259	dv::io::camera::DAVIS::DVS_STATISTICS_FILTERED_BACKGROUND_ACTIVITY (C++ member), 260
dv::io::camera::DAVIS::DVS_FILTER_POLARITY_SUPPRESS (C++ member), 259	dv::io::camera::DAVIS::DVS_STATISTICS_FILTERED_PIXELS (C++ member), 260
dv::io::camera::DAVIS::DVS_FILTER_POLARITY_SUPPRESS_TYPE (C++ member), 259	dv::io::camera::DAVIS::DVS_STATISTICS_FILTERED_REFRACTORY_PERIOD (C++ member), 260
dv::io::camera::DAVIS::DVS_FILTER_REFRACTORY_PERIOD (C++ member), 258	dv::io::camera::DAVIS::DVS_WAIT_ON_TRANSFER_STALL (C++ member), 256
dv::io::camera::DAVIS::DVS_FILTER_REFRACTORY_PERIOD_TIME (C++ member), 258	dv::io::camera::DAVIS::dvsHasBackgroundActivityFilter (C++ member), 249
dv::io::camera::DAVIS::DVS_FILTER_ROI_END_COLUMN (C++ member), 259	dv::io::camera::DAVIS::dvsHasPixelFilter (C++ member), 249
dv::io::camera::DAVIS::DVS_FILTER_ROI_END_ROW (C++ member), 259	dv::io::camera::DAVIS::dvsHasPolarityFilter (C++ member), 249
dv::io::camera::DAVIS::DVS_FILTER_ROI_START_COLUMN (C++ member), 258	dv::io::camera::DAVIS::dvsHasROIFilter (C++ member), 249
dv::io::camera::DAVIS::DVS_FILTER_ROI_START_ROW (C++ member), 258	dv::io::camera::DAVIS::dvsHasSkipFilter (C++ member), 249
dv::io::camera::DAVIS::DVS_FILTER_SKIP_EVENTS (C++ member), 259	dv::io::camera::DAVIS::dvsHasStatistics (C++ member), 249
dv::io::camera::DAVIS::DVS_FILTER_SKIP_EVENTS_EVERY (C++ member), 259	dv::io::camera::DAVIS::EXPOSURE_MAX (C++ member), 270
dv::io::camera::DAVIS::DVS_HAS_BACKGROUND_ACTIVITY_FILTER (C++ member), 258	dv::io::camera::DAVIS::EXT_INPUT_TIME_MAX (C++ member), 270
dv::io::camera::DAVIS::DVS_HAS_PIXEL_FILTER (C++ member), 256	dv::io::camera::DAVIS::EXTINPUT_DETECT_FALLING_EDGES (C++ member), 264
dv::io::camera::DAVIS::DVS_HAS_POLARITY_FILTER (C++ member), 259	dv::io::camera::DAVIS::EXTINPUT_DETECT_PULSE_LENGTH (C++ member), 264
dv::io::camera::DAVIS::DVS_HAS_ROI_FILTER (C++ member), 258	dv::io::camera::DAVIS::EXTINPUT_DETECT_PULSE_POLARITY (C++ member), 264
dv::io::camera::DAVIS::DVS_HAS_SKIP_FILTER (C++ member), 259	dv::io::camera::DAVIS::EXTINPUT_DETECT_PULSES (C++ member), 264
dv::io::camera::DAVIS::DVS_HAS_STATISTICS (C++ member), 259	dv::io::camera::DAVIS::EXTINPUT_DETECT_RISING_EDGES (C++ member), 264
dv::io::camera::DAVIS::DVS_ORIENTATION_INFO (C++ member), 256	dv::io::camera::DAVIS::EXTINPUT_GENERATE_INJECT_ON_FALLING_EDGE (C++ member), 265
dv::io::camera::DAVIS::DVS_RUN (C++ member), 256	dv::io::camera::DAVIS::EXTINPUT_GENERATE_INJECT_ON_RISING_EDGE (C++ member), 265
dv::io::camera::DAVIS::DVS_SIZE_COLUMNS (C++ member), 256	dv::io::camera::DAVIS::EXTINPUT_GENERATE_PULSE_INTERVAL (C++ member), 264
dv::io::camera::DAVIS::DVS_SIZE_ROWS (C++ member), 256	dv::io::camera::DAVIS::EXTINPUT_GENERATE_PULSE_LENGTH (C++ member), 265
dv::io::camera::DAVIS::DVS_STATISTICS_EVENTS_COLUMN (C++ member), 260	dv::io::camera::DAVIS::EXTINPUT_GENERATE-
dv::io::camera::DAVIS::DVS_STATISTICS-	

ATE_PULSE_POLARITY (C++ member), 264

dv::io::camera::DAVIS::EXTINPUT_HAS_GENERATOR (C++ member), 264

dv::io::camera::DAVIS::EXTINPUT_RUN_DETECTOR (C++ member), 264

dv::io::camera::DAVIS::EXTINPUT_RUN_GENERATOR (C++ member), 264

dv::io::camera::DAVIS::extInputHasGenerator (C++ member), 249

dv::io::camera::DAVIS::findDevices (C++ function), 250

dv::io::camera::DAVIS::FRAME_INTERVAL_MAX (C++ member), 270

dv::io::camera::DAVIS::FX2_FIRMWARE_REQUIRED_VERSION (C++ member), 269

dv::io::camera::DAVIS::FX2_LOGIC_MINIMUM_PATCH (C++ member), 269

dv::io::camera::DAVIS::FX2_LOGIC_REQUIRED_VERSION (C++ member), 269

dv::io::camera::DAVIS::FX3_FIRMWARE_REQUIRED_VERSION (C++ member), 269

dv::io::camera::DAVIS::FX3_LOGIC_MINIMUM_PATCH (C++ member), 269

dv::io::camera::DAVIS::FX3_LOGIC_REQUIRED_VERSION (C++ member), 269

dv::io::camera::DAVIS::getAutoExposure (C++ function), 242

dv::io::camera::DAVIS::getBackgroundActivityFilter (C++ function), 240

dv::io::camera::DAVIS::getCameraName (C++ function), 237

dv::io::camera::DAVIS::getCDavisAdjustOVG1Low (C++ function), 248

dv::io::camera::DAVIS::getCDavisAdjustOVG2Low (C++ function), 248

dv::io::camera::DAVIS::getCDavisAdjustTX2OVGHigh (C++ function), 248

dv::io::camera::DAVIS::getCDavisBiasCoarseFine (C++ function), 248

dv::io::camera::DAVIS::getCDavisBiasVoltage (C++ function), 248

dv::io::camera::DAVIS::getColorMode (C++ function), 242

dv::io::camera::DAVIS::getCropArea (C++ function), 241

dv::io::camera::DAVIS::getCropAreaEvents (C++ function), 241

dv::io::camera::DAVIS::getCropAreaFrames (C++ function), 243

dv::io::camera::DAVIS::getDavis240BiasCoarseFine (C++ function), 247

dv::io::camera::DAVIS::getDavis346BiasCoarseFine (C++ function), 247

dv::io::camera::DAVIS::getDavis346BiasVoltage (C++ function), 247

dv::io::camera::DAVIS::getDetectorFallingEdges (C++ function), 245

dv::io::camera::DAVIS::getDetectorRisingEdges (C++ function), 245

dv::io::camera::DAVIS::getEventColor (C++ function), 239

dv::io::camera::DAVIS::getEventResolution (C++ function), 237

dv::io::camera::DAVIS::getExposureDuration (C++ function), 242

dv::io::camera::DAVIS::getExternalAERControl (C++ function), 240

dv::io::camera::DAVIS::getFlatten (C++ function), 241

dv::io::camera::DAVIS::getFlipHorizontal (C++ function), 239

dv::io::camera::DAVIS::getFlipHorizontalEvents (C++ function), 240

dv::io::camera::DAVIS::getFlipHorizontalFrames (C++ function), 243

dv::io::camera::DAVIS::getFlipVertical (C++ function), 240

dv::io::camera::DAVIS::getFlipVerticalEvents (C++ function), 240

dv::io::camera::DAVIS::getFlipVerticalFrames (C++ function), 243

dv::io::camera::DAVIS::getFrameInterval (C++ function), 243

dv::io::camera::DAVIS::getFrameResolution (C++ function), 237

dv::io::camera::DAVIS::getGeneratorHighTime (C++ function), 246

dv::io::camera::DAVIS::getGeneratorHighTimeInternal (C++ function), 250

dv::io::camera::DAVIS::getGeneratorInjectTriggerOnFallingEdge (C++ function), 246

dv::io::camera::DAVIS::getGeneratorInjectTriggerOnRisingEdge (C++ function), 246

dv::io::camera::DAVIS::getGeneratorLowTime (C++ function), 246

dv::io::camera::DAVIS::getGeneratorLowTimeInternal (C++ function), 250

dv::io::camera::DAVIS::getImuAccelRange (C++ function), 245

dv::io::camera::DAVIS::getIMUFlipX (C++ function), 244

dv::io::camera::DAVIS::getIMUFlipY (C++ function), 244

dv::io::camera::DAVIS::getIMUFlipZ (C++ function), 244

dv::io::camera::DAVIS::getImuGyroRange (C++ function), 245

dv::io::camera::DAVIS::getImuModel (C++ function), 237

dv::io::camera::DAVIS::getLogicPatchLevel (C++ function), 237

dv::io::camera::DAVIS::getLogicVersion (C++ function), 237

dv::io::camera::DAVIS::getPixelArrangementEvents (C++ function), 239

dv::io::camera::DAVIS::getPixelArrangementFrames (C++ function), 239

dv::io::camera::DAVIS::getPixelPitch (C++ function), 238

dv::io::camera::DAVIS::getRefractoryPeriodFilter (C++ function), 241

dv::io::camera::DAVIS::getSensorModel (C++ function), 237

dv::io::camera::DAVIS::getSkipFilter (C++ function), 241

dv::io::camera::DAVIS::getTimeInterval (C++ function), 239

dv::io::camera::DAVIS::getTimestampOffset (C++ function), 239

dv::io::camera::DAVIS::getUSBEarlyPacketDelay (C++ function), 248

dv::io::camera::DAVIS::IMU_ACCEL_DLPF (C++ member), 263

dv::io::camera::DAVIS::IMU_ACCEL_FULL_SCALE (C++ member), 263

dv::io::camera::DAVIS::IMU_GYRO_DLPF (C++ member), 263

dv::io::camera::DAVIS::IMU_GYRO_FULL_SCALE (C++ member), 263

dv::io::camera::DAVIS::IMU_ORIENTATION_INFO (C++ member), 262

dv::io::camera::DAVIS::IMU_RUN_ACCELEROMETER (C++ member), 262

dv::io::camera::DAVIS::IMU_RUN_GYROSCOPE (C++ member), 262

dv::io::camera::DAVIS::IMU_RUN_TEMPERATURE (C++ member), 263

dv::io::camera::DAVIS::IMU_SAMPLE_RATE_DIVIDER (C++ member), 263

dv::io::camera::DAVIS::IMU_TYPE (C++ member), 262

dv::io::camera::DAVIS::isDetectorRunning (C++ function), 245

dv::io::camera::DAVIS::isEventsRunning (C++ function), 239

dv::io::camera::DAVIS::isEventStreamAvailable (C++ function), 238

dv::io::camera::DAVIS::isFramesRunning (C++ function), 242

dv::io::camera::DAVIS::isFrameStreamAvailable (C++ function), 238

dv::io::camera::DAVIS::isGeneratorRunning (C++ function), 246

dv::io::camera::DAVIS::isImuRunningAccelerometer (C++ function), 243

dv::io::camera::DAVIS::isImuRunningGyroscope (C++ function), 244

dv::io::camera::DAVIS::isImuRunningTemperature (C++ function), 244

dv::io::camera::DAVIS::isImuStreamAvailable (C++ function), 238

dv::io::camera::DAVIS::isMaster (C++ function), 238

dv::io::camera::DAVIS::isRunning (C++ function), 238

dv::io::camera::DAVIS::isRunningAny (C++ function), 238

dv::io::camera::DAVIS::isTriggerStreamAvailable (C++ function), 238

dv::io::camera::DAVIS::mADCClockActual (C++ member), 252

dv::io::camera::DAVIS::mAutoExposure (C++ member), 253

dv::io::camera::DAVIS::mAutoExposureCallback (C++ member), 252

dv::io::camera::DAVIS::mAutoExposureEnabled (C++ member), 252

dv::io::camera::DAVIS::mAutoExposureLast (C++ member), 253

dv::io::camera::DAVIS::mCallbackConfigLock (C++ member), 253

dv::io::camera::DAVIS::mCoarseFineBiases (C++ member), 252

dv::io::camera::DAVIS::mConfigLock (C++ member), 252

dv::io::camera::DAVIS::mCropAreaEvents (C++ member), 252

dv::io::camera::DAVIS::mCropAreaFrames (C++ member), 252

dv::io::camera::DAVIS::mDvsColorFilter (C++ member), 252

dv::io::camera::DAVIS::mEventResolution (C++ member), 252

dv::io::camera::DAVIS::mFrameResolution (C++ member), 252

dv::io::camera::DAVIS::mImuModel (C++ member), 252

dv::io::camera::DAVIS::mInfo (C++ member), 253

dv::io::camera::DAVIS::mIsRunning (C++ member), 253

dv::io::camera::DAVIS::mLogicClockActual (C++ member), 252

dv::io::camera::DAVIS::mLogicPatch (C++ member), 252
 dv::io::camera::DAVIS::mLogicVersion (C++ member), 252
 dv::io::camera::DAVIS::MODULE_APS (C++ member), 254
 dv::io::camera::DAVIS::MODULE_BIAS (C++ member), 254
 dv::io::camera::DAVIS::MODULE_CHIP (C++ member), 255
 dv::io::camera::DAVIS::MODULE_DVS (C++ member), 254
 dv::io::camera::DAVIS::MODULE_EXTERNAL_INPUT (C++ member), 254
 dv::io::camera::DAVIS::MODULE_IMU (C++ member), 254
 dv::io::camera::DAVIS::MODULE_MULTIPLEXER (C++ member), 254
 dv::io::camera::DAVIS::MODULE_SYSINFO (C++ member), 255
 dv::io::camera::DAVIS::MODULE_USB (C++ member), 255
 dv::io::camera::DAVIS::mParser (C++ member), 252
 dv::io::camera::DAVIS::mSensorModel (C++ member), 252
 dv::io::camera::DAVIS::mShiftedSourceNBias (C++ member), 252
 dv::io::camera::DAVIS::mShiftedSourcePBias (C++ member), 252
 dv::io::camera::DAVIS::msvHistogram (C++ member), 249
 dv::io::camera::DAVIS::mTimestampMaster (C++ member), 253
 dv::io::camera::DAVIS::mUSBClockActual (C++ member), 252
 dv::io::camera::DAVIS::MUX_DROP_DVS_ON_TRANSFER_STALL (C++ member), 255
 dv::io::camera::DAVIS::MUX_DROP_EXTINPUT_ON_TRANSFER_STALL (C++ member), 255
 dv::io::camera::DAVIS::MUX_HAS_STATISTICS (C++ member), 255
 dv::io::camera::DAVIS::MUX_RUN (C++ member), 255
 dv::io::camera::DAVIS::MUX_RUN_CHIP (C++ member), 255
 dv::io::camera::DAVIS::MUX_STATISTICS_DVS_DROPPED (C++ member), 255
 dv::io::camera::DAVIS::MUX_STATISTICS_EXTINPUT_DROPPED (C++ member), 255
 dv::io::camera::DAVIS::MUX_TIMESTAMP_RESET (C++ member), 255
 dv::io::camera::DAVIS::MUX_TIMESTAMP_RUN (C++ member), 255
 dv::io::camera::DAVIS::muxHasStatistics (C++ member), 249
 dv::io::camera::DAVIS::mVDACBiases (C++ member), 252
 dv::io::camera::DAVIS::PID_DAVIS_FX2 (C++ member), 269
 dv::io::camera::DAVIS::PID_DAVIS_FX3 (C++ member), 269
 dv::io::camera::DAVIS::PIXEL_FILTER_MAX_SIZE (C++ member), 270
 dv::io::camera::DAVIS::pixelHistogram (C++ member), 249
 dv::io::camera::DAVIS::sendCoarseFineBias (C++ function), 251
 dv::io::camera::DAVIS::sendShiftedSourceBias (C++ function), 251
 dv::io::camera::DAVIS::sendTimestampReset (C++ function), 250
 dv::io::camera::DAVIS::sendVDACBias (C++ function), 251
 dv::io::camera::DAVIS::setAutoExposure (C++ function), 242
 dv::io::camera::DAVIS::setAutoExposureCallback (C++ function), 242
 dv::io::camera::DAVIS::setBackgroundActivityFilter (C++ function), 241
 dv::io::camera::DAVIS::setCDavisAdjustOVG1Low (C++ function), 248
 dv::io::camera::DAVIS::setCDavisAdjustOVG2Low (C++ function), 248
 dv::io::camera::DAVIS::setCDavisAdjustTX2OVGHigh (C++ function), 248
 dv::io::camera::DAVIS::setCDavisBiasCoarseFine (C++ function), 248
 dv::io::camera::DAVIS::setCDavisBiasVoltage (C++ function), 248
 dv::io::camera::DAVIS::setColorMode (C++ function), 242
 dv::io::camera::DAVIS::setCropArea (C++ function), 241
 dv::io::camera::DAVIS::setCropAreaEvents (C++ function), 242
 dv::io::camera::DAVIS::setCropAreaEventsInternal (C++ function), 250
 dv::io::camera::DAVIS::setCropAreaFrames (C++ function), 243
 dv::io::camera::DAVIS::setCropAreaFramesInternal (C++ function), 250
 dv::io::camera::DAVIS::setDavis240BiasCoarseFine (C++ function), 247
 dv::io::camera::DAVIS::setDavis346Bias-

CoarseFine (C++ function), 247

dv::io::camera::DAVIS::setDavis346BiasVoltage (C++ function), 247

dv::io::camera::DAVIS::setDetectorFallingEdges (C++ function), 246

dv::io::camera::DAVIS::setDetectorRisingEdges (C++ function), 245

dv::io::camera::DAVIS::setDetectorRunning (C++ function), 245

dv::io::camera::DAVIS::setEventsRunning (C++ function), 239

dv::io::camera::DAVIS::setExposureDuration (C++ function), 243

dv::io::camera::DAVIS::setExternalAERControl (C++ function), 240

dv::io::camera::DAVIS::setFlatten (C++ function), 241

dv::io::camera::DAVIS::setFlipHorizontal (C++ function), 240

dv::io::camera::DAVIS::setFlipHorizontalEvents (C++ function), 240

dv::io::camera::DAVIS::setFlipHorizontalFrames (C++ function), 243

dv::io::camera::DAVIS::setFlipVertical (C++ function), 240

dv::io::camera::DAVIS::setFlipVerticalEvents (C++ function), 240

dv::io::camera::DAVIS::setFlipVerticalFrames (C++ function), 243

dv::io::camera::DAVIS::setFrameInterval (C++ function), 243

dv::io::camera::DAVIS::setFramesRunning (C++ function), 242

dv::io::camera::DAVIS::setGeneratorHighTime (C++ function), 246

dv::io::camera::DAVIS::setGeneratorInjectTriggerOnFallingEdge (C++ function), 247

dv::io::camera::DAVIS::setGeneratorInjectTriggerOnRisingEdge (C++ function), 246

dv::io::camera::DAVIS::setGeneratorLowTime (C++ function), 246

dv::io::camera::DAVIS::setGeneratorRunning (C++ function), 246

dv::io::camera::DAVIS::setImuAccelRange (C++ function), 245

dv::io::camera::DAVIS::setIMUFlipX (C++ function), 244

dv::io::camera::DAVIS::setIMUFlipY (C++ function), 244

dv::io::camera::DAVIS::setIMUFlipZ (C++ function), 245

dv::io::camera::DAVIS::setImuGyroRange (C++ function), 245

dv::io::camera::DAVIS::setImuRunningAccelerometer (C++ function), 244

dv::io::camera::DAVIS::setImuRunningGyroscope (C++ function), 244

dv::io::camera::DAVIS::setImuRunningTemperature (C++ function), 244

dv::io::camera::DAVIS::setRefractoryPeriodFilter (C++ function), 241

dv::io::camera::DAVIS::setSkipFilter (C++ function), 241

dv::io::camera::DAVIS::setTimeInterval (C++ function), 239

dv::io::camera::DAVIS::setTimestampOffset (C++ function), 250

dv::io::camera::DAVIS::setUSBEarlyPacketDelay (C++ function), 249

dv::io::camera::DAVIS::ShiftedSourceBias (C++ struct), 503

dv::io::camera::DAVIS::ShiftedSourceBias::operatingMode (C++ member), 504

dv::io::camera::DAVIS::ShiftedSourceBias::refValue (C++ member), 504

dv::io::camera::DAVIS::ShiftedSourceBias::regValue (C++ member), 504

dv::io::camera::DAVIS::ShiftedSourceBias::ShiftedSourceBias (C++ function), 503

dv::io::camera::DAVIS::ShiftedSourceBias::voltageLevel (C++ member), 504

dv::io::camera::DAVIS::ShiftedSourceBias::operatingMode (C++ enum), 251

dv::io::camera::DAVIS::ShiftedSourceBias::operatingMode::HI_Z (C++ enumerator), 251

dv::io::camera::DAVIS::ShiftedSourceBias::operatingMode::SHIFTED_SOURCE (C++ enumerator), 251

dv::io::camera::DAVIS::ShiftedSourceBias::operatingMode::TIED_TO_RAIL (C++ enumerator), 251

dv::io::camera::DAVIS::ShiftedSourceBias::VoltageLevel (C++ enum), 251

dv::io::camera::DAVIS::ShiftedSourceBias::asVoltageLevel::DOUBLE_DIODE (C++ enumerator), 251

dv::io::camera::DAVIS::ShiftedSourceBias::asVoltageLevel::SINGLE_DIODE (C++ enumerator), 251

dv::io::camera::DAVIS::ShiftedSourceBias::asVoltageLevel::SPLIT_GATE (C++ enumerator), 251

dv::io::camera::DAVIS::shutdownCallback (C++ function), 251
 dv::io::camera::DAVIS::snapshot (C++ function), 242
 dv::io::camera::DAVIS::SYSINFO_ADC_CLOCK (C++ member), 265
 dv::io::camera::DAVIS::SYSINFO_CHIP_IDENTIFIER (C++ member), 265
 dv::io::camera::DAVIS::SYSINFO_CLOCK_DEVIATION (C++ member), 266
 dv::io::camera::DAVIS::SYSINFO_DEVICE_IS_MASTER (C++ member), 265
 dv::io::camera::DAVIS::SYSINFO_LOGIC_CLOCK (C++ member), 265
 dv::io::camera::DAVIS::SYSINFO_LOGIC_PATCH (C++ member), 266
 dv::io::camera::DAVIS::SYSINFO_LOGIC_VERSION (C++ member), 265
 dv::io::camera::DAVIS::SYSINFO_USB_CLOCK (C++ member), 265
 dv::io::camera::DAVIS::timeInitCallback (C++ function), 251
 dv::io::camera::DAVIS::USB_EARLY_PACKET_DELAY (C++ member), 266
 dv::io::camera::DAVIS::USB_EARLY_PACKET_DELAY_MAX (C++ member), 270
 dv::io::camera::DAVIS::USB_RUN (C++ member), 266
 dv::io::camera::DAVIS::usbDataCallback (C++ function), 251
 dv::io::camera::DAVIS::VDACBias (C++ struct), 566
 dv::io::camera::DAVIS::VDACBias::currentValue (C++ member), 566
 dv::io::camera::DAVIS::VDACBias::VDACBias (C++ function), 566
 dv::io::camera::DAVIS::VDACBias::voltageValue (C++ member), 566
 dv::io::camera::discover (C++ function), 617
 dv::io::camera::DVS128 (C++ class), 276
 dv::io::camera::DVS128::~DVS128 (C++ function), 277
 dv::io::camera::DVS128::Bias (C++ enum), 276
 dv::io::camera::DVS128::Bias::CAS (C++ enumerator), 276
 dv::io::camera::DVS128::Bias::DIFF (C++ enumerator), 277
 dv::io::camera::DVS128::Bias::DIFF_OFF (C++ enumerator), 276
 dv::io::camera::DVS128::Bias::DIFF_ON (C++ enumerator), 277
 dv::io::camera::DVS128::Bias::FOLL (C++ enumerator), 277
 dv::io::camera::DVS128::Bias::INJGND (C++ enumerator), 276
 dv::io::camera::DVS128::Bias::PR (C++ enumerator), 277
 dv::io::camera::DVS128::Bias::PUX (C++ enumerator), 276
 dv::io::camera::DVS128::Bias::PUY (C++ enumerator), 277
 dv::io::camera::DVS128::Bias::REFR (C++ enumerator), 277
 dv::io::camera::DVS128::Bias::REQ (C++ enumerator), 277
 dv::io::camera::DVS128::Bias::REQPD (C++ enumerator), 276
 dv::io::camera::DVS128::COMPATIBLE_CAMERA (C++ member), 282
 dv::io::camera::DVS128::DATA_ENDPOINT (C++ member), 282
 dv::io::camera::DVS128::dataParserCallback (C++ function), 281
 dv::io::camera::DVS128::DVS128 (C++ function), 277
 dv::io::camera::DVS128::findDevices (C++ function), 280
 dv::io::camera::DVS128::FIRMWARE_REQUIRED_VERSION (C++ member), 281
 dv::io::camera::DVS128::getBias (C++ function), 279
 dv::io::camera::DVS128::getCameraName (C++ function), 277
 dv::io::camera::DVS128::getCropArea (C++ function), 280
 dv::io::camera::DVS128::getEventResolution (C++ function), 278
 dv::io::camera::DVS128::getFlatten (C++ function), 280
 dv::io::camera::DVS128::getFlipHorizontal (C++ function), 280
 dv::io::camera::DVS128::getFlipVertical (C++ function), 280
 dv::io::camera::DVS128::getFrameResolution (C++ function), 278
 dv::io::camera::DVS128::getImuModel (C++ function), 278
 dv::io::camera::DVS128::getPixelPitch (C++ function), 278
 dv::io::camera::DVS128::getTimeInterval (C++ function), 279
 dv::io::camera::DVS128::getTimestampOffset (C++ function), 279
 dv::io::camera::DVS128::isEventStreamAvailable (C++ function), 278
 dv::io::camera::DVS128::isFrameStreamAvailable (C++ function), 278
 dv::io::camera::DVS128::isImuStreamAvail-

able (C++ function), 278

dv::io::camera::DVS128::isMaster (C++ function), 278

dv::io::camera::DVS128::isRunning (C++ function), 279

dv::io::camera::DVS128::isRunningAny (C++ function), 279

dv::io::camera::DVS128::isTriggerStreamAvailable (C++ function), 278

dv::io::camera::DVS128::mBiases (C++ member), 281

dv::io::camera::DVS128::mCallbackConfigLock (C++ member), 281

dv::io::camera::DVS128::mConfigLock (C++ member), 281

dv::io::camera::DVS128::mCropArea (C++ member), 281

dv::io::camera::DVS128::mFlatten (C++ member), 281

dv::io::camera::DVS128::mIsRunning (C++ member), 281

dv::io::camera::DVS128::mParser (C++ member), 281

dv::io::camera::DVS128::mTimestampMaster (C++ member), 281

dv::io::camera::DVS128::PID_DVS128 (C++ member), 281

dv::io::camera::DVS128::sendTimestampReset (C++ function), 281

dv::io::camera::DVS128::setBias (C++ function), 279

dv::io::camera::DVS128::setBiasInternal (C++ function), 281

dv::io::camera::DVS128::setCropArea (C++ function), 280

dv::io::camera::DVS128::setFlatten (C++ function), 280

dv::io::camera::DVS128::setFlipHorizontal (C++ function), 280

dv::io::camera::DVS128::setFlipVertical (C++ function), 280

dv::io::camera::DVS128::setMaster (C++ function), 278

dv::io::camera::DVS128::setTimeInterval (C++ function), 279

dv::io::camera::DVS128::setTimestampOffset (C++ function), 281

dv::io::camera::DVS128::shutdownCallback (C++ function), 281

dv::io::camera::DVS128::timeInitCallback (C++ function), 281

dv::io::camera::DVS128::usbDataCallback (C++ function), 281

dv::io::camera::DVS128::VENDOR_REQUEST_RE-

SET_ARRAY (C++ member), 282

dv::io::camera::DVS128::VENDOR_REQUEST_RE-

SET_TS (C++ member), 282

dv::io::camera::DVS128::VENDOR_RE-

QUEST_SEND_BIASES (C++ member), 282

dv::io::camera::DVS128::VENDOR_RE-

QUEST_START_TRANSFER (C++ member), 282

dv::io::camera::DVS128::VENDOR_RE-

QUEST_STOP_TRANSFER (C++ member), 282

dv::io::camera::DVS128::VENDOR_RE-

QUEST_TS_MASTER (C++ member), 282

dv::io::camera::DVXplorer (C++ class), 282

dv::io::camera::DVXplorer::~~DVXplorer (C++ function), 284

dv::io::camera::DVXplorer::COMPATIBLE_CAMERA (C++ member), 303

dv::io::camera::DVXplorer::dataParserCallback (C++ function), 293

dv::io::camera::DVX-

plorer::DVS_CHIP_DTAG_CONTROL_RESTART (C++ member), 302

dv::io::camera::DVX-

plorer::DVS_CHIP_DTAG_CONTROL_START (C++ member), 302

dv::io::camera::DVX-

plorer::DVS_CHIP_DTAG_CONTROL_STOP (C++ member), 302

dv::io::camera::DVX-

plorer::DVS_CHIP_MODE_MONITOR (C++ member), 302

dv::io::camera::DVX-

plorer::DVS_CHIP_MODE_OFF (C++ member), 302

dv::io::camera::DVX-

plorer::DVS_CHIP_MODE_STREAM (C++ member), 302

dv::io::camera::DVXplorer::DVS_ORIENTA-

TION_INFO (C++ member), 296

dv::io::camera::DVXplorer::DVS_RUN (C++ member), 296

dv::io::camera::DVX-

plorer::DVS_SIZE_COLUMNS (C++ member), 296

dv::io::camera::DVXplorer::DVS_SIZE_ROWS (C++ member), 296

dv::io::camera::DVXplorer::DVS_STATIS-

TICS_COLUMN (C++ member), 296

dv::io::camera::DVXplorer::DVS_STATIS-

TICS_DROPPED_COLUMN (C++ member), 296

dv::io::camera::DVXplorer::DVS_STATIS-

TICS_DROPPED_GROUP (C++ member),

- 296
- dv::io::camera::DVXplorer::DVS_STATISTICS_GROUP (C++ member), 296
- dv::io::camera::DVXplorer::DVXplorer (C++ function), 283
- dv::io::camera::DVXplorer::EXT_INPUT_TIME_MAX (C++ member), 303
- dv::io::camera::DVXplorer::EXTINPUT_DETECT_FALLING_EDGES (C++ member), 297
- dv::io::camera::DVXplorer::EXTINPUT_DETECT_RISING_EDGES (C++ member), 297
- dv::io::camera::DVXplorer::EXTINPUT_GENERATE_INJECT_ON_FALLING_EDGE (C++ member), 298
- dv::io::camera::DVXplorer::EXTINPUT_GENERATE_INJECT_ON_RISING_EDGE (C++ member), 298
- dv::io::camera::DVXplorer::EXTINPUT_GENERATE_PULSE_INTERVAL (C++ member), 298
- dv::io::camera::DVXplorer::EXTINPUT_GENERATE_PULSE_LENGTH (C++ member), 298
- dv::io::camera::DVXplorer::EXTINPUT_RUN_DETECTOR (C++ member), 297
- dv::io::camera::DVXplorer::EXTINPUT_RUN_GENERATOR (C++ member), 298
- dv::io::camera::DVXplorer::findDevices (C++ function), 293
- dv::io::camera::DVXplorer::FIRMWARE_REQUIRED_VERSION (C++ member), 303
- dv::io::camera::DVXplorer::getCameraName (C++ function), 284
- dv::io::camera::DVXplorer::getContrastThresholdOff (C++ function), 288
- dv::io::camera::DVXplorer::getContrastThresholdOn (C++ function), 288
- dv::io::camera::DVXplorer::getCropArea (C++ function), 288
- dv::io::camera::DVXplorer::getDetectorFallingEdges (C++ function), 291
- dv::io::camera::DVXplorer::getDetectorRisingEdges (C++ function), 291
- dv::io::camera::DVXplorer::getDualBinning (C++ function), 287
- dv::io::camera::DVXplorer::getEventResolution (C++ function), 284
- dv::io::camera::DVXplorer::getFlatten (C++ function), 286
- dv::io::camera::DVXplorer::getFlipHorizontal (C++ function), 286
- dv::io::camera::DVXplorer::getFlipVertical (C++ function), 286
- dv::io::camera::DVXplorer::getFrameResolution (C++ function), 284
- dv::io::camera::DVXplorer::getGeneratorHighTime (C++ function), 292
- dv::io::camera::DVXplorer::getGeneratorHighTimeInternal (C++ function), 293
- dv::io::camera::DVXplorer::getGeneratorInjectTriggerOnFallingEdge (C++ function), 292
- dv::io::camera::DVXplorer::getGeneratorInjectTriggerOnRisingEdge (C++ function), 292
- dv::io::camera::DVXplorer::getGeneratorLowTime (C++ function), 291
- dv::io::camera::DVXplorer::getGeneratorLowTimeInternal (C++ function), 293
- dv::io::camera::DVXplorer::getGlobalHold (C++ function), 287
- dv::io::camera::DVXplorer::getGlobalReset (C++ function), 287
- dv::io::camera::DVXplorer::getImuAccelerDataRate (C++ function), 289
- dv::io::camera::DVXplorer::getImuAccelerFilter (C++ function), 290
- dv::io::camera::DVXplorer::getImuAccelerRange (C++ function), 290
- dv::io::camera::DVXplorer::getIMUFlipX (C++ function), 289
- dv::io::camera::DVXplorer::getIMUFlipY (C++ function), 289
- dv::io::camera::DVXplorer::getIMUFlipZ (C++ function), 289
- dv::io::camera::DVXplorer::getImuGyroDataRate (C++ function), 290
- dv::io::camera::DVXplorer::getImuGyroFilter (C++ function), 290
- dv::io::camera::DVXplorer::getImuGyroRange (C++ function), 290
- dv::io::camera::DVXplorer::getImuModel (C++ function), 284
- dv::io::camera::DVXplorer::getLogicPatchLevel (C++ function), 284
- dv::io::camera::DVXplorer::getLogicVersion (C++ function), 284
- dv::io::camera::DVXplorer::getPixelPitch (C++ function), 284
- dv::io::camera::DVXplorer::getReadoutFPS (C++ function), 287
- dv::io::camera::DVXplorer::getSubSampleHorizontal (C++ function), 286
- dv::io::camera::DVXplorer::getSubSampleVertical (C++ function), 287
- dv::io::camera::DVXplorer::getTimeInterval (C++ function), 285

dv::io::camera::DVXplorer::getTimestampOffset (C++ *function*), 285
 dv::io::camera::DVXplorer::getUSBEarlyPacketDelay (C++ *function*), 292
 dv::io::camera::DVXplorer::IMU_ACCEL_DATA_RATE (C++ *member*), 297
 dv::io::camera::DVXplorer::IMU_ACCEL_FILTER (C++ *member*), 297
 dv::io::camera::DVXplorer::IMU_ACCEL_RANGE (C++ *member*), 297
 dv::io::camera::DVXplorer::IMU_GYRO_DATA_RATE (C++ *member*), 297
 dv::io::camera::DVXplorer::IMU_GYRO_FILTER (C++ *member*), 297
 dv::io::camera::DVXplorer::IMU_GYRO_RANGE (C++ *member*), 297
 dv::io::camera::DVXplorer::IMU_ORIENTATION_INFO (C++ *member*), 296
 dv::io::camera::DVXplorer::IMU_RUN_ACCELEROMETER (C++ *member*), 297
 dv::io::camera::DVXplorer::IMU_RUN_GYROSCOPE (C++ *member*), 297
 dv::io::camera::DVXplorer::IMU_RUN_TEMPERATURE (C++ *member*), 297
 dv::io::camera::DVXplorer::IMU_TYPE (C++ *member*), 296
 dv::io::camera::DVXplorer::isDetectorRunning (C++ *function*), 291
 dv::io::camera::DVXplorer::isEventsRunning (C++ *function*), 286
 dv::io::camera::DVXplorer::isEventStreamAvailable (C++ *function*), 284
 dv::io::camera::DVXplorer::isFrameStreamAvailable (C++ *function*), 285
 dv::io::camera::DVXplorer::isGeneratorRunning (C++ *function*), 291
 dv::io::camera::DVXplorer::isImuRunningAccelerometer (C++ *function*), 288
 dv::io::camera::DVXplorer::isImuRunningGyroscope (C++ *function*), 288
 dv::io::camera::DVXplorer::isImuRunningTemperature (C++ *function*), 289
 dv::io::camera::DVXplorer::isImuStreamAvailable (C++ *function*), 285
 dv::io::camera::DVXplorer::isMaster (C++ *function*), 284
 dv::io::camera::DVXplorer::isRunning (C++ *function*), 285
 dv::io::camera::DVXplorer::isRunningAny (C++ *function*), 285
 dv::io::camera::DVXplorer::isTriggerStreamAvailable (C++ *function*), 285
 dv::io::camera::DVXplorer::LOGIC_MINI-MUM_PATCH (C++ *member*), 303
 dv::io::camera::DVXplorer::LOGIC_REQUIRED_VERSION (C++ *member*), 303
 dv::io::camera::DVXplorer::mConfigLock (C++ *member*), 294
 dv::io::camera::DVXplorer::mContrastThresholdOff (C++ *member*), 294
 dv::io::camera::DVXplorer::mContrastThresholdOn (C++ *member*), 294
 dv::io::camera::DVXplorer::mCropArea (C++ *member*), 294
 dv::io::camera::DVXplorer::mImuModel (C++ *member*), 294
 dv::io::camera::DVXplorer::mIsRunning (C++ *member*), 294
 dv::io::camera::DVXplorer::mLogicClockActual (C++ *member*), 294
 dv::io::camera::DVXplorer::mLogicPatch (C++ *member*), 293
 dv::io::camera::DVXplorer::mLogicVersion (C++ *member*), 293
 dv::io::camera::DVXplorer::MODULE_DEVICE (C++ *member*), 299
 dv::io::camera::DVXplorer::MODULE_DVS (C++ *member*), 294
 dv::io::camera::DVXplorer::MODULE_EXTERNAL_INPUT (C++ *member*), 294
 dv::io::camera::DVXplorer::MODULE_IMU (C++ *member*), 294
 dv::io::camera::DVXplorer::MODULE_MULTIPLEXER (C++ *member*), 294
 dv::io::camera::DVXplorer::MODULE_SYSINFO (C++ *member*), 295
 dv::io::camera::DVXplorer::MODULE_USB (C++ *member*), 295
 dv::io::camera::DVXplorer::mParser (C++ *member*), 294
 dv::io::camera::DVXplorer::mReadoutFPS (C++ *member*), 294
 dv::io::camera::DVXplorer::mResolution (C++ *member*), 294
 dv::io::camera::DVXplorer::mSensorModel (C++ *member*), 294
 dv::io::camera::DVXplorer::mTimestampMaster (C++ *member*), 294
 dv::io::camera::DVXplorer::mUSBClockActual (C++ *member*), 294
 dv::io::camera::DVXplorer::MUX_DROP_DVS_ON_TRANSFER_STALL (C++ *member*), 295
 dv::io::camera::DVXplorer::MUX_DROP_EXTINPUT_ON_TRANSFER_STALL (C++ *member*), 295
 dv::io::camera::DVXplorer::MUX_RUN (C++

member), 295

dv::io::camera::DVXplorer::MUX_RUN_CHIP (C++ member), 295

dv::io::camera::DVXplorer::MUX_STATISTICS_DVS_DROPPED (C++ member), 295

dv::io::camera::DVXplorer::MUX_STATISTICS_EXTINPUT_DROPPED (C++ member), 295

dv::io::camera::DVXplorer::MUX_TIMES_TAMP_RESET (C++ member), 295

dv::io::camera::DVXplorer::MUX_TIMES_TAMP_RUN (C++ member), 295

dv::io::camera::DVXplorer::operator<< (C++ function), 303

dv::io::camera::DVXplorer::PID_DVXPLOER (C++ member), 303

dv::io::camera::DVXplorer::ReadoutFPS (C++ enum), 282

dv::io::camera::DVXplorer::ReadoutFPS::CONSTANT_100 (C++ enumerator), 283

dv::io::camera::DVXplorer::ReadoutFPS::CONSTANT_200 (C++ enumerator), 283

dv::io::camera::DVXplorer::ReadoutFPS::CONSTANT_500 (C++ enumerator), 283

dv::io::camera::DVXplorer::ReadoutFPS::CONSTANT_1000 (C++ enumerator), 283

dv::io::camera::DVXplorer::ReadoutFPS::CONSTANT_LOSSY_2000 (C++ enumerator), 283

dv::io::camera::DVXplorer::ReadoutFPS::CONSTANT_LOSSY_5000 (C++ enumerator), 283

dv::io::camera::DVXplorer::ReadoutFPS::CONSTANT_LOSSY_10000 (C++ enumerator), 283

dv::io::camera::DVXplorer::ReadoutFPS::VARIABLE_2000 (C++ enumerator), 283

dv::io::camera::DVXplorer::ReadoutFPS::VARIABLE_5000 (C++ enumerator), 283

dv::io::camera::DVXplorer::ReadoutFPS::VARIABLE_10000 (C++ enumerator), 283

dv::io::camera::DVXplorer::ReadoutFPS::VARIABLE_15000 (C++ enumerator), 283

dv::io::camera::DVXplorer::REGISTER_ACTIVITY_DECISION_BYPASS (C++ member), 302

dv::io::camera::DVXplorer::REGISTER_TER_BIAS_CURRENT_AMP (C++ member), 300

dv::io::camera::DVXplorer::REGISTER_TER_BIAS_CURRENT_LEVEL_SFOFF (C++ member), 299

dv::io::camera::DVXplorer::REGISTER_TER_BIAS_CURRENT_OFF (C++ member), 300

dv::io::camera::DVXplorer::REGISTER_TER_BIAS_CURRENT_ON (C++ member), 300

dv::io::camera::DVXplorer::REGISTER_TER_BIAS_CURRENT_RANGE_SELECT_LOGALOGD_MONITOR (C++ member), 299

dv::io::camera::DVXplorer::REGISTER_TER_BIAS_CURRENT_RANGE_SELECT_LOGSFONREST (C++ member), 299

dv::io::camera::DVXplorer::REGISTER_TER_BIAS_OTP_TRIM (C++ member), 299

dv::io::camera::DVXplorer::REGISTER_TER_BIAS_PINS_BUFN (C++ member), 299

dv::io::camera::DVXplorer::REGISTER_TER_BIAS_PINS_BUFN (C++ member), 299

dv::io::camera::DVXplorer::REGISTER_TER_BIAS_PINS_DBGN (C++ member), 299

dv::io::camera::DVXplorer::REGISTER_TER_BIAS_PINS_DBGP (C++ member), 299

dv::io::camera::DVXplorer::REGISTER_TER_BIAS_PINS_DOB (C++ member), 299

dv::io::camera::DVXplorer::REGISTER_CONTROL_CLOCK_DIVIDER_PVI (C++ member), 300

dv::io::camera::DVXplorer::REGISTER_CONTROL_CLOCK_DIVIDER_SYS (C++ member), 300

dv::io::camera::DVXplorer::REGISTER_CONTROL_INTERRUPT_ACKNOWLEDGE (C++ member), 300

dv::io::camera::DVXplorer::REGISTER_CONTROL_INTERRUPT_AUTO_MODE (C++ member), 300

dv::io::camera::DVXplorer::REGISTER_CONTROL_INTERRUPT_ENABLE_TIME (C++ member), 300

dv::io::camera::DVXplorer::REGISTER_CONTROL_INTERRUPT_RELEASE_TIME (C++ member), 300

dv::io::camera::DVXplorer::REGISTER_CONTROL_INTERRUPT_SOURCE (C++ member), 300

300

dv::io::camera::DVXplorer::REGISTER_CONTROL_MODE (C++ member), 300

dv::io::camera::DVXplorer::REGISTER_CONTROL_PACKET_FORMAT (C++ member), 300

dv::io::camera::DVXplorer::REGISTER_CONTROL_PARALLEL_OUT_CONTROL (C++ member), 300

dv::io::camera::DVXplorer::REGISTER_CONTROL_PARALLEL_OUT_ENABLE (C++ member), 300

dv::io::camera::DVXplorer::REGISTER_CONTROL_PLL_M (C++ member), 300

dv::io::camera::DVXplorer::REGISTER_CONTROL_PLL_P (C++ member), 300

dv::io::camera::DVXplorer::REGISTER_CONTROL_PLL_S (C++ member), 300

dv::io::camera::DVXplorer::REGISTER_CROPPER_BYPASS (C++ member), 302

dv::io::camera::DVXplorer::REGISTER_CROPPER_X_END_ADDRESS (C++ member), 302

dv::io::camera::DVXplorer::REGISTER_CROPPER_X_START_ADDRESS (C++ member), 302

dv::io::camera::DVXplorer::REGISTER_CROPPER_Y_END_GROUP (C++ member), 302

dv::io::camera::DVXplorer::REGISTER_CROPPER_Y_END_MASK (C++ member), 302

dv::io::camera::DVXplorer::REGISTER_CROPPER_Y_START_GROUP (C++ member), 302

dv::io::camera::DVXplorer::REGISTER_CROPPER_Y_START_MASK (C++ member), 302

dv::io::camera::DVXplorer::REGISTER_DIGITAL_AREA_BLOCK (C++ member), 301

dv::io::camera::DVXplorer::REGISTER_DIGITAL_BOOT_SEQUENCE (C++ member), 302

dv::io::camera::DVXplorer::REGISTER_DIGITAL_DTAG_REFERENCE (C++ member), 301

dv::io::camera::DVXplorer::REGISTER_DIGITAL_DUAL_BINNING (C++ member), 300

dv::io::camera::DVXplorer::REGISTER_DIGITAL_ENABLE (C++ member), 300

dv::io::camera::DVXplorer::REGISTER_DIGITAL_EXTERNAL_TRIGGER (C++ member), 302

dv::io::camera::DVXplorer::REGISTER_DIGITAL_FIXED_READ_TIME (C++ member), 302

dv::io::camera::DVXplorer::REGISTER_DIGITAL_GLOBAL_RESET_READOUT (C++ member), 301

dv::io::camera::DVXplorer::REGISTER_DIGITAL_MODE_CONTROL (C++ member), 301

dv::io::camera::DVXplorer::REGISTER_DIGITAL_RESTART (C++ member), 300

dv::io::camera::DVXplorer::REGISTER_DIGITAL_SUBSAMPLE_RATIO (C++ member), 300

dv::io::camera::DVXplorer::REGISTER_DIGITAL_TIMESTAMP_REFUNIT (C++ member), 301

dv::io::camera::DVXplorer::REGISTER_DIGITAL_TIMESTAMP_RESET (C++ member), 301

dv::io::camera::DVXplorer::REGISTER_DIGITAL_TIMESTAMP_SUBUNIT (C++ member), 301

dv::io::camera::DVXplorer::REGISTER_SPATIAL_HISTOGRAM_OFF (C++ member), 302

dv::io::camera::DVXplorer::REGISTER_TIMING_AY_END (C++ member), 301

dv::io::camera::DVXplorer::REGISTER_TIMING_AY_START (C++ member), 301

dv::io::camera::DVXplorer::REGISTER_TIMING_FIRST_SELX_START (C++ member), 301

dv::io::camera::DVXplorer::REGISTER_TIMING_GH_COUNT (C++ member), 301

dv::io::camera::DVXplorer::REGISTER_TIMING_GH_COUNT_FINE (C++ member), 301

dv::io::camera::DVXplorer::REGISTER_TIMING_GRS_COUNT (C++ member), 301

dv::io::camera::DVXplorer::REGISTER_TIMING_GRS_COUNT_FINE (C++ member), 301

dv::io::camera::DVXplorer::REGISTER_TIMING_GRS_END (C++ member), 301

dv::io::camera::DVXplorer::REGISTER_TIMING_GRS_END_FINE (C++ member), 301

dv::io::camera::DVXplorer::REGISTER_TIMING_MAX_EVENT_NUM (C++ member), 301

dv::io::camera::DVXplorer::REGISTER_TIMING_NEXT_GH_CNT (C++ member), 301

dv::io::camera::DVXplorer::REGISTER_TIMING_NEXT_SELX_START (C++ member), 302

dv::io::camera::DVXplorer::REGISTER_TIMING_R_END (C++ member), 301

dv::io::camera::DVXplorer::REGISTER_TIMING_R_START (C++ member), 301

dv::io::camera::DVXplorer::REGISTER_TIMING_READ_TIME_INTERVAL (C++ member), 302

dv::io::camera::DVXplorer::REGISTER_TIMING_SELX_WIDTH (C++ member), 301

dv::io::camera::DVXplorer::sendTimestamp-
 Reset (C++ function), 293
 dv::io::camera::DVXplorer::SensorModel (C++
 enum), 293
 dv::io::camera::DVXplorer::Sensor-
 Model::DVX_VGA (C++ enumerator), 293
 dv::io::camera::DVXplorer::Sensor-
 Model::LITE_QVGA (C++ enumerator),
 293
 dv::io::camera::DVXplorer::setContrast-
 ThresholdOff (C++ function), 288
 dv::io::camera::DVXplorer::setContrast-
 ThresholdOn (C++ function), 288
 dv::io::camera::DVXplorer::setCropArea (C++
 function), 288
 dv::io::camera::DVXplorer::setCropAreaIn-
 ternal (C++ function), 293
 dv::io::camera::DVXplorer::setDetector-
 FallingEdges (C++ function), 291
 dv::io::camera::DVXplorer::setDetector-
 RisingEdges (C++ function), 291
 dv::io::camera::DVXplorer::setDetectorRun-
 ning (C++ function), 291
 dv::io::camera::DVXplorer::setDualBinning
 (C++ function), 287
 dv::io::camera::DVXplorer::setEventsRun-
 ning (C++ function), 286
 dv::io::camera::DVXplorer::setFlatten (C++
 function), 286
 dv::io::camera::DVXplorer::setFlipHorizon-
 tal (C++ function), 286
 dv::io::camera::DVXplorer::setFlipVertical
 (C++ function), 286
 dv::io::camera::DVXplorer::setGenera-
 torHighTime (C++ function), 292
 dv::io::camera::DVXplorer::setGenerator-
 InjectTriggerOnFallingEdge (C++
 function), 292
 dv::io::camera::DVXplorer::setGeneratorIn-
 jectTriggerOnRisingEdge (C++ function),
 292
 dv::io::camera::DVXplorer::setGenerator-
 LowTime (C++ function), 292
 dv::io::camera::DVXplorer::setGenerator-
 Running (C++ function), 291
 dv::io::camera::DVXplorer::setGlobalHold
 (C++ function), 287
 dv::io::camera::DVXplorer::setGlobalReset
 (C++ function), 287
 dv::io::camera::DVXplorer::setImuAccel-
 DataRate (C++ function), 290
 dv::io::camera::DVXplorer::setImuAccelFil-
 ter (C++ function), 290
 dv::io::camera::DVXplorer::setImuAccel-
 Range (C++ function), 290
 dv::io::camera::DVXplorer::setIMUFlipX (C++
 function), 289
 dv::io::camera::DVXplorer::setIMUFlipY (C++
 function), 289
 dv::io::camera::DVXplorer::setIMUFlipZ (C++
 function), 289
 dv::io::camera::DVXplorer::setImuGyro-
 DataRate (C++ function), 290
 dv::io::camera::DVXplorer::setImuGyroFil-
 ter (C++ function), 290
 dv::io::camera::DVXplorer::setImuGyroRange
 (C++ function), 291
 dv::io::camera::DVXplorer::setImuRun-
 ningAccelerometer (C++ function),
 288
 dv::io::camera::DVXplorer::setImuRunning-
 Gyroscope (C++ function), 289
 dv::io::camera::DVXplorer::setImuRun-
 ningTemperature (C++ function), 289
 dv::io::camera::DVXplorer::setReadoutFPS
 (C++ function), 288
 dv::io::camera::DVXplorer::setSubSample-
 Horizontal (C++ function), 286
 dv::io::camera::DVXplorer::setSubSampleV-
 ertical (C++ function), 287
 dv::io::camera::DVXplorer::setTimeInterval
 (C++ function), 285
 dv::io::camera::DVXplorer::setTimestam-
 pOffset (C++ function), 293
 dv::io::camera::DVXplorer::setUSBEarly-
 PacketDelay (C++ function), 292
 dv::io::camera::DVXplorer::shutdownCall-
 back (C++ function), 293
 dv::io::camera::DVXplorer::SubSample (C++
 enum), 282
 dv::io::camera::DVXplorer::SubSam-
 ple::BAND_OF_FOUR (C++ enumerator),
 282
 dv::io::camera::DVXplorer::SubSam-
 ple::BAND_OF_TWO (C++ enumerator),
 282
 dv::io::camera::DVXplorer::SubSam-
 ple::BIN_1010_0000 (C++ enumerator),
 282
 dv::io::camera::DVXplorer::SubSam-
 ple::BIN_1100_0000 (C++ enumerator),
 282
 dv::io::camera::DVXplorer::SubSample::EV-
 ERY_EIGHTH (C++ enumerator), 282
 dv::io::camera::DVXplorer::SubSample::EV-
 ERY_FOURTH (C++ enumerator), 282
 dv::io::camera::DVXplorer::SubSample::EV-
 ERY_PIXEL (C++ enumerator), 282

dv::io::camera::DVXplorer::SubSample::EV-
 ERY_SECOND (C++ *enumerator*), 282
 dv::io::camera::DVXplorer::SYS-
 INFO_CHIP_IDENTIFIER (C++ *member*),
 298
 dv::io::camera::DVXplorer::SYS-
 INFO_CLOCK_DEVIATION (C++ *member*),
 299
 dv::io::camera::DVXplorer::SYSINFO_DE-
 VICE_IS_MASTER (C++ *member*), 298
 dv::io::camera::DVXplorer::SYS-
 INFO_LOGIC_CLOCK (C++ *member*), 298
 dv::io::camera::DVXplorer::SYS-
 INFO_LOGIC_PATCH (C++ *member*), 299
 dv::io::camera::DVXplorer::SYS-
 INFO_LOGIC_VERSION (C++ *member*),
 298
 dv::io::camera::DVXplorer::SYS-
 INFO_USB_CLOCK (C++ *member*), 298
 dv::io::camera::DVXplorer::SYS-
 TEM_CLOCK_FREQUENCY (C++ *member*),
 302
 dv::io::camera::DVXplorer::timeInitCall-
 back (C++ *function*), 293
 dv::io::camera::DVX-
 plorer::USB_EARLY_PACKET_DELAY
 (C++ *member*), 299
 dv::io::camera::DVX-
 plorer::USB_EARLY_PACKET_DELAY_MAX
 (C++ *member*), 303
 dv::io::camera::DVXplorer::USB_RUN (C++
member), 299
 dv::io::camera::DVXplorer::usbDataCallback
 (C++ *function*), 293
 dv::io::camera::DVXplorerM (C++ *class*), 303
 dv::io::camera::DVXplorerM::~~DVXplorerM
 (C++ *function*), 304
 dv::io::camera::DVXplorerM::COMPATI-
 BLE_CAMERA (C++ *member*), 314
 dv::io::camera::DVXplorerM::configLoad-
 FromFlash (C++ *function*), 304
 dv::io::camera::DVXplorerM::config-
 StoreToFlash (C++ *function*), 304
 dv::io::camera::DVXplorerM::dataParser-
 Callback (C++ *function*), 311
 dv::io::camera::DVXplor-
 erM::DVS_AREA_BLOCKING_BLOCK (C++
member), 313
 dv::io::camera::DVXplor-
 erM::DVS_AREA_BLOCKING_X_BLOCK
 (C++ *member*), 313
 dv::io::camera::DVXplor-
 erM::DVS_AREA_BLOCKING_Y_BLOCK
 (C++ *member*), 313
 dv::io::camera::DVXplorerM::DVS_CON-
 TRAST_THRESHOLD_OFF (C++ *member*),
 313
 dv::io::camera::DVXplorerM::DVS_CON-
 TRAST_THRESHOLD_ON (C++ *member*),
 312
 dv::io::camera::DVXplorerM::DVS_CROP_APPLY
 (C++ *member*), 313
 dv::io::camera::DVXplor-
 erM::DVS_CROP_HEIGHT (C++ *member*),
 313
 dv::io::camera::DVXplorerM::DVS_CROP_WIDTH
 (C++ *member*), 313
 dv::io::camera::DVXplorerM::DVS_CROP_X (C++
member), 313
 dv::io::camera::DVXplorerM::DVS_CROP_Y (C++
member), 313
 dv::io::camera::DVXplor-
 erM::DVS_EFPS_S5K231Y (C++ *member*),
 312
 dv::io::camera::DVXplorerM::DVS_FLATTEN
 (C++ *member*), 312
 dv::io::camera::DVXplorerM::DVS_FLIP_HORI-
 ZONTAL (C++ *member*), 313
 dv::io::camera::DVXplorerM::DVS_FLIP_VER-
 TICAL (C++ *member*), 313
 dv::io::camera::DVXplor-
 erM::DVS_GLOBAL_HOLD (C++ *member*),
 312
 dv::io::camera::DVXplorerM::DVS_GLOBAL_RE-
 SET (C++ *member*), 312
 dv::io::camera::DVXplorerM::DVS_GLOBAL_RE-
 SET_SKIP (C++ *member*), 312
 dv::io::camera::DVXplorerM::DVS_MIPI_TIME-
 OUT_ENABLE (C++ *member*), 312
 dv::io::camera::DVXplorerM::DVS_MIPI_TIME-
 OUT_VALUE (C++ *member*), 312
 dv::io::camera::DVXplorerM::DVS_ORIENTA-
 TION (C++ *member*), 312
 dv::io::camera::DVXplorerM::DVS_RESOLU-
 TION_X (C++ *member*), 312
 dv::io::camera::DVXplorerM::DVS_RESOLU-
 TION_Y (C++ *member*), 312
 dv::io::camera::DVXplorerM::DVS_RUN (C++
member), 312
 dv::io::camera::DVXplorerM::DVS_SUBSAM-
 PLE_HORIZONTAL (C++ *member*), 312
 dv::io::camera::DVXplorerM::DVS_SUBSAM-
 PLE_VERTICAL (C++ *member*), 312
 dv::io::camera::DVXplorerM::DVXplorerM (C++
function), 303
 dv::io::camera::DVXplorerM::findDevices
 (C++ *function*), 311
 dv::io::camera::DVXplorerM::FIRMWARE_RE-

QUIRED_VERSION (C++ member), 314
 dv::io::camera::DVXplorerM::getAreaBlock (C++ function), 308
 dv::io::camera::DVXplorerM::getCameraName (C++ function), 304
 dv::io::camera::DVXplorerM::getContrastThresholdOff (C++ function), 308
 dv::io::camera::DVXplorerM::getContrastThresholdOn (C++ function), 308
 dv::io::camera::DVXplorerM::getCropArea (C++ function), 308
 dv::io::camera::DVXplorerM::getDVSOrientationX (C++ function), 311
 dv::io::camera::DVXplorerM::getDVSOrientationY (C++ function), 311
 dv::io::camera::DVXplorerM::getEventResolution (C++ function), 304
 dv::io::camera::DVXplorerM::getFlatten (C++ function), 306
 dv::io::camera::DVXplorerM::getFlipHorizontal (C++ function), 305
 dv::io::camera::DVXplorerM::getFlipVertical (C++ function), 306
 dv::io::camera::DVXplorerM::getFrameResolution (C++ function), 304
 dv::io::camera::DVXplorerM::getGlobalHold (C++ function), 306
 dv::io::camera::DVXplorerM::getGlobalReset (C++ function), 307
 dv::io::camera::DVXplorerM::getGlobalResetSkip (C++ function), 307
 dv::io::camera::DVXplorerM::getImuAccelerDataRate (C++ function), 309
 dv::io::camera::DVXplorerM::getImuAccelerFilter (C++ function), 310
 dv::io::camera::DVXplorerM::getImuAccelerRange (C++ function), 310
 dv::io::camera::DVXplorerM::getImuGyroDataRate (C++ function), 310
 dv::io::camera::DVXplorerM::getImuGyroFilter (C++ function), 310
 dv::io::camera::DVXplorerM::getImuGyroRange (C++ function), 310
 dv::io::camera::DVXplorerM::getImuModel (C++ function), 304
 dv::io::camera::DVXplorerM::getMIPITimeoutEnable (C++ function), 307
 dv::io::camera::DVXplorerM::getMIPITimeoutValue (C++ function), 307
 dv::io::camera::DVXplorerM::getPixelPitch (C++ function), 304
 dv::io::camera::DVXplorerM::getReadoutFPS (C++ function), 308
 dv::io::camera::DVXplorerM::getSubSampleHorizontal (C++ function), 306
 dv::io::camera::DVXplorerM::getSubSampleVertical (C++ function), 306
 dv::io::camera::DVXplorerM::getTimeInterval (C++ function), 305
 dv::io::camera::DVXplorerM::getTimeStampOffset (C++ function), 305
 dv::io::camera::DVXplorerM::IMU_ACCEL_DATA_RATE (C++ member), 313
 dv::io::camera::DVXplorerM::IMU_ACCELER_FILTER (C++ member), 313
 dv::io::camera::DVXplorerM::IMU_ACCELER_RANGE (C++ member), 313
 dv::io::camera::DVXplorerM::IMU_FOC_ACCELER_X (C++ member), 314
 dv::io::camera::DVXplorerM::IMU_FOC_ACCELER_Y (C++ member), 314
 dv::io::camera::DVXplorerM::IMU_FOC_ACCELER_Z (C++ member), 314
 dv::io::camera::DVXplorerM::IMU_FOC_APPLY (C++ member), 314
 dv::io::camera::DVXplorerM::IMU_FOC_GYRO_X (C++ member), 314
 dv::io::camera::DVXplorerM::IMU_FOC_GYRO_Y (C++ member), 314
 dv::io::camera::DVXplorerM::IMU_FOC_GYRO_Z (C++ member), 314
 dv::io::camera::DVXplorerM::IMU_FOC_RUN (C++ member), 314
 dv::io::camera::DVXplorerM::IMU_GYRO_DATA_RATE (C++ member), 313
 dv::io::camera::DVXplorerM::IMU_GYRO_FILTER (C++ member), 314
 dv::io::camera::DVXplorerM::IMU_GYRO_RANGE (C++ member), 314
 dv::io::camera::DVXplorerM::IMU_MODEL (C++ member), 313
 dv::io::camera::DVXplorerM::IMU_ORIENTATION (C++ member), 313
 dv::io::camera::DVXplorerM::IMU_RUN_ACCELEROMETER (C++ member), 313
 dv::io::camera::DVXplorerM::IMU_RUN_GYROSCOPE (C++ member), 313
 dv::io::camera::DVXplorerM::IMU_RUN_TEMPERATURE (C++ member), 313
 dv::io::camera::DVXplorerM::isEventStreamAvailable (C++ function), 304
 dv::io::camera::DVXplorerM::isFrameStreamAvailable (C++ function), 304
 dv::io::camera::DVXplorerM::isImuRunningAccelerometer (C++ function), 309

dv::io::camera::DVXplorerM::isImuRunning-
 Gyroscope (C++ function), 309
 dv::io::camera::DVXplorerM::isImuRun-
 ningTemperature (C++ function), 309
 dv::io::camera::DVXplorerM::isImuStrea-
 mAvailable (C++ function), 304
 dv::io::camera::DVXplorerM::isRunning (C++
 function), 305
 dv::io::camera::DVXplorerM::isRunningAny
 (C++ function), 305
 dv::io::camera::DVXplorerM::isTrigger-
 StreamAvailable (C++ function), 305
 dv::io::camera::DVXplorerM::mConfigLock
 (C++ member), 312
 dv::io::camera::DVXplorerM::mImuModel (C++
 member), 312
 dv::io::camera::DVXplorerM::mIsRunning (C++
 member), 312
 dv::io::camera::DVXplorerM::MODULE_DVS (C++
 member), 312
 dv::io::camera::DVXplorerM::MODULE_IMU (C++
 member), 313
 dv::io::camera::DVXplorerM::mParser (C++
 member), 312
 dv::io::camera::DVXplorerM::mResolution
 (C++ member), 312
 dv::io::camera::DVXplorerM::mSensorModel
 (C++ member), 312
 dv::io::camera::DVXplorerM::PID_DVXPLOER
 (C++ member), 314
 dv::io::camera::DVXplorerM::SensorModel
 (C++ enum), 311
 dv::io::camera::DVXplorerM::Sensor-
 Model::S5K231Y (C++ enumerator), 311
 dv::io::camera::DVXplorerM::Sensor-
 Model::S5KRC1S (C++ enumerator), 311
 dv::io::camera::DVXplorerM::setAreaBlock
 (C++ function), 309
 dv::io::camera::DVXplorerM::setContrast-
 ThresholdOff (C++ function), 308
 dv::io::camera::DVXplorerM::setContrast-
 ThresholdOn (C++ function), 308
 dv::io::camera::DVXplorerM::setCropArea
 (C++ function), 308
 dv::io::camera::DVXplorerM::setFlatten (C++
 function), 306
 dv::io::camera::DVXplorerM::setFlipHori-
 zontal (C++ function), 305
 dv::io::camera::DVXplorerM::setFlipVerti-
 cal (C++ function), 306
 dv::io::camera::DVXplorerM::setGlobalHold
 (C++ function), 307
 dv::io::camera::DVXplorerM::setGlobalReset
 (C++ function), 307
 dv::io::camera::DVXplorerM::setGlobalRe-
 setSkip (C++ function), 307
 dv::io::camera::DVXplorerM::setImuAccel-
 DataRate (C++ function), 310
 dv::io::camera::DVXplorerM::se-
 tImuAccelFilter (C++ function), 310
 dv::io::camera::DVXplorerM::setImuAccel-
 Range (C++ function), 310
 dv::io::camera::DVXplorerM::setImuGyro-
 DataRate (C++ function), 310
 dv::io::camera::DVXplorerM::setImuGyroFil-
 ter (C++ function), 310
 dv::io::camera::DVXplorerM::setImuGyroR-
 ange (C++ function), 311
 dv::io::camera::DVXplorerM::setImuRun-
 ningAccelerometer (C++ function), 309
 dv::io::camera::DVXplorerM::setImuRunning-
 Gyroscope (C++ function), 309
 dv::io::camera::DVXplorerM::setImuRun-
 ningTemperature (C++ function), 309
 dv::io::camera::DVXplorerM::setMIPITime-
 outEnable (C++ function), 307
 dv::io::camera::DVXplorerM::setMIPITime-
 outValue (C++ function), 307
 dv::io::camera::DVXplorerM::setReadoutFPS
 (C++ function), 308
 dv::io::camera::DVXplorerM::setSubSample-
 Horizontal (C++ function), 306
 dv::io::camera::DVXplorerM::setSubSampleV-
 ertical (C++ function), 306
 dv::io::camera::DVXplorerM::setTimeInter-
 val (C++ function), 305
 dv::io::camera::DVXplorerM::shutdownCall-
 back (C++ function), 311
 dv::io::camera::DVXplorerM::timeInitCall-
 back (C++ function), 311
 dv::io::camera::DVXplorerM::usbDataCall-
 back (C++ function), 311
 dv::io::camera::DVXplorerM::usbDebugCall-
 back (C++ function), 311
 dv::io::camera::DVXplorerM::VR_CON-
 FIG_RELOAD (C++ member), 314
 dv::io::camera::DVXplorerM::VR_CON-
 FIG_STORE (C++ member), 314
 dv::io::camera::imu (C++ type), 619
 dv::io::camera::imu::BoschBMI160Accel-
 DataRate (C++ enum), 621
 dv::io::camera::imu::BoschBMI160Accel-
 DataRate::RATE_12_5HZ (C++ enumera-
 tor), 621
 dv::io::camera::imu::BoschBMI160Accel-
 DataRate::RATE_25HZ (C++ enumerator),
 621
 dv::io::camera::imu::BoschBMI160Accel-

```

    DataRate::RATE_50HZ (C++ enumerator), 621
dv::io::camera::imu::BoschBMI160Accel-
    DataRate::RATE_100HZ (C++ enumerator), 621
dv::io::camera::imu::BoschBMI160Accel-
    DataRate::RATE_200HZ (C++ enumerator), 621
dv::io::camera::imu::BoschBMI160Accel-
    DataRate::RATE_400HZ (C++ enumerator), 621
dv::io::camera::imu::BoschBMI160Accel-
    DataRate::RATE_800HZ (C++ enumerator), 621
dv::io::camera::imu::BoschBMI160Accel-
    DataRate::RATE_1600HZ (C++ enumerator), 621
dv::io::camera::imu::BoschBMI160AccelFil-
    ter (C++ enum), 621
dv::io::camera::imu::BoschBMI160AccelFil-
    ter::FILTER_NORMAL (C++ enumerator), 621
dv::io::camera::imu::BoschBMI160AccelFil-
    ter::FILTER_OSR2 (C++ enumerator), 621
dv::io::camera::imu::BoschBMI160AccelFil-
    ter::FILTER_OSR4 (C++ enumerator), 621
dv::io::camera::imu::BoschBMI160AccelRange
    (C++ enum), 620
dv::io::camera::imu::BoschBMI160Accel-
    Range::RANGE_2G (C++ enumerator), 620
dv::io::camera::imu::BoschBMI160Accel-
    Range::RANGE_4G (C++ enumerator), 621
dv::io::camera::imu::BoschBMI160Accel-
    Range::RANGE_8G (C++ enumerator), 621
dv::io::camera::imu::BoschBMI160Accel-
    Range::RANGE_16G (C++ enumerator), 621
dv::io::camera::imu::boschBMI160Calcu-
    lateIMUAccelScale (C++ function), 623
dv::io::camera::imu::boschBMI160Calcu-
    lateIMUGyroScale (C++ function), 623
dv::io::camera::imu::BoschBMI160Gyro-
    DataRate (C++ enum), 622
dv::io::camera::imu::BoschBMI160Gyro-
    DataRate::RATE_25HZ (C++ enumerator), 622
dv::io::camera::imu::BoschBMI160Gyro-
    DataRate::RATE_50HZ (C++ enumerator), 622
dv::io::camera::imu::BoschBMI160Gyro-
    DataRate::RATE_100HZ (C++ enumerator), 622
    DataRate::RATE_200HZ (C++ enumerator), 622
    DataRate::RATE_400HZ (C++ enumerator), 622
    DataRate::RATE_800HZ (C++ enumerator), 622
    DataRate::RATE_1600HZ (C++ enumerator), 622
    DataRate::RATE_3200HZ (C++ enumerator), 622
dv::io::camera::imu::BoschBMI160GyroFilter
    (C++ enum), 622
dv::io::camera::imu::BoschBMI160GyroFil-
    ter::FILTER_NORMAL (C++ enumerator), 622
dv::io::camera::imu::BoschBMI160GyroFil-
    ter::FILTER_OSR2 (C++ enumerator), 622
dv::io::camera::imu::BoschBMI160GyroFil-
    ter::FILTER_OSR4 (C++ enumerator), 622
dv::io::camera::imu::BoschBMI160GyroRange
    (C++ enum), 621
dv::io::camera::imu::BoschBMI160GyroR-
    ange::RANGE_125DPS (C++ enumerator), 622
dv::io::camera::imu::BoschBMI160GyroR-
    ange::RANGE_250DPS (C++ enumerator), 622
dv::io::camera::imu::BoschBMI160GyroR-
    ange::RANGE_500DPS (C++ enumerator), 622
dv::io::camera::imu::BoschBMI160GyroR-
    ange::RANGE_1000DPS (C++ enumerator), 622
dv::io::camera::imu::BoschBMI160GyroR-
    ange::RANGE_2000DPS (C++ enumerator), 621
dv::io::camera::imu::ImuModel (C++ enum), 620
dv::io::camera::imu::Imu-
    Model::IMU_BOSCH_BMI_160 (C++ enumerator), 620
dv::io::camera::imu::Imu-
    Model::IMU_BOSCH_BMI_270 (C++ enumerator), 620
dv::io::camera::imu::ImuModel::IMU_IN-
    VENSENSE_6050_6150 (C++ enumerator), 620
dv::io::camera::imu::ImuModel::IMU_IN-

```

VENSENSE_6500_9250 (C++ *enumerator*), 620
 dv::io::camera::imu::ImuModel::IMU_NONE (C++ *enumerator*), 620
 dv::io::camera::imu::InvensenseAccelRange (C++ *enum*), 620
 dv::io::camera::imu::InvensenseAccelRange::RANGE_2G (C++ *enumerator*), 620
 dv::io::camera::imu::InvensenseAccelRange::RANGE_4G (C++ *enumerator*), 620
 dv::io::camera::imu::InvensenseAccelRange::RANGE_8G (C++ *enumerator*), 620
 dv::io::camera::imu::InvensenseAccelRange::RANGE_16G (C++ *enumerator*), 620
 dv::io::camera::imu::InvensenseGyroRange (C++ *enum*), 620
 dv::io::camera::imu::InvensenseGyroRange::RANGE_250DPS (C++ *enumerator*), 620
 dv::io::camera::imu::InvensenseGyroRange::RANGE_500DPS (C++ *enumerator*), 620
 dv::io::camera::imu::InvensenseGyroRange::RANGE_1000DPS (C++ *enumerator*), 620
 dv::io::camera::imu::InvensenseGyroRange::RANGE_2000DPS (C++ *enumerator*), 620
 dv::io::camera::imu::operator<< (C++ *function*), 623
 dv::io::camera::imu::safeFlip16 (C++ *function*), 623
 dv::io::camera::open (C++ *function*), 617, 618
 dv::io::camera::openSync (C++ *function*), 618, 619
 dv::io::camera::operator<< (C++ *function*), 617
 dv::io::camera::parser (C++ *type*), 623
 dv::io::camera::parser::DAVIS (C++ *type*), 623
 dv::io::camera::parser::DAVIS::ColorMode (C++ *enum*), 623
 dv::io::camera::parser::DAVIS::ColorMode::DEFAULT (C++ *enumerator*), 623
 dv::io::camera::parser::DAVIS::ColorMode::GRAYSCALE (C++ *enumerator*), 623
 dv::io::camera::parser::DAVIS::ColorMode::ORIGINAL (C++ *enumerator*), 623
 dv::io::camera::parser::DAVIS::ColorMode::ORIGINAL_SPLIT (C++ *enumerator*), 624
 dv::io::camera::parser::DAVIS::operator<< (C++ *function*), 624
 dv::io::camera::parser::DAVIS::Parser (C++ *class*), 456
 dv::io::camera::parser::DAVIS::Parser::accelScale (C++ *member*), 459
 dv::io::camera::parser::DAVIS::Parser::APS_RESET_CUTOFF (C++ *member*), 460
 dv::io::camera::parser::DAVIS::Parser::apsEndFrame (C++ *function*), 460
 dv::io::camera::parser::DAVIS::Parser::apsInitFrame (C++ *function*), 460
 dv::io::camera::parser::DAVIS::Parser::ApsReadoutType (C++ *enum*), 459
 dv::io::camera::parser::DAVIS::Parser::ApsReadoutType::RESET (C++ *enumerator*), 459
 dv::io::camera::parser::DAVIS::Parser::ApsReadoutType::SIGNAL (C++ *enumerator*), 459
 dv::io::camera::parser::DAVIS::Parser::apsROIUpdateSizes (C++ *function*), 460
 dv::io::camera::parser::DAVIS::Parser::apsUpdateFrame (C++ *function*), 460
 dv::io::camera::parser::DAVIS::Parser::calculateIMUAccelScale (C++ *function*), 460
 dv::io::camera::parser::DAVIS::Parser::calculateIMUGyroScale (C++ *function*), 460
 dv::io::camera::parser::DAVIS::Parser::CDavisOffset (C++ *enum*), 459
 dv::io::camera::parser::DAVIS::Parser::CDavisOffset::DECREASING (C++ *enumerator*), 459
 dv::io::camera::parser::DAVIS::Parser::CDavisOffset::INCREASING (C++ *enumerator*), 459
 dv::io::camera::parser::DAVIS::Parser::cDavisSupport (C++ *member*), 459
 dv::io::camera::parser::DAVIS::Parser::cdavisUpdateFrame (C++ *function*), 460
 dv::io::camera::parser::DAVIS::Parser::colorFilterControl (C++ *member*), 457
 dv::io::camera::parser::DAVIS::Parser::colorFilterSync (C++ *member*), 458
 dv::io::camera::parser::DAVIS::Parser::colorMode (C++ *member*), 457
 dv::io::camera::parser::DAVIS::Parser::convertAllOn (C++ *member*), 457
 dv::io::camera::parser::DAVIS::Parser::count (C++ *member*), 459
 dv::io::camera::parser::DAVIS::Parser::countX (C++ *member*), 458
 dv::io::camera::parser::DAVIS::Parser::countY (C++ *member*), 458

dv::io::camera::parser::DAVIS::Parser::currentEvent (C++ member), 459	dv::io::camera::parser::DAVIS::Parser::getIMUFlipZ (C++ function), 457
dv::io::camera::parser::DAVIS::Parser::currentFrameExposure (C++ member), 458	dv::io::camera::parser::DAVIS::Parser::globalShutter (C++ member), 458
dv::io::camera::parser::DAVIS::Parser::currentReadoutType (C++ member), 458	dv::io::camera::parser::DAVIS::Parser::gyroScale (C++ member), 459
dv::io::camera::parser::DAVIS::Parser::direction (C++ member), 459	dv::io::camera::parser::DAVIS::Parser::IMU_TOTAL_COUNT (C++ member), 460
dv::io::camera::parser::DAVIS::Parser::expectedCountX (C++ member), 458	dv::io::camera::parser::DAVIS::Parser::IMU_TYPE_ACCEL (C++ member), 460
dv::io::camera::parser::DAVIS::Parser::expectedCountY (C++ member), 458	dv::io::camera::parser::DAVIS::Parser::IMU_TYPE_GYRO (C++ member), 460
dv::io::camera::parser::DAVIS::Parser::exposure (C++ member), 458	dv::io::camera::parser::DAVIS::Parser::IMU_TYPE_TEMP (C++ member), 460
dv::io::camera::parser::DAVIS::Parser::flipX (C++ member), 458	dv::io::camera::parser::DAVIS::Parser::invertXY (C++ member), 457
dv::io::camera::parser::DAVIS::Parser::flipXControl (C++ member), 457	dv::io::camera::parser::DAVIS::Parser::lastY (C++ member), 457
dv::io::camera::parser::DAVIS::Parser::flipXSync (C++ member), 457	dv::io::camera::parser::DAVIS::Parser::mDvs (C++ member), 460
dv::io::camera::parser::DAVIS::Parser::flipY (C++ member), 458	dv::io::camera::parser::DAVIS::Parser::mFrame (C++ member), 460
dv::io::camera::parser::DAVIS::Parser::flipYControl (C++ member), 457	dv::io::camera::parser::DAVIS::Parser::mImu (C++ member), 460
dv::io::camera::parser::DAVIS::Parser::flipYSync (C++ member), 457	dv::io::camera::parser::DAVIS::Parser::mModel (C++ member), 460
dv::io::camera::parser::DAVIS::Parser::flipZ (C++ member), 459	dv::io::camera::parser::DAVIS::Parser::model (C++ member), 459
dv::io::camera::parser::DAVIS::Parser::flipZControl (C++ member), 459	dv::io::camera::parser::DAVIS::Parser::mTimestamps (C++ member), 460
dv::io::camera::parser::DAVIS::Parser::frameDebayer (C++ function), 460	dv::io::camera::parser::DAVIS::Parser::parseData (C++ function), 457
dv::io::camera::parser::DAVIS::Parser::frameFixCDavis (C++ function), 460	dv::io::camera::parser::DAVIS::Parser::Parser (C++ function), 456
dv::io::camera::parser::DAVIS::Parser::frameSplitIntoQuadrants (C++ function), 460	dv::io::camera::parser::DAVIS::Parser::pixels (C++ member), 458
dv::io::camera::parser::DAVIS::Parser::getDVSConvertAllOn (C++ function), 456	dv::io::camera::parser::DAVIS::Parser::positionX (C++ member), 458
dv::io::camera::parser::DAVIS::Parser::getDVSFlipHorizontal (C++ function), 456	dv::io::camera::parser::DAVIS::Parser::positionY (C++ member), 458
dv::io::camera::parser::DAVIS::Parser::getDVSFlipVertical (C++ function), 456	dv::io::camera::parser::DAVIS::Parser::roi (C++ member), 458
dv::io::camera::parser::DAVIS::Parser::getFrameColorFilter (C++ function), 457	dv::io::camera::parser::DAVIS::Parser::setDVSConvertAllOn (C++ function), 456
dv::io::camera::parser::DAVIS::Parser::getFrameColorMode (C++ function), 457	dv::io::camera::parser::DAVIS::Parser::setDVSFlipHorizontal (C++ function), 456
dv::io::camera::parser::DAVIS::Parser::getFrameFlipHorizontal (C++ function), 456	dv::io::camera::parser::DAVIS::Parser::setDVSFlipVertical (C++ function), 456
dv::io::camera::parser::DAVIS::Parser::getFrameFlipVertical (C++ function), 457	dv::io::camera::parser::DAVIS::Parser::setFrameColorMode (C++ function), 457
dv::io::camera::parser::DAVIS::Parser::getIMUFlipX (C++ function), 457	dv::io::camera::parser::DAVIS::Parser::setFrameFlipHorizontal (C++ function), 457
dv::io::camera::parser::DAVIS::Parser::getIMUFlipY (C++ function), 457	dv::io::camera::parser::DAVIS::Parser::setFrameFlipVertical (C++ function), 457

dv::io::camera::parser::DAVIS::Parser::setIMUFlipX (C++ function), 457
 dv::io::camera::parser::DAVIS::Parser::setIMUFlipY (C++ function), 457
 dv::io::camera::parser::DAVIS::Parser::setIMUFlipZ (C++ function), 457
 dv::io::camera::parser::DAVIS::Parser::sizeDevice (C++ member), 457
 dv::io::camera::parser::DAVIS::Parser::sizeUser (C++ member), 457
 dv::io::camera::parser::DAVIS::Parser::sizeX (C++ member), 458
 dv::io::camera::parser::DAVIS::Parser::sizeY (C++ member), 458
 dv::io::camera::parser::DAVIS::Parser::startOfExposureTimestamp (C++ member), 458
 dv::io::camera::parser::DAVIS::Parser::startOfPositionOdd (C++ member), 459
 dv::io::camera::parser::DAVIS::Parser::timestampData (C++ member), 458
 dv::io::camera::parser::DAVIS::Parser::TS_WRAP_ADD (C++ member), 460
 dv::io::camera::parser::DAVIS::Parser::type (C++ member), 459
 dv::io::camera::parser::DAVIS::Parser::update (C++ member), 458
 dv::io::camera::parser::DAVIS::Parser::wrapAdd (C++ member), 457
 dv::io::camera::parser::DAVIS::SensorModel (C++ enum), 623
 dv::io::camera::parser::DAVIS::SensorModel::CDAVIS (C++ enumerator), 623
 dv::io::camera::parser::DAVIS::SensorModel::DAVIS240A (C++ enumerator), 623
 dv::io::camera::parser::DAVIS::SensorModel::DAVIS240B (C++ enumerator), 623
 dv::io::camera::parser::DAVIS::SensorModel::DAVIS240C (C++ enumerator), 623
 dv::io::camera::parser::DAVIS::SensorModel::DAVIS346 (C++ enumerator), 623
 dv::io::camera::parser::DAVIS::SensorModel::DAVIS640 (C++ enumerator), 623
 dv::io::camera::parser::DVS128 (C++ type), 624
 dv::io::camera::parser::DVS128::HEIGHT (C++ member), 624
 dv::io::camera::parser::DVS128::Parser (C++ class), 461
 dv::io::camera::parser::DVS128::Parser::convertAllOn (C++ member), 461
 dv::io::camera::parser::DVS128::Parser::DVS128_PO-LARITY_MASK (C++ member), 461
 dv::io::camera::parser::DVS128::Parser::DVS128_PO-LARITY_SHIFT (C++ member), 461
 dv::io::camera::parser::DVS128::Parser::DVS128_SYNC_EVENTS (C++ member), 462
 dv::io::camera::parser::DVS128::Parser::DVS128_TIMES-TAMP_RESET_MASK (C++ member), 461
 dv::io::camera::parser::DVS128::Parser::DVS128_TIMES-TAMP_WRAP_MASK (C++ member), 461
 dv::io::camera::parser::DVS128::Parser::DVS128_X_ADDR_MASK (C++ member), 462
 dv::io::camera::parser::DVS128::Parser::DVS128_X_ADDR_SHIFT (C++ member), 462
 dv::io::camera::parser::DVS128::Parser::DVS128_Y_ADDR_MASK (C++ member), 462
 dv::io::camera::parser::DVS128::Parser::DVS128_Y_ADDR_SHIFT (C++ member), 461
 dv::io::camera::parser::DVS128::Parser::flipX (C++ member), 461
 dv::io::camera::parser::DVS128::Parser::flipY (C++ member), 461
 dv::io::camera::parser::DVS128::Parser::getDVSConvertAllOn (C++ function), 461
 dv::io::camera::parser::DVS128::Parser::getDVSFlipHorizontal (C++ function), 461
 dv::io::camera::parser::DVS128::Parser::getDVSFlipVertical (C++ function), 461
 dv::io::camera::parser::DVS128::Parser::mDvs (C++ member), 461
 dv::io::camera::parser::DVS128::Parser::mTimestamps (C++ member), 461
 dv::io::camera::parser::DVS128::Parser::parseData (C++ function), 461
 dv::io::camera::parser::DVS128::Parser::Parser (C++ function), 461
 dv::io::camera::parser::DVS128::Parser::setDVSConvertAllOn (C++ function), 461
 dv::io::camera::parser::DVS128::Parser::setDVSFlipHorizontal (C++ function), 461
 dv::io::camera::parser::DVS128::Parser::setDVSFlipVertical (C++ function), 461
 dv::io::camera::parser::DVS128::Parser::TS_WRAP_ADD (C++ member), 462
 dv::io::camera::parser::DVS128::Parser::wrapAdd (C++ member), 461
 dv::io::camera::parser::DVS128::WIDTH (C++ member), 624
 dv::io::camera::parser::DVXplorer (C++ type), 624
 dv::io::camera::parser::DVXplorer::Parser (C++ class), 462
 dv::io::camera::parser::DVXplorer::Parser::accelScale (C++ member), 463
 dv::io::camera::parser::DVX-

plorer::Parser::calculateIMUAccelScale (C++ <i>function</i>), 464	dv::io::camera::parser::DVX-plorer::Parser::IMU_TOTAL_COUNT (C++ <i>member</i>), 464
dv::io::camera::parser::DVX-plorer::Parser::calculateIMUGyroScale (C++ <i>function</i>), 464	dv::io::camera::parser::DVX-plorer::Parser::IMU_TYPE_ACCEL (C++ <i>member</i>), 464
dv::io::camera::parser::DVX-plorer::Parser::count (C++ <i>member</i>), 463	dv::io::camera::parser::DVX-plorer::Parser::IMU_TYPE_GYRO (C++ <i>member</i>), 464
dv::io::camera::parser::DVX-plorer::Parser::currentEvent (C++ <i>member</i>), 463	dv::io::camera::parser::DVX-plorer::Parser::IMU_TYPE_TEMP (C++ <i>member</i>), 464
dv::io::camera::parser::DVX-plorer::Parser::dualBinning (C++ <i>member</i>), 463	dv::io::camera::parser::DVX-plorer::Parser::lastX (C++ <i>member</i>), 463
dv::io::camera::parser::DVX-plorer::Parser::flipX (C++ <i>member</i>), 462	dv::io::camera::parser::DVX-plorer::Parser::lastYG1 (C++ <i>member</i>), 463
dv::io::camera::parser::DVX-plorer::Parser::flipXControl (C++ <i>member</i>), 463	dv::io::camera::parser::DVX-plorer::Parser::lastYG2 (C++ <i>member</i>), 463
dv::io::camera::parser::DVX-plorer::Parser::flipY (C++ <i>member</i>), 463	dv::io::camera::parser::DVX-plorer::Parser::mDvs (C++ <i>member</i>), 463
dv::io::camera::parser::DVX-plorer::Parser::flipYControl (C++ <i>member</i>), 463	dv::io::camera::parser::DVX-plorer::Parser::mImu (C++ <i>member</i>), 463
dv::io::camera::parser::DVX-plorer::Parser::flipZ (C++ <i>member</i>), 463	dv::io::camera::parser::DVX-plorer::Parser::mTimestamps (C++ <i>member</i>), 463
dv::io::camera::parser::DVX-plorer::Parser::flipZControl (C++ <i>member</i>), 463	dv::io::camera::parser::DVX-plorer::Parser::parseData (C++ <i>function</i>), 462
dv::io::camera::parser::DVX-plorer::Parser::getDVSDualBinning (C++ <i>function</i>), 462	dv::io::camera::parser::DVX-plorer::Parser::Parser (C++ <i>function</i>), 462
dv::io::camera::parser::DVX-plorer::Parser::getDVSFlipHorizontal (C++ <i>function</i>), 462	dv::io::camera::parser::DVX-plorer::Parser::setDVSDualBinning (C++ <i>function</i>), 462
dv::io::camera::parser::DVX-plorer::Parser::getDVSFlipVertical (C++ <i>function</i>), 462	dv::io::camera::parser::DVX-plorer::Parser::setDVSFlipHorizontal (C++ <i>function</i>), 462
dv::io::camera::parser::DVX-plorer::Parser::getIMUFlipX (C++ <i>function</i>), 462	dv::io::camera::parser::DVX-plorer::Parser::setDVSFlipVertical (C++ <i>function</i>), 462
dv::io::camera::parser::DVX-plorer::Parser::getIMUFlipY (C++ <i>function</i>), 462	dv::io::camera::parser::DVX-plorer::Parser::setIMUFlipX (C++ <i>function</i>), 462
dv::io::camera::parser::DVX-plorer::Parser::getIMUFlipZ (C++ <i>function</i>), 462	dv::io::camera::parser::DVX-plorer::Parser::setIMUFlipY (C++ <i>function</i>), 462
dv::io::camera::parser::DVX-plorer::Parser::gyroScale (C++ <i>member</i>), 463	dv::io::camera::parser::DVX-plorer::Parser::setIMUFlipZ (C++ <i>function</i>), 462

dv::io::camera::parser::DVX-
 plorer::Parser::size (C++ member),
 462

dv::io::camera::parser::DVX-
 plorer::Parser::tmpData (C++ member),
 463

dv::io::camera::parser::DVX-
 plorer::Parser::TS_WRAP_ADD (C++
 member), 464

dv::io::camera::parser::DVX-
 plorer::Parser::type (C++ member),
 463

dv::io::camera::parser::DVX-
 plorer::Parser::wrapAdd (C++ member),
 462

dv::io::camera::parser::ParsedData (C++
 struct), 456

dv::io::camera::parser::ParsedData::clear
 (C++ function), 456

dv::io::camera::parser::ParsedData::events
 (C++ member), 456

dv::io::camera::parser::ParsedData::frames
 (C++ member), 456

dv::io::camera::parser::ParsedData::imu
 (C++ member), 456

dv::io::camera::parser::ParsedData::trig-
 gers (C++ member), 456

dv::io::camera::parser::ParserBase (C++
 class), 468

dv::io::camera::parser::Parser-
 Base::~~ParserBase (C++ function),
 469

dv::io::camera::parser::ParserBase::adjust
 (C++ member), 469

dv::io::camera::parser::ParserBase::ad-
 justTimestamps (C++ function), 469

dv::io::camera::parser::ParserBase::com-
 mitInterval (C++ member), 469

dv::io::camera::parser::ParserBase::cur-
 rent (C++ member), 469

dv::io::camera::parser::ParserBase::data-
 Commit (C++ function), 469

dv::io::camera::parser::ParserBase::eDat-
 aLost (C++ struct), 315

dv::io::camera::parser::Parser-
 Base::getCurrentTimestamp (C++
 function), 469

dv::io::camera::parser::Parser-
 Base::getOutstandingTimestampAd-
 justment (C++ function), 469

dv::io::camera::parser::ParserBase::get-
 SystemOffset (C++ function), 469

dv::io::camera::parser::ParserBase::get-
 TimeInterval (C++ function), 469

dv::io::camera::parser::ParserBase::last-
 Device (C++ member), 469

dv::io::camera::parser::Parser-
 Base::mBuffers (C++ member), 470

dv::io::camera::parser::ParserBase::mData-
 CommitCallback (C++ member), 470

dv::io::camera::parser::ParserBase::mLog-
 ger (C++ member), 470

dv::io::camera::parser::Parser-
 Base::mTimeInitCallback (C++ member),
 470

dv::io::camera::parser::Parser-
 Base::mTimestamps (C++ member), 470

dv::io::camera::parser::Parser-
 Base::nextCommit (C++ member), 469

dv::io::camera::parser::ParserBase::parse-
 Data (C++ function), 469

dv::io::camera::parser::ParserBase::reset-
 Timing (C++ function), 470

dv::io::camera::parser::ParserBase::reset-
 TimingAsync (C++ member), 469

dv::io::camera::parser::ParserBase::set-
 DataCommitCallback (C++ function),
 469

dv::io::camera::parser::ParserBase::set-
 Logger (C++ function), 469

dv::io::camera::parser::ParserBase::set-
 SystemOffset (C++ function), 469

dv::io::camera::parser::ParserBase::set-
 TimeInitCallback (C++ function), 469

dv::io::camera::parser::ParserBase::set-
 TimeInterval (C++ function), 469

dv::io::camera::parser::ParserBase::syste-
 mOffset (C++ member), 469

dv::io::camera::parser::ParserBase::times-
 tampInit (C++ function), 470

dv::io::camera::parser::ParserBase::up-
 dateTimestamp (C++ function), 469

dv::io::camera::parser::ParserDataCommit-
 Callback (C++ type), 623

dv::io::camera::parser::ParserLoggerCall-
 back (C++ type), 623

dv::io::camera::parser::ParserTimeInit-
 Callback (C++ type), 623

dv::io::camera::parser::S5K231Y (C++ type),
 624

dv::io::camera::parser::S5K231Y::HEIGHT
 (C++ member), 624

dv::io::camera::parser::S5K231Y::Parser
 (C++ class), 464

dv::io::camera::parser::S5K231Y::Parser::aFrameS-
 tart (C++ function), 465

dv::io::camera::parser::S5K231Y::Parser::aGen-
 erateEventsFromSMGroup (C++ function),

465 (C++ member), 321

dv::io::camera::parser::S5K231Y::Parser::aRe-dv::io::camera::parser::S5K231Y::Parser::EVENT_GROUPS_FA
setEventFrame (C++ function), 465 (C++ member), 466

dv::io::camera::parser::S5K231Y::Parser::aSetdv::io::camera::parser::S5K231Y::Parser::EVENT_GROUPS_RE
ColumnData (C++ function), 465 PEAT (C++ member), 466

dv::io::camera::parser::S5K231Y::Parser::aTrans- dv::io::camera::parser::S5K231Y::Parser::flipHor-
formPreviousColumn (C++ function), 465 izontal (C++ member), 464, 465

dv::io::camera::parser::S5K231Y::Parser::aUpdv::io::camera::parser::S5K231Y::Parser::flipVer-
dateTimestampReference (C++ function), tical (C++ member), 464, 465
465 dv::io::camera::parser::S5K231Y::Parser::get-
dv::io::camera::parser::S5K231Y::Parser::bitIn- FlipHorizontal (C++ function), 464
dexToOutput (C++ function), 465 dv::io::camera::parser::S5K231Y::Parser::get-
dv::io::camera::parser::S5K231Y::Parser::column- FlipVertical (C++ function), 464
nAddress (C++ member), 464 dv::io::camera::parser::S5K231Y::Parser::gIs-
dv::io::camera::parser::S5K231Y::Parser::column- FrameStart (C++ function), 466
nEvents (C++ member), 464 dv::io::camera::parser::S5K231Y::Parser::gIs-
dv::io::camera::parser::S5K231Y::Parser::eCol- GroupParsedAlready (C++ function), 466
umn (C++ struct), 314 dv::io::camera::parser::S5K231Y::Parser::gIs-
dv::io::camera::parser::S5K231Y::Parser::eCol- IncreasingColumnAddress (C++ function),
umn::columnAddress (C++ member), 315 466

dv::io::camera::parser::S5K231Y::Parser::eCol- dv::io::camera::parser::S5K231Y::Parser::gIs-
umn::eColumn (C++ function), 314 IncreasingTimestamp (C++ function), 465
dv::io::camera::parser::S5K231Y::Parser::eCol- dv::io::camera::parser::S5K231Y::Parser::gIs-
umn::startOfFrame (C++ member), 315 SameSubTimestamp (C++ function), 465
dv::io::camera::parser::S5K231Y::Parser::eCol- dv::io::camera::parser::S5K231Y::Parser::group-
umn::timestampSubUnit (C++ member), sUsed (C++ member), 464
315 dv::io::camera::parser::S5K231Y::Parser::in-
dv::io::camera::parser::S5K231Y::Parser::eS- jectImu (C++ function), 464
MGroup (C++ struct), 320 dv::io::camera::parser::S5K231Y::Parser::last-
dv::io::camera::parser::S5K231Y::Parser::eS- CommittedEvents (C++ member), 464
MGroup::eSMGroup (C++ function), 320 dv::io::camera::parser::S5K231Y::Parser::las-
dv::io::camera::parser::S5K231Y::Parser::eS- tReference (C++ member), 465
MGroup::group1Address (C++ member), dv::io::camera::parser::S5K231Y::Parser::las-
320 tUsedReference (C++ member), 465
dv::io::camera::parser::S5K231Y::Parser::eS- dv::io::camera::parser::S5K231Y::Parser::las-
MGroup::group1Events (C++ member), 320 tUsedSub (C++ member), 465
dv::io::camera::parser::S5K231Y::Parser::eS- dv::io::camera::parser::S5K231Y::Parser::mCon-
MGroup::group1Polarity (C++ member), trols (C++ member), 466
320 dv::io::camera::parser::S5K231Y::Parser::mDvs
dv::io::camera::parser::S5K231Y::Parser::eS- (C++ member), 466
MGroup::group2Address (C++ member), dv::io::camera::parser::S5K231Y::Parser::mState
320 (C++ member), 466
dv::io::camera::parser::S5K231Y::Parser::eS- dv::io::camera::parser::S5K231Y::Parser::mTimes-
MGroup::group2Events (C++ member), 320 tamps (C++ member), 466
dv::io::camera::parser::S5K231Y::Parser::eS- dv::io::camera::parser::S5K231Y::Parser::NUM_GROUPS
MGroup::group2Polarity (C++ member), (C++ member), 466
320 dv::io::camera::parser::S5K231Y::Parser::parse-
dv::io::camera::parser::S5K231Y::Parser::eTimes- Data (C++ function), 464
tampReference (C++ struct), 321 dv::io::camera::parser::S5K231Y::Parser::Parser
dv::io::camera::parser::S5K231Y::Parser::eTimes- (C++ function), 464
tampReference::eTimestampReference dv::io::camera::parser::S5K231Y::Parser::pro-
(C++ function), 321 cessEvent (C++ function), 466
dv::io::camera::parser::S5K231Y::Parser::eTimes- dv::io::camera::parser::S5K231Y::Parser::ref-
tampReference::timestampReference erence (C++ member), 465

dv::io::camera::parser::S5KRC1S::Parser::EVENT_GROUP SAME_SUB_UNIT (C++ enumerator), 467
 PEAT (C++ member), 468 dv::io::camera::parser::S5KRC1S::WIDTH (C++ member), 624
 dv::io::camera::parser::S5KRC1S::Parser::frameNumber (C++ member), 467 dv::io::camera::SyncCameraInputBase (C++ class), 533
 dv::io::camera::parser::S5KRC1S::Parser::gIsFrameStart (C++ function), 468 dv::io::camera::SyncCameraInputBase::gotReset (C++ function), 534
 dv::io::camera::parser::S5KRC1S::Parser::gIsGroupParsedAlready (C++ function), 468 dv::io::camera::SyncCameraInputBase::isMaster (C++ function), 533
 dv::io::camera::parser::S5KRC1S::Parser::gIsSameFrameNumber (C++ function), 468 dv::io::camera::SyncCameraInputBase::isSynchronized (C++ function), 533
 dv::io::camera::parser::S5KRC1S::Parser::groupsUsed (C++ member), 467 dv::io::camera::SyncCameraInputBase::mSyncState (C++ member), 534
 dv::io::camera::parser::S5KRC1S::Parser::gVerifyFrameNumber (C++ function), 468 dv::io::camera::SyncCameraInputBase::sendTimestampReset (C++ function), 534
 dv::io::camera::parser::S5KRC1S::Parser::injectImu (C++ function), 466 dv::io::camera::SyncCameraInputBase::setTimestampOffset (C++ function), 534
 dv::io::camera::parser::S5KRC1S::Parser::lastCommittedEvents (C++ member), 467 dv::io::camera::SyncCameraInputBase::synchronizeWith (C++ function), 533
 dv::io::camera::parser::S5KRC1S::Parser::lastReference (C++ member), 467 dv::io::camera::SyncCameraInputBase::SyncState (C++ enum), 534
 dv::io::camera::parser::S5KRC1S::Parser::mDvs (C++ member), 468 dv::io::camera::SyncCameraInputBase::SyncState::GOT_RESET (C++ enumerator), 534
 dv::io::camera::parser::S5KRC1S::Parser::mirrorMode (C++ member), 467 dv::io::camera::SyncCameraInputBase::SyncState::SYNC_OK (C++ enumerator), 534
 dv::io::camera::parser::S5KRC1S::Parser::mState (C++ member), 468 dv::io::camera::SyncCameraInputBase::SyncState::WAIT_RESET (C++ enumerator), 534
 dv::io::camera::parser::S5KRC1S::Parser::mTimestamps (C++ member), 468 dv::io::camera::SyncCameraInputBase::waitForTimestampReset (C++ function), 534
 dv::io::camera::parser::S5KRC1S::Parser::NUM_GROUPS (C++ member), 468 dv::io::camera::SyncCameraPtr (C++ type), 616
 dv::io::camera::parser::S5KRC1S::Parser::parseData (C++ function), 466 dv::io::camera::synchronizeAny (C++ function), 619
 dv::io::camera::parser::S5KRC1S::Parser::processEvent (C++ function), 468 dv::io::camera::synchronizeAnyTwo (C++ function), 619
 dv::io::camera::parser::S5KRC1S::Parser::reference (C++ member), 467 dv::io::camera::USBDevice (C++ class), 558
 dv::io::camera::parser::S5KRC1S::Parser::referenceOverflow (C++ member), 467 dv::io::camera::USBDevice::~~USBDevice (C++ function), 560
 dv::io::camera::parser::S5KRC1S::Parser::resetIndex (C++ member), 467 dv::io::camera::USBDevice::cleanupBuffers (C++ function), 560
 dv::io::camera::parser::S5KRC1S::Parser::State::WAIT_FOR_RESET (C++ enumerator), 467 dv::io::camera::USBDevice::compatibleCameraCallbackType (C++ type), 559
 UMN_OR_SMGROUP_OR_FRAME_END (C++ enumerator), 467 dv::io::camera::USBDevice::controlInBuffer (C++ member), 231
 dv::io::camera::parser::S5KRC1S::Parser::State::WAIT_FOR_RESET (C++ enumerator), 467 dv::io::camera::USBDevice::controlInBuffer (C++ member), 231
 dv::io::camera::parser::S5KRC1S::Parser::State::WAIT_FOR_RESET (C++ enumerator), 467 dv::io::camera::USBDevice::controlInBuffer (C++ member), 231
 TAMP_REFERENCE (C++ enumerator), 467 dv::io::camera::USBDevice::controlInBuffer (C++ member), 231
 dv::io::camera::parser::S5KRC1S::Parser::State::WAIT_FOR_RESET (C++ enumerator), 467 dv::io::camera::USBDevice::controlInBuffer (C++ member), 231

dv::io::camera::USBDevice::controlInCall-
 backType (C++ type), 560
 dv::io::camera::USBDevice::controlOut-
 Buffer (C++ struct), 231
 dv::io::camera::USBDevice::controlOut-
 Buffer::buffer (C++ member), 231
 dv::io::camera::USBDevice::controlOut-
 Buffer::callback (C++ member), 231
 dv::io::camera::USBDevice::controlOut-
 Buffer::setup (C++ member), 231
 dv::io::camera::USBDevice::controlOutCall-
 backType (C++ type), 559
 dv::io::camera::USBDevice::DEBUG_ENDPOINT
 (C++ member), 564
 dv::io::camera::USBDevice::DEBUG_TRANS-
 FERS_NUMBER (C++ member), 564
 dv::io::camera::USBDevice::DEBUG_TRANS-
 FERS_SIZE (C++ member), 565
 dv::io::camera::USBDevice::DE-
 FAULT_DATA_ENDPOINT (C++ member),
 564
 dv::io::camera::USBDevice::DeviceDescrip-
 tor (C++ struct), 275
 dv::io::camera::USBDevice::DeviceDescrip-
 tor::busNumber (C++ member), 276
 dv::io::camera::USBDevice::DeviceDescrip-
 tor::cameraModel (C++ member), 276
 dv::io::camera::USBDevice::DeviceDescrip-
 tor::devAddress (C++ member), 276
 dv::io::camera::USBDevice::DeviceDescrip-
 tor::deviceType (C++ member), 275
 dv::io::camera::USBDevice::DeviceDescrip-
 tor::firmwareVersion (C++ member),
 276
 dv::io::camera::USBDevice::DeviceDescrip-
 tor::pid (C++ member), 275
 dv::io::camera::USBDevice::DeviceDescrip-
 tor::serialNumber (C++ member), 275
 dv::io::camera::USBDevice::DeviceDescrip-
 tor::vid (C++ member), 275
 dv::io::camera::USBDevice::errorPrint (C++
 function), 564
 dv::io::camera::USBDevice::fetchSerialNum-
 ber (C++ function), 564
 dv::io::camera::USBDevice::filterCompati-
 bleDevices (C++ function), 562
 dv::io::camera::USBDevice::findCompati-
 bleDevices (C++ function), 562
 dv::io::camera::USBDevice::generateRe-
 peatableSerialNumber (C++ function),
 564
 dv::io::camera::USBDevice::getConnection-
 Speed (C++ function), 559
 dv::io::camera::USBDevice::getDataEndpoint
 (C++ function), 561
 dv::io::camera::USBDevice::getDe-
 viceDescriptor (C++ function), 558
 dv::io::camera::USBDevice::getDeviceName
 (C++ function), 559
 dv::io::camera::USBDevice::getFirmwareVer-
 sion (C++ function), 558
 dv::io::camera::USBDevice::getLogLevel (C++
 function), 559
 dv::io::camera::USBDevice::getSerialNumber
 (C++ function), 558
 dv::io::camera::USBDevice::getTransfer-
 sNumber (C++ function), 559
 dv::io::camera::USBDevice::getTransfers-
 Size (C++ function), 559
 dv::io::camera::USBDevice::isUSBDataTrans-
 ferRunning (C++ function), 561
 dv::io::camera::USBDevice::isUSBThreadRun-
 ning (C++ function), 561
 dv::io::camera::USBDevice::LIBUSB_LOG_MAP
 (C++ member), 565
 dv::io::camera::USBDe-
 vice::LIBUSB_LOG_MAP_LOCK (C++
 member), 565
 dv::io::camera::USBDevice::libusbCon-
 textDeleter (C++ member), 565
 dv::io::camera::USBDevice::libusbDevice-
 HandleDeleter (C++ member), 565
 dv::io::camera::USBDevice::libusbDe-
 viceListDeleter (C++ member), 565
 dv::io::camera::USBDevice::LibusbManaged-
 Context (C++ type), 562
 dv::io::camera::USBDevice::LibusbManaged-
 DeviceHandle (C++ type), 562
 dv::io::camera::USBDevice::LibusbManaged-
 DeviceList (C++ type), 562
 dv::io::camera::USBDevice::LibusbManaged-
 Transfer (C++ type), 562
 dv::io::camera::USBDevice::libusbTransfer-
 Deleter (C++ member), 565
 dv::io::camera::USBDevice::log (C++ function),
 561
 dv::io::camera::USBDevice::loggerCallback-
 Type (C++ type), 558
 dv::io::camera::USBDevice::LogLevel (C++
 enum), 558
 dv::io::camera::USBDe-
 vice::LogLevel::LVL_DEBUG (C++
 enumerator), 558
 dv::io::camera::USBDe-
 vice::LogLevel::LVL_ERROR (C++
 enumerator), 558
 dv::io::camera::USBDe-
 vice::LogLevel::LVL_INFO (C++ enumer-

ator), 558

dv::io::camera::USBDevice::LogLevel::LVL_NONE (C++ enumerator), 558

dv::io::camera::USBDevice::LogLevel::LVL_WARNING (C++ enumerator), 558

dv::io::camera::USBDevice::MAX_CONTROL_TRANSFER_SIZE (C++ member), 564

dv::io::camera::USBDevice::MAX_SERIAL_NUMBER_LENGTH (C++ member), 564

dv::io::camera::USBDevice::mDataEndPoint (C++ member), 563

dv::io::camera::USBDevice::mDataTransfersActiveNLCK (C++ member), 563

dv::io::camera::USBDevice::mDataTransfersFailedNLCK (C++ member), 563

dv::io::camera::USBDevice::mDataTransferLock (C++ member), 563

dv::io::camera::USBDevice::mDataTransferNLCK (C++ member), 563

dv::io::camera::USBDevice::mDataTransferSizeNLCK (C++ member), 563

dv::io::camera::USBDevice::mDataTransferRun (C++ member), 563

dv::io::camera::USBDevice::mDataTransferSizeNLCK (C++ member), 563

dv::io::camera::USBDevice::mDebugTransfers (C++ member), 564

dv::io::camera::USBDevice::mDebugTransferActive (C++ member), 564

dv::io::camera::USBDevice::mDescriptor (C++ member), 563

dv::io::camera::USBDevice::mDeviceContext (C++ member), 563

dv::io::camera::USBDevice::mDeviceHandle (C++ member), 563

dv::io::camera::USBDevice::mUsbDataCallback (C++ member), 564

dv::io::camera::USBDevice::mUsbDebugCallback (C++ member), 564

dv::io::camera::USBDevice::mUsbDeviceLogger (C++ member), 563

dv::io::camera::USBDevice::mUsbDeviceName (C++ member), 563

dv::io::camera::USBDevice::mUsbDeviceNameLock (C++ member), 563

dv::io::camera::USBDevice::mUsbLogLevel (C++ member), 563

dv::io::camera::USBDevice::mUsbOpsLock (C++ member), 563

dv::io::camera::USBDevice::mUsbShutdownCallback (C++ member), 564

dv::io::camera::USBDevice::mUsbThread (C++ member), 563

dv::io::camera::USBDevice::mUsbThreadName (C++ member), 563

dv::io::camera::USBDevice::mUsbThreadRun (C++ member), 563

dv::io::camera::USBDevice::operator<< (C++ function), 565

dv::io::camera::USBDevice::resetCameraName (C++ function), 561

dv::io::camera::USBDevice::setDataCallback (C++ function), 561

dv::io::camera::USBDevice::setDataEndpoint (C++ function), 561

dv::io::camera::USBDevice::setDebugCallback (C++ function), 561

dv::io::camera::USBDevice::setLogLevel (C++ function), 559

dv::io::camera::USBDevice::setShutdownCallback (C++ function), 561

dv::io::camera::USBDevice::setTransferSize (C++ function), 559

dv::io::camera::USBDevice::setTransfersSize (C++ function), 559

dv::io::camera::USBDevice::spiConfigReceive (C++ function), 561

dv::io::camera::USBDevice::spiConfigReceiveAsync (C++ function), 561

dv::io::camera::USBDevice::spiConfigSend (C++ function), 561

dv::io::camera::USBDevice::spiConfigSendAsync (C++ function), 561

dv::io::camera::USBDevice::spiConfigSendMultiple (C++ function), 560

dv::io::camera::USBDevice::spiConfigSendMultipleAsync (C++ function), 561

dv::io::camera::USBDevice::SPIConfigurationParameters (C++ struct), 516

dv::io::camera::USBDevice::SPIConfigurationParameters::moduleAddress (C++ member), 516

dv::io::camera::USBDevice::SPIConfigurationParameters::parameterAddress (C++ member), 516

dv::io::camera::USBDevice::SPIConfigurationParameters::parameterValue (C++ member), 516

dv::io::camera::USBDevice::usbAllocateDataTransfersNLCK (C++ function), 562

dv::io::camera::USBDevice::usbAllocateDebugTransfers (C++ function), 563

dv::io::camera::USBDevice::usbAllocateTransfers (C++ function), 562

dv::io::camera::USBDevice::usbCancelAnd-

DeallocateDataTransfersNLCK (C++ function), 562
 dv::io::camera::USBDevice::usbCancelAndDeallocateDebugTransfers (C++ function), 563
 dv::io::camera::USBDevice::usbCancelTransfers (C++ function), 562
 dv::io::camera::USBDevice::usbControlTransferIn (C++ function), 560
 dv::io::camera::USBDevice::usbControlTransferInAsync (C++ function), 560
 dv::io::camera::USBDevice::usbControlTransferOut (C++ function), 560
 dv::io::camera::USBDevice::usbControlTransferOutAsync (C++ function), 560
 dv::io::camera::USBDevice::usbDataTransfersStart (C++ function), 562
 dv::io::camera::USBDevice::usbDataTransfersStop (C++ function), 562
 dv::io::camera::USBDevice::usbDebugTransfersStart (C++ function), 562
 dv::io::camera::USBDevice::usbDebugTransfersStop (C++ function), 562
 dv::io::camera::USBDevice::USBDevice (C++ function), 560
 dv::io::camera::USBDevice::usbShutdownCallbackType (C++ type), 560
 dv::io::camera::USBDevice::usbThreadFunc (C++ function), 564
 dv::io::camera::USBDevice::usbThreadStart (C++ function), 562
 dv::io::camera::USBDevice::usbThreadStop (C++ function), 562
 dv::io::camera::USBDevice::usbTransferCallbackType (C++ type), 560
 dv::io::camera::USBDevice::VENDOR_REQUEST_DATA_CLEANUP (C++ member), 565
 dv::io::camera::USBDevice::VENDOR_REQUEST_LOG_LEVEL (C++ member), 565
 dv::io::camera::USBDevice::VENDOR_REQUEST_SPI_CONFIG (C++ member), 562
 dv::io::camera::USBDevice::VENDOR_REQUEST_SPI_CONFIG_MULTIPLE (C++ member), 565
 dv::io::camera::USBDevice::VID_INIVATION (C++ member), 562
 dv::io::camera::USBDeviceNextGen (C++ class), 565
 dv::io::camera::USBDeviceNextGen::spiConfigReceive (C++ function), 566
 dv::io::camera::USBDeviceNextGen::spiConfigReceiveAsync (C++ function), 566
 dv::io::camera::USBDeviceNextGen::spiConfigSend (C++ function), 565
 dv::io::camera::USBDeviceNextGen::spiConfigSendAsync (C++ function), 566
 dv::io::camera::USBDeviceNextGen::spiConfigSendMultiple (C++ function), 565
 dv::io::camera::USBDeviceNextGen::spiConfigSendMultipleAsync (C++ function), 565
 dv::io::camera::USBDeviceNextGen::USBDeviceNextGen (C++ function), 565
 dv::io::camera::USBDeviceType (C++ enum), 617
 dv::io::camera::USBDeviceType::CX3_MIPI (C++ enumerator), 617
 dv::io::camera::USBDeviceType::FX2 (C++ enumerator), 617
 dv::io::camera::USBDeviceType::FX3_BLUE (C++ enumerator), 617
 dv::io::camera::USBDeviceType::FX3_GEN2 (C++ enumerator), 617
 dv::io::camera::USBDeviceType::FX3_MB (C++ enumerator), 617
 dv::io::camera::USBDeviceType::FX3_RED (C++ enumerator), 617
 dv::io::compression (C++ type), 624
 dv::io::compression::CompressionSupport (C++ class), 227
 dv::io::compression::CompressionSupport::~~CompressionSupport (C++ function), 227
 dv::io::compression::CompressionSupport::compress (C++ function), 227
 dv::io::compression::CompressionSupport::CompressionSupport (C++ function), 227
 dv::io::compression::CompressionSupport::getCompressionType (C++ function), 227
 dv::io::compression::CompressionSupport::mType (C++ member), 227
 dv::io::compression::createCompressionSupport (C++ function), 624
 dv::io::compression::createDecompressionSupport (C++ function), 624
 dv::io::compression::DecompressionSupport (C++ class), 270
 dv::io::compression::DecompressionSupport::~~DecompressionSupport (C++ function), 271
 dv::io::compression::DecompressionSupport::decompress (C++ function), 271
 dv::io::compression::DecompressionSupport::DecompressionSupport (C++

function), 271

dv::io::compression::DecompressionSupport::getCompressionType (C++ *function*), 271

dv::io::compression::DecompressionSupport::mType (C++ *member*), 271

dv::io::compression::Lz4CompressionSupport (C++ *class*), 395

dv::io::compression::Lz4CompressionSupport::compress (C++ *function*), 395

dv::io::compression::Lz4CompressionSupport::LZ4_COMPRESSION_CHUNK_SIZE (C++ *member*), 395

dv::io::compression::Lz4CompressionSupport::lz4CompressionPreferences (C++ *member*), 395

dv::io::compression::Lz4CompressionSupport::Lz4CompressionSupport (C++ *function*), 395

dv::io::compression::Lz4CompressionSupport::lz4HighCompressionPreferences (C++ *member*), 395

dv::io::compression::Lz4CompressionSupport::mChunkSize (C++ *member*), 395

dv::io::compression::Lz4CompressionSupport::mContext (C++ *member*), 395

dv::io::compression::Lz4CompressionSupport::mEndSize (C++ *member*), 395

dv::io::compression::Lz4CompressionSupport::mPrefs (C++ *member*), 395

dv::io::compression::Lz4DecompressionSupport (C++ *class*), 395

dv::io::compression::Lz4DecompressionSupport::decompress (C++ *function*), 396

dv::io::compression::Lz4DecompressionSupport::initDecompressionContext (C++ *function*), 396

dv::io::compression::Lz4DecompressionSupport::LZ4_DECOMPRESSION_CHUNK_SIZE (C++ *member*), 396

dv::io::compression::Lz4DecompressionSupport::Lz4DecompressionSupport (C++ *function*), 396

dv::io::compression::Lz4DecompressionSupport::mContext (C++ *member*), 396

dv::io::compression::NoneCompressionSupport (C++ *class*), 450

dv::io::compression::NoneCompressionSupport::compress (C++ *function*), 451

dv::io::compression::NoneCompressionSupport::NoneCompressionSupport (C++ *function*), 451

dv::io::compression::NoneDecompressionSupport (C++ *class*), 451

dv::io::compression::NoneDecompressionSupport::decompress (C++ *function*), 451

dv::io::compression::NoneDecompressionSupport::NoneDecompressionSupport (C++ *function*), 451

dv::io::compression::ZstdCompressionSupport (C++ *class*), 572

dv::io::compression::ZstdCompressionSupport::compress (C++ *function*), 573

dv::io::compression::ZstdCompressionSupport::mContext (C++ *member*), 573

dv::io::compression::ZstdCompressionSupport::mLevel (C++ *member*), 573

dv::io::compression::ZstdCompressionSupport::ZstdCompressionSupport (C++ *function*), 572

dv::io::compression::ZstdDecompressionSupport (C++ *class*), 573

dv::io::compression::ZstdDecompressionSupport::decompress (C++ *function*), 573

dv::io::compression::ZstdDecompressionSupport::initDecompressionContext (C++ *function*), 573

dv::io::compression::ZstdDecompressionSupport::mContext (C++ *member*), 573

dv::io::compression::ZstdDecompressionSupport::ZstdDecompressionSupport (C++ *function*), 573

dv::io::DataReadHandler (C++ *struct*), 231

dv::io::DataReadHandler::eof (C++ *member*), 232

dv::io::DataReadHandler::mEventHandler (C++ *member*), 232

dv::io::DataReadHandler::mFrameHandler (C++ *member*), 232

dv::io::DataReadHandler::mImuHandler (C++ *member*), 232

dv::io::DataReadHandler::mOutputFlagHandler (C++ *member*), 232

dv::io::DataReadHandler::mTriggersHandler (C++ *member*), 232

dv::io::DataReadHandler::operator() (C++ *function*), 231, 232

dv::io::DataReadHandler::OutputFlag (C++ *enum*), 231

dv::io::DataReadHandler::OutputFlag::CONTINUE (C++ *enumerator*), 231

dv::io::DataReadHandler::OutputFlag::END_OF_FILE (C++ *enumerator*), 231

dv::io::DataReadHandler::seek (C++ *member*), 232

`dv::io::DataReadVariant` (C++ type), 615
`dv::io::FileInfo` (C++ struct), 353
`dv::io::FileInfo::mCompression` (C++ member), 353
`dv::io::FileInfo::mDataTable` (C++ member), 353
`dv::io::FileInfo::mDataTablePosition` (C++ member), 353
`dv::io::FileInfo::mDataTableSize` (C++ member), 353
`dv::io::FileInfo::mFileSize` (C++ member), 353
`dv::io::FileInfo::mPerStreamDataTables` (C++ member), 354
`dv::io::FileInfo::mStreams` (C++ member), 354
`dv::io::FileInfo::mTimeDifference` (C++ member), 354
`dv::io::FileInfo::mTimeHighest` (C++ member), 353
`dv::io::FileInfo::mTimeLowest` (C++ member), 353
`dv::io::FileInfo::mTimeShift` (C++ member), 354
`dv::io::InputBase` (C++ class), 372
`dv::io::InputBase::~~InputBase` (C++ function), 373
`dv::io::InputBase::getCameraName` (C++ function), 374
`dv::io::InputBase::getEventResolution` (C++ function), 373
`dv::io::InputBase::getFrameResolution` (C++ function), 373
`dv::io::InputBase::getNextEventBatch` (C++ function), 373
`dv::io::InputBase::getNextFrame` (C++ function), 373
`dv::io::InputBase::getNextImuBatch` (C++ function), 373
`dv::io::InputBase::getNextTriggerBatch` (C++ function), 373
`dv::io::InputBase::isEventStreamAvailable` (C++ function), 373
`dv::io::InputBase::isFrameStreamAvailable` (C++ function), 373
`dv::io::InputBase::isImuStreamAvailable` (C++ function), 373
`dv::io::InputBase::isRunning` (C++ function), 374
`dv::io::InputBase::isRunningAny` (C++ function), 374
`dv::io::InputBase::isStreamAvailable` (C++ function), 374
`dv::io::InputBase::isTriggerStreamAvailable` (C++ function), 374
`dv::io::ModeFlags` (C++ enum), 616
`dv::io::ModeFlags::READ` (C++ enumerator), 616
`dv::io::ModeFlags::WRITE` (C++ enumerator), 616
`dv::io::MonoCameraRecording` (C++ class), 416
`dv::io::MonoCameraRecording::eofReached` (C++ member), 425
`dv::io::MonoCameraRecording::getCameraName` (C++ function), 422
`dv::io::MonoCameraRecording::getDuration` (C++ function), 422
`dv::io::MonoCameraRecording::getEventResolution` (C++ function), 423
`dv::io::MonoCameraRecording::getEventsTimeRange` (C++ function), 419
`dv::io::MonoCameraRecording::getFrameResolution` (C++ function), 423
`dv::io::MonoCameraRecording::getFramesTimeRange` (C++ function), 420
`dv::io::MonoCameraRecording::getImuTimeRange` (C++ function), 420
`dv::io::MonoCameraRecording::getNextEventBatch` (C++ function), 418
`dv::io::MonoCameraRecording::getNextFrame` (C++ function), 417
`dv::io::MonoCameraRecording::getNextImuBatch` (C++ function), 418
`dv::io::MonoCameraRecording::getNextPacket` (C++ function), 425
`dv::io::MonoCameraRecording::getNextStreamPacket` (C++ function), 418
`dv::io::MonoCameraRecording::getNextTriggerBatch` (C++ function), 419
`dv::io::MonoCameraRecording::getStream` (C++ function), 424
`dv::io::MonoCameraRecording::getStreamInfo` (C++ function), 424
`dv::io::MonoCameraRecording::getStreamMetadata` (C++ function), 423
`dv::io::MonoCameraRecording::getStreamMetadataValue` (C++ function), 424
`dv::io::MonoCameraRecording::getStreamNames` (C++ function), 417
`dv::io::MonoCameraRecording::getStreamTimeRange` (C++ function), 420
`dv::io::MonoCameraRecording::getTimeRange` (C++ function), 422
`dv::io::MonoCameraRecording::getTriggerTimeRange` (C++ function), 421
`dv::io::MonoCameraRecording::handleNext` (C++ function), 423
`dv::io::MonoCameraRecording::isEventStreamAvailable` (C++ function), 421
`dv::io::MonoCameraRecording::isFrameStreamAvailable` (C++ function), 421

dv::io::MonoCameraRecording::isImuStreamAvailable (C++ function), 421, 422
 dv::io::MonoCameraRecording::isRunning (C++ function), 419
 dv::io::MonoCameraRecording::isRunningAny (C++ function), 419
 dv::io::MonoCameraRecording::isStreamAvailable (C++ function), 417
 dv::io::MonoCameraRecording::isStreamOfDataType (C++ function), 424
 dv::io::MonoCameraRecording::isTriggerStreamAvailable (C++ function), 422
 dv::io::MonoCameraRecording::mCameraName (C++ member), 425
 dv::io::MonoCameraRecording::mInfo (C++ member), 425
 dv::io::MonoCameraRecording::MonoCameraRecording (C++ function), 417
 dv::io::MonoCameraRecording::mPacketIter (C++ member), 425
 dv::io::MonoCameraRecording::mReader (C++ member), 425
 dv::io::MonoCameraRecording::mStreamInfo (C++ member), 425
 dv::io::MonoCameraRecording::parseStreamIds (C++ function), 424
 dv::io::MonoCameraRecording::readNext (C++ function), 422
 dv::io::MonoCameraRecording::resetSequentialRead (C++ function), 419
 dv::io::MonoCameraRecording::run (C++ function), 423
 dv::io::MonoCameraRecording::StreamDescriptor (C++ struct), 529
 dv::io::MonoCameraRecording::StreamDescriptor::mMetadata (C++ member), 529
 dv::io::MonoCameraRecording::StreamDescriptor::mSeekIndex (C++ member), 529
 dv::io::MonoCameraRecording::StreamDescriptor::mStream (C++ member), 529
 dv::io::MonoCameraRecording::StreamDescriptor::StreamDescriptor (C++ function), 529
 dv::io::MonoCameraRecording::StreamInfoMap (C++ type), 424
 dv::io::MonoCameraRecording::trimVector (C++ function), 425
 dv::io::MonoCameraWriter (C++ class), 425
 dv::io::MonoCameraWriter::~~MonoCameraWriter (C++ function), 430
 dv::io::MonoCameraWriter::CaptureConfig (C++ function), 431
 dv::io::MonoCameraWriter::Config (C++ class), 227
 dv::io::MonoCameraWriter::Config::addEventStream (C++ function), 227
 dv::io::MonoCameraWriter::Config::addFrameStream (C++ function), 228
 dv::io::MonoCameraWriter::Config::addImuStream (C++ function), 228
 dv::io::MonoCameraWriter::Config::addStream (C++ function), 228
 dv::io::MonoCameraWriter::Config::addStreamMetadata (C++ function), 227
 dv::io::MonoCameraWriter::Config::addTriggerStream (C++ function), 228
 dv::io::MonoCameraWriter::Config::cameraName (C++ member), 229
 dv::io::MonoCameraWriter::Config::compression (C++ member), 229
 dv::io::MonoCameraWriter::Config::Config (C++ function), 228
 dv::io::MonoCameraWriter::Config::customDataStreams (C++ member), 229
 dv::io::MonoCameraWriter::Config::customDataStreamsMetadata (C++ member), 229
 dv::io::MonoCameraWriter::Config::findStreamResolution (C++ function), 228
 dv::io::MonoCameraWriter::createHeader (C++ function), 432
 dv::io::MonoCameraWriter::DAVISConfig (C++ function), 431
 dv::io::MonoCameraWriter::DVSCConfig (C++ function), 431
 dv::io::MonoCameraWriter::EventOnlyConfig (C++ function), 430
 dv::io::MonoCameraWriter::findStreamDescriptor (C++ function), 432
 dv::io::MonoCameraWriter::FrameOnlyConfig (C++ function), 431
 dv::io::MonoCameraWriter::getCameraName (C++ function), 430
 dv::io::MonoCameraWriter::inputConfig (C++ member), 432
 dv::io::MonoCameraWriter::isEventStreamConfigured (C++ function), 429
 dv::io::MonoCameraWriter::isFrameStreamConfigured (C++ function), 429
 dv::io::MonoCameraWriter::isImuStreamConfigured (C++ function), 429
 dv::io::MonoCameraWriter::isStreamConfigured (C++ function), 430
 dv::io::MonoCameraWriter::isTriggerStream-

Configured (C++ function), 429
 dv::io::MonoCameraWriter::MonoCameraWriter (C++ function), 425, 432
 dv::io::MonoCameraWriter::mOutput (C++ member), 432
 dv::io::MonoCameraWriter::mOutputStreamDescriptors (C++ member), 432
 dv::io::MonoCameraWriter::mPackagingCount (C++ member), 432
 dv::io::MonoCameraWriter::mRoot (C++ member), 432
 dv::io::MonoCameraWriter::setPackagingCount (C++ function), 429
 dv::io::MonoCameraWriter::StreamDescriptor (C++ struct), 529
 dv::io::MonoCameraWriter::StreamDescriptor::~StreamDescriptor (C++ function), 530
 dv::io::MonoCameraWriter::StreamDescriptor::elementBuffer (C++ member), 530
 dv::io::MonoCameraWriter::StreamDescriptor::freeElementBufferCall (C++ member), 530
 dv::io::MonoCameraWriter::StreamDescriptor::id (C++ member), 530
 dv::io::MonoCameraWriter::StreamDescriptor::lastTimestamp (C++ member), 530
 dv::io::MonoCameraWriter::StreamDescriptor::StreamDescriptor (C++ function), 530
 dv::io::MonoCameraWriter::StreamDescriptor::type (C++ member), 530
 dv::io::MonoCameraWriter::StreamDescriptorMap (C++ type), 432
 dv::io::MonoCameraWriter::validateConfig (C++ function), 432
 dv::io::MonoCameraWriter::writeEventPacket (C++ function), 426
 dv::io::MonoCameraWriter::writeEvents (C++ function), 426, 430
 dv::io::MonoCameraWriter::writeFrame (C++ function), 426, 430
 dv::io::MonoCameraWriter::writeImu (C++ function), 427, 430
 dv::io::MonoCameraWriter::writeImuPacket (C++ function), 427
 dv::io::MonoCameraWriter::writePacket (C++ function), 428
 dv::io::MonoCameraWriter::writePacketElement (C++ function), 428
 dv::io::MonoCameraWriter::writeTrigger (C++ function), 428
 dv::io::MonoCameraWriter::writeTriggerPacket (C++ function), 427
 dv::io::MonoCameraWriter::writeTriggers (C++ function), 430
 dv::io::network (C++ type), 624
 dv::io::network::asioTCP (C++ type), 625
 dv::io::network::asioUNIX (C++ type), 625
 dv::io::network::encrypt (C++ type), 625
 dv::io::network::encrypt::createEncryptionContext (C++ function), 625
 dv::io::network::encrypt::defaultEncryptionClient (C++ function), 625
 dv::io::network::encrypt::defaultEncryptionServer (C++ function), 625
 dv::io::network::SocketBase (C++ class), 509
 dv::io::network::SocketBase::~SocketBase (C++ function), 510
 dv::io::network::SocketBase::close (C++ function), 510
 dv::io::network::SocketBase::CompletionHandler (C++ type), 509
 dv::io::network::SocketBase::isOpen (C++ function), 510
 dv::io::network::SocketBase::read (C++ function), 510
 dv::io::network::SocketBase::syncRead (C++ function), 510
 dv::io::network::SocketBase::syncWrite (C++ function), 510
 dv::io::network::SocketBase::write (C++ function), 510
 dv::io::network::TCPTLSSocket (C++ class), 534
 dv::io::network::TCPTLSSocket::~TCPTLSSocket (C++ function), 535
 dv::io::network::TCPTLSSocket::baseSocket (C++ function), 536
 dv::io::network::TCPTLSSocket::close (C++ function), 535
 dv::io::network::TCPTLSSocket::isOpen (C++ function), 535
 dv::io::network::TCPTLSSocket::isSecured (C++ function), 535
 dv::io::network::TCPTLSSocket::local_address (C++ function), 536
 dv::io::network::TCPTLSSocket::local_endpoint (C++ function), 535
 dv::io::network::TCPTLSSocket::local_port (C++ function), 536
 dv::io::network::TCPTLSSocket::mLocalEndpoint (C++ member), 536
 dv::io::network::TCPTLSSocket::mRemoteEndpoint (C++ member), 536
 dv::io::network::TCPTLSSocket::mSecureConnection (C++ member), 536

dv::io::network::TCPTLSSocket::mSocket (C++ member), 536	Socket::mWriteQueue (C++ member), 568
dv::io::network::TCPTLSSocket::mSocket-Closed (C++ member), 536	dv::io::network::WriteOrderedSocket::read (C++ function), 568
dv::io::network::TCPTLSSocket::read (C++ function), 535	dv::io::network::WriteOrderedSocket::write (C++ function), 568
dv::io::network::TCPTLSSocket::remote_address (C++ function), 536	dv::io::network::WriteOrderedSocket::WriteJob (C++ struct), 566
dv::io::network::TCPTLSSocket::remote_endpoint (C++ function), 536	dv::io::network::WriteOrderedSocket::WriteJob::mBuffer (C++ member), 566
dv::io::network::TCPTLSSocket::remote_port (C++ function), 536	dv::io::network::WriteOrderedSocket::WriteJob::mHandler (C++ member), 566
dv::io::network::TCPTLSSocket::socketType (C++ type), 535	dv::io::network::WriteOrderedSocket::WriteJob::WriteJob (C++ function), 566
dv::io::network::TCPTLSSocket::syncRead (C++ function), 535	dv::io::network::WriteOrderedSocket::WriteOrderedSocket (C++ function), 568
dv::io::network::TCPTLSSocket::syncWrite (C++ function), 535	dv::io::NetworkReader (C++ class), 442
dv::io::network::TCPTLSSocket::TCPTLSSocket (C++ function), 535	dv::io::NetworkReader::~~NetworkReader (C++ function), 443
dv::io::network::TCPTLSSocket::write (C++ function), 535	dv::io::NetworkReader::close (C++ function), 445
dv::io::network::UNIXSocket (C++ class), 556	dv::io::NetworkReader::connectTCP (C++ function), 446
dv::io::network::UNIXSocket::~~UNIXSocket (C++ function), 556	dv::io::NetworkReader::connectUNIX (C++ function), 446
dv::io::network::UNIXSocket::close (C++ function), 556	dv::io::NetworkReader::getCameraName (C++ function), 445
dv::io::network::UNIXSocket::isOpen (C++ function), 556	dv::io::NetworkReader::getEventResolution (C++ function), 444
dv::io::network::UNIXSocket::read (C++ function), 557	dv::io::NetworkReader::getFrameResolution (C++ function), 444
dv::io::network::UNIXSocket::socket (C++ member), 557	dv::io::NetworkReader::getNextEventBatch (C++ function), 443
dv::io::network::UNIXSocket::socketClosed (C++ member), 557	dv::io::NetworkReader::getNextFrame (C++ function), 443
dv::io::network::UNIXSocket::socketType (C++ type), 556	dv::io::NetworkReader::getNextImuBatch (C++ function), 443
dv::io::network::UNIXSocket::syncRead (C++ function), 557	dv::io::NetworkReader::getNextPacket (C++ function), 444
dv::io::network::UNIXSocket::syncWrite (C++ function), 557	dv::io::NetworkReader::getNextTriggerBatch (C++ function), 444
dv::io::network::UNIXSocket::UNIXSocket (C++ function), 556	dv::io::NetworkReader::getStreamDefinition (C++ function), 445
dv::io::network::UNIXSocket::write (C++ function), 556	dv::io::NetworkReader::initializeReader (C++ function), 446
dv::io::network::WriteOrderedSocket (C++ class), 568	dv::io::NetworkReader::isEventStreamAvailable (C++ function), 444
dv::io::network::WriteOrderedSocket::close (C++ function), 568	dv::io::NetworkReader::isFrameStreamAvailable (C++ function), 444
dv::io::network::WriteOrderedSocket::isOpen (C++ function), 568	dv::io::NetworkReader::isImuStreamAvailable (C++ function), 444
dv::io::network::WriteOrderedSocket::mSocket (C++ member), 568	
dv::io::network::WriteOrdered-	

able (C++ function), 444

dv::io::NetworkReader::isRunning (C++ function), 445

dv::io::NetworkReader::isRunningAny (C++ function), 445

dv::io::NetworkReader::isStreamAvailable (C++ function), 445

dv::io::NetworkReader::isTriggerStreamAvailable (C++ function), 444

dv::io::NetworkReader::mAedat4Reader (C++ member), 446

dv::io::NetworkReader::mCameraName (C++ member), 447

dv::io::NetworkReader::mException (C++ member), 447

dv::io::NetworkReader::mExceptionThrown (C++ member), 447

dv::io::NetworkReader::mIOService (C++ member), 446

dv::io::NetworkReader::mKeepReading (C++ member), 447

dv::io::NetworkReader::mPacketQueue (C++ member), 447

dv::io::NetworkReader::mReadHandler (C++ member), 446

dv::io::NetworkReader::mReadingThread (C++ member), 447

dv::io::NetworkReader::mSocket (C++ member), 446

dv::io::NetworkReader::mStream (C++ member), 446

dv::io::NetworkReader::mTLSContext (C++ member), 446

dv::io::NetworkReader::mTLSEnabled (C++ member), 446

dv::io::NetworkReader::NetworkReader (C++ function), 443

dv::io::NetworkReader::PacketQueue (C++ type), 446

dv::io::NetworkReader::readClbk (C++ function), 446

dv::io::NetworkReader::readThread (C++ function), 446

dv::io::NetworkWriter (C++ class), 447

dv::io::NetworkWriter::~~NetworkWriter (C++ function), 448

dv::io::NetworkWriter::acceptStart (C++ function), 449

dv::io::NetworkWriter::Connection (C++ class), 229

dv::io::NetworkWriter::Connection::~~Connection (C++ function), 229

dv::io::NetworkWriter::Connection::close (C++ function), 229

dv::io::NetworkWriter::Connection::Connection (C++ function), 229

dv::io::NetworkWriter::Connection::handleError (C++ function), 229

dv::io::NetworkWriter::Connection::isOpen (C++ function), 229

dv::io::NetworkWriter::Connection::keepAliveByReading (C++ function), 229

dv::io::NetworkWriter::Connection::mKeepAliveReadSpace (C++ member), 230

dv::io::NetworkWriter::Connection::mParent (C++ member), 230

dv::io::NetworkWriter::Connection::mSocket (C++ member), 230

dv::io::NetworkWriter::Connection::start (C++ function), 229

dv::io::NetworkWriter::Connection::writeIOHeader (C++ function), 229

dv::io::NetworkWriter::Connection::writePacket (C++ function), 229

dv::io::NetworkWriter::connectTCP (C++ function), 449

dv::io::NetworkWriter::connectUNIX (C++ function), 449

dv::io::NetworkWriter::ErrorMessageCallback (C++ type), 447

dv::io::NetworkWriter::generateHeaderContent (C++ function), 449

dv::io::NetworkWriter::getCameraName (C++ function), 449

dv::io::NetworkWriter::getClientCount (C++ function), 449

dv::io::NetworkWriter::getQueuedPacketCount (C++ function), 449

dv::io::NetworkWriter::ioThread (C++ function), 449

dv::io::NetworkWriter::mAcceptorTcp (C++ member), 450

dv::io::NetworkWriter::mAcceptorUnix (C++ member), 450

dv::io::NetworkWriter::mAedat4Writer (C++ member), 450

dv::io::NetworkWriter::mCameraName (C++ member), 450

dv::io::NetworkWriter::mClients (C++ member), 450

dv::io::NetworkWriter::mClientsMutex (C++ member), 450

dv::io::NetworkWriter::mErrorMessageHandler (C++ member), 450

dv::io::NetworkWriter::mInfoNode (C++ member), 450
 dv::io::NetworkWriter::mIoService (C++ member), 450
 dv::io::NetworkWriter::mIOThread (C++ member), 450
 dv::io::NetworkWriter::mMaxConnections (C++ member), 450
 dv::io::NetworkWriter::mQueuedPackets (C++ member), 450
 dv::io::NetworkWriter::mShutdownRequested (C++ member), 450
 dv::io::NetworkWriter::mSocketPath (C++ member), 450
 dv::io::NetworkWriter::mStreamId (C++ member), 450
 dv::io::NetworkWriter::mTLSContext (C++ member), 450
 dv::io::NetworkWriter::mTLSEnabled (C++ member), 450
 dv::io::NetworkWriter::mWriteQueue (C++ member), 450
 dv::io::NetworkWriter::NetworkWriter (C++ function), 447, 448
 dv::io::NetworkWriter::removeClient (C++ function), 449
 dv::io::NetworkWriter::writeEvents (C++ function), 448
 dv::io::NetworkWriter::writeFrame (C++ function), 448
 dv::io::NetworkWriter::writeImu (C++ function), 448
 dv::io::NetworkWriter::writePacket (C++ function), 449
 dv::io::NetworkWriter::writePacketToClients (C++ function), 449
 dv::io::NetworkWriter::WriteQueue (C++ type), 449
 dv::io::NetworkWriter::writeTriggers (C++ function), 449
 dv::io::operator& (C++ function), 616
 dv::io::operator| (C++ function), 616
 dv::io::operator|= (C++ function), 616
 dv::io::OutputBase (C++ class), 455
 dv::io::OutputBase::~~OutputBase (C++ function), 455
 dv::io::OutputBase::getCameraName (C++ function), 455
 dv::io::OutputBase::writeEvents (C++ function), 455
 dv::io::OutputBase::writeFrame (C++ function), 455
 dv::io::OutputBase::writeImu (C++ function), 455
 dv::io::OutputBase::writeTriggers (C++ function), 455
 dv::io::Reader (C++ class), 492
 dv::io::Reader::~~Reader (C++ function), 492
 dv::io::Reader::buildFileDataTable (C++ function), 494
 dv::io::Reader::decodeFileDataTable (C++ function), 494
 dv::io::Reader::decodeHeader (C++ function), 494
 dv::io::Reader::decodePacketBody (C++ function), 494
 dv::io::Reader::decompressData (C++ function), 494
 dv::io::Reader::getCompressionType (C++ function), 494
 dv::io::Reader::getStreams (C++ function), 494
 dv::io::Reader::mDecompressBuffer (C++ member), 494
 dv::io::Reader::mDecompressionSupport (C++ member), 494
 dv::io::Reader::mReadBuffer (C++ member), 494
 dv::io::Reader::mStats (C++ member), 494
 dv::io::Reader::mStreams (C++ member), 494
 dv::io::Reader::mTypeResolver (C++ member), 494
 dv::io::Reader::operator= (C++ function), 492
 dv::io::Reader::Reader (C++ function), 492
 dv::io::Reader::readFileDataTable (C++ function), 492
 dv::io::Reader::readFromInput (C++ function), 494
 dv::io::Reader::ReadHandler (C++ type), 492
 dv::io::Reader::readHeader (C++ function), 492
 dv::io::Reader::readPacket (C++ function), 492
 dv::io::Reader::readPacketBody (C++ function), 493
 dv::io::Reader::readPacketHeader (C++ function), 493
 dv::io::Reader::verifyVersion (C++ function), 492
 dv::io::ReadOnlyFile (C++ class), 495
 dv::io::ReadOnlyFile::aheadOfRange (C++ function), 496
 dv::io::ReadOnlyFile::createFileInfo (C++ function), 496
 dv::io::ReadOnlyFile::getFileInfo (C++ function), 495
 dv::io::ReadOnlyFile::getStartingPointForTimeRangeSearch (C++ function), 496
 dv::io::ReadOnlyFile::inRange (C++ function), 496
 dv::io::ReadOnlyFile::loadFileDataTable (C++ function), 496

`dv::io::ReadOnlyFile::mFileInfo` (C++ member), 496

`dv::io::ReadOnlyFile::mReader` (C++ member), 496

`dv::io::ReadOnlyFile::parseHeader` (C++ function), 496

`dv::io::ReadOnlyFile::pastRange` (C++ function), 496

`dv::io::ReadOnlyFile::read` (C++ function), 495

`dv::io::ReadOnlyFile::readClbk` (C++ function), 496

`dv::io::ReadOnlyFile::ReadOnlyFile` (C++ function), 495

`dv::io::SeekFlags` (C++ enum), 616

`dv::io::SeekFlags::CURRENT` (C++ enumerator), 616

`dv::io::SeekFlags::END` (C++ enumerator), 616

`dv::io::SeekFlags::START` (C++ enumerator), 616

`dv::io::SimpleFile` (C++ class), 504

`dv::io::SimpleFile::~SimpleFile` (C++ function), 504

`dv::io::SimpleFile::close` (C++ function), 505

`dv::io::SimpleFile::f` (C++ member), 505

`dv::io::SimpleFile::fBuffer` (C++ member), 505

`dv::io::SimpleFile::fileSize` (C++ function), 505

`dv::io::SimpleFile::flush` (C++ function), 504

`dv::io::SimpleFile::format` (C++ function), 505

`dv::io::SimpleFile::fPath` (C++ member), 505

`dv::io::SimpleFile::isOpen` (C++ function), 504

`dv::io::SimpleFile::operator=` (C++ function), 504

`dv::io::SimpleFile::path` (C++ function), 505

`dv::io::SimpleFile::read` (C++ function), 505

`dv::io::SimpleFile::readAll` (C++ function), 505

`dv::io::SimpleFile::readInto` (C++ function), 505

`dv::io::SimpleFile::rewind` (C++ function), 505

`dv::io::SimpleFile::seek` (C++ function), 505

`dv::io::SimpleFile::SimpleFile` (C++ function), 504

`dv::io::SimpleFile::tell` (C++ function), 505

`dv::io::SimpleFile::write` (C++ function), 504, 505

`dv::io::SimpleReadOnlyFile` (C++ class), 505

`dv::io::SimpleReadOnlyFile::fileSize` (C++ function), 505

`dv::io::SimpleReadOnlyFile::isOpen` (C++ function), 505

`dv::io::SimpleReadOnlyFile::path` (C++ function), 505

`dv::io::SimpleReadOnlyFile::read` (C++ function), 506

`dv::io::SimpleReadOnlyFile::readAll` (C++ function), 506

`dv::io::SimpleReadOnlyFile::readInto` (C++ function), 506

`dv::io::SimpleReadOnlyFile::rewind` (C++ function), 506

`dv::io::SimpleReadOnlyFile::seek` (C++ function), 506

`dv::io::SimpleReadOnlyFile::SimpleReadOnlyFile` (C++ function), 505

`dv::io::SimpleReadOnlyFile::tell` (C++ function), 506

`dv::io::SimpleWriteOnlyFile` (C++ class), 506

`dv::io::SimpleWriteOnlyFile::fileSize` (C++ function), 506

`dv::io::SimpleWriteOnlyFile::flush` (C++ function), 506

`dv::io::SimpleWriteOnlyFile::format` (C++ function), 506

`dv::io::SimpleWriteOnlyFile::isOpen` (C++ function), 506

`dv::io::SimpleWriteOnlyFile::path` (C++ function), 506

`dv::io::SimpleWriteOnlyFile::rewind` (C++ function), 506

`dv::io::SimpleWriteOnlyFile::seek` (C++ function), 506

`dv::io::SimpleWriteOnlyFile::SimpleWriteOnlyFile` (C++ function), 506

`dv::io::SimpleWriteOnlyFile::tell` (C++ function), 506

`dv::io::SimpleWriteOnlyFile::write` (C++ function), 506

`dv::io::StereoCameraRecording` (C++ class), 517

`dv::io::StereoCameraRecording::getLeftReader` (C++ function), 518

`dv::io::StereoCameraRecording::getRightReader` (C++ function), 518

`dv::io::StereoCameraRecording::mLeftCamera` (C++ member), 518

`dv::io::StereoCameraRecording::mReader` (C++ member), 518

`dv::io::StereoCameraRecording::mRightCamera` (C++ member), 518

`dv::io::StereoCameraRecording::StereoCameraRecording` (C++ function), 518

`dv::io::StereoCameraWriter` (C++ class), 518

`dv::io::StereoCameraWriter::configureCameraOutput` (C++ function), 520

`dv::io::StereoCameraWriter::configureStreamIds` (C++ function), 519

`dv::io::StereoCameraWriter::createStereoHeader` (C++ function), 519

`dv::io::StereoCameraWriter::file` (C++ member), 506

ber), 519
 dv::io::StereoCameraWriter::left (C++ member), 519
 dv::io::StereoCameraWriter::leftIds (C++ member), 519
 dv::io::StereoCameraWriter::leftUpdated-Config (C++ member), 519
 dv::io::StereoCameraWriter::mLeftOutputStreamDescriptors (C++ member), 519
 dv::io::StereoCameraWriter::mRightOutputStreamDescriptors (C++ member), 519
 dv::io::StereoCameraWriter::right (C++ member), 519
 dv::io::StereoCameraWriter::rightIds (C++ member), 519
 dv::io::StereoCameraWriter::rightUpdated-Config (C++ member), 519
 dv::io::StereoCameraWriter::StereoCameraWriter (C++ function), 518
 dv::io::StereoCameraWriter::StreamIdContainer (C++ struct), 530
 dv::io::StereoCameraWriter::StreamIdContainer::mEventStreamId (C++ member), 530
 dv::io::StereoCameraWriter::StreamIdContainer::mFrameStreamId (C++ member), 530
 dv::io::StereoCameraWriter::StreamIdContainer::mImuStreamId (C++ member), 530
 dv::io::StereoCameraWriter::StreamIdContainer::mTriggerStreamId (C++ member), 530
 dv::io::Stream (C++ struct), 525
 dv::io::Stream::addMetadata (C++ function), 525
 dv::io::Stream::EventStream (C++ function), 528
 dv::io::Stream::FrameStream (C++ function), 528
 dv::io::Stream::getAttribute (C++ function), 527
 dv::io::Stream::getAttributeValue (C++ function), 527
 dv::io::Stream::getCompression (C++ function), 526
 dv::io::Stream::getMetadataValue (C++ function), 525
 dv::io::Stream::getModuleName (C++ function), 526
 dv::io::Stream::getOutputName (C++ function), 526
 dv::io::Stream::getResolution (C++ function), 527
 dv::io::Stream::getSource (C++ function), 527
 dv::io::Stream::getTypeDescription (C++ function), 526
 dv::io::Stream::IMUStream (C++ function), 528
 dv::io::Stream::mId (C++ member), 527
 dv::io::Stream::mName (C++ member), 527
 dv::io::Stream::mType (C++ member), 528
 dv::io::Stream::mTypeIdIdentifier (C++ member), 528
 dv::io::Stream::mXMLNode (C++ member), 528
 dv::io::Stream::setAttribute (C++ function), 526
 dv::io::Stream::setCompression (C++ function), 526
 dv::io::Stream::setModuleName (C++ function), 526
 dv::io::Stream::setOutputName (C++ function), 526
 dv::io::Stream::setResolution (C++ function), 527
 dv::io::Stream::setSource (C++ function), 527
 dv::io::Stream::setTypeDescription (C++ function), 526
 dv::io::Stream::Stream (C++ function), 525
 dv::io::Stream::TriggerStream (C++ function), 529
 dv::io::Stream::TypedStream (C++ function), 529
 dv::io::support (C++ type), 626
 dv::io::support::AEDAT4_FILE_EXTENSION (C++ member), 626
 dv::io::support::AEDAT4_HEADER_VERSION (C++ member), 626
 dv::io::support::defaultTypeResolver (C++ function), 626
 dv::io::support::IODataBuffer (C++ class), 375
 dv::io::support::IODataBuffer::getBuffer (C++ function), 375
 dv::io::support::IODataBuffer::getBuilder (C++ function), 375
 dv::io::support::IODataBuffer::getData (C++ function), 375
 dv::io::support::IODataBuffer::getDataSize (C++ function), 375
 dv::io::support::IODataBuffer::getHeader (C++ function), 375
 dv::io::support::IODataBuffer::INITIAL_SIZE (C++ member), 375
 dv::io::support::IODataBuffer::IODataBuffer (C++ function), 375
 dv::io::support::IODataBuffer::mBuffer (C++ member), 375
 dv::io::support::IODataBuffer::mBuilder (C++ member), 375
 dv::io::support::IODataBuffer::mHeader (C++ member), 375
 dv::io::support::IODataBuffer::mIsFlat-

Buffer (C++ member), 375

dv::io::support::IODataBuffer::switchTo-
Buffer (C++ function), 375

dv::io::support::IOStatistics (C++ class), 377

dv::io::support::IOStatistics::~~IOStatist-
tics (C++ function), 377

dv::io::support::IOStatistics::addBytes
(C++ function), 377

dv::io::support::IOStatistics::IOStatist-
tics (C++ function), 377

dv::io::support::IOStatistics::mDataSize
(C++ member), 378

dv::io::support::IOStatistics::mPacketsE-
lements (C++ member), 378

dv::io::support::IOStatistics::mPacket-
sNumber (C++ member), 378

dv::io::support::IOStatistics::mPackets-
Size (C++ member), 378

dv::io::support::IOStatistics::operator=
(C++ function), 377

dv::io::support::IOStatistics::publish (C++
function), 377

dv::io::support::IOStatistics::update (C++
function), 377

dv::io::support::packetToObject (C++ func-
tion), 626

dv::io::support::Sizes (C++ struct), 506

dv::io::support::Sizes::mDataSize (C++ mem-
ber), 507

dv::io::support::Sizes::mPacketElements
(C++ member), 507

dv::io::support::Sizes::mPacketSize (C++
member), 507

dv::io::support::ThreadExtra (C++ class), 536

dv::io::support::ThreadExtra::getName (C++
function), 537

dv::io::support::ThreadExtra::setName (C++
function), 537

dv::io::support::ThreadExtra::setPriority-
Highest (C++ function), 537

dv::io::support::ThreadNameSwitch (C++ class),
537

dv::io::support::Thread-
NameSwitch::~~ThreadNameSwitch (C++
function), 537

dv::io::support::ThreadNameSwitch::origi-
nalName (C++ member), 537

dv::io::support::ThreadNameSwitch::Thread-
NameSwitch (C++ function), 537

dv::io::support::TypeResolver (C++ type), 626

dv::io::support::VariantValueOwning (C++
type), 626

dv::io::support::XMLConfigReader (C++ class),
570

dv::io::support::XMLConfigReader::con-
sumeXML (C++ function), 571

dv::io::support::XMLConfigReader::getRoot
(C++ function), 570

dv::io::support::XMLConfigReader::mRoot
(C++ member), 570

dv::io::support::XMLConfigReader::parseXML
(C++ function), 570

dv::io::support::XMLConfigReader::string-
ToValueConverter (C++ function), 571

dv::io::support::XMLConfigReader::XMLCon-
figReader (C++ function), 570

dv::io::support::XMLConfigReader::xmlFil-
terChildNodes (C++ function), 570

dv::io::support::XMLConfigWriter (C++ class),
571

dv::io::support::XMLConfigWriter::genera-
teXML (C++ function), 571

dv::io::support::XMLConfigWriter::getXML-
Content (C++ function), 571

dv::io::support::XMLConfigWriter::mXMLOut-
putContent (C++ member), 571

dv::io::support::XMLConfigWriter::value-
ToStringConverter (C++ function), 571

dv::io::support::XMLConfigWriter::writeXML
(C++ function), 571

dv::io::support::XMLConfigWriter::XMLCon-
figWriter (C++ function), 571

dv::io::support::XMLTreeAttribute (C++
struct), 571

dv::io::support::XMLTreeAttribute::mValue
(C++ member), 572

dv::io::support::XMLTreeAttribute::XML-
TreeAttribute (C++ function), 571

dv::io::support::XMLTreeCommon (C++ struct),
572

dv::io::support::XMLTreeCommon::mName (C++
member), 572

dv::io::support::XMLTreeCommon::opera-
tor<=> (C++ function), 572

dv::io::support::XMLTreeCommon::operator==
(C++ function), 572

dv::io::support::XMLTreeCommon::XML-
TreeCommon (C++ function), 572

dv::io::support::XMLTreeNode (C++ struct), 572

dv::io::support::XMLTreeNode::mAttributes
(C++ member), 572

dv::io::support::XMLTreeNode::mChildren
(C++ member), 572

dv::io::support::XMLTreeNode::XMLTreeNode
(C++ function), 572

dv::io::WriteFlags (C++ enum), 616

dv::io::WriteFlags::APPEND (C++ enumerator),
616

dv::io::WriteFlags::NONE (C++ enumerator), 616
 dv::io::WriteFlags::TRUNCATE (C++ enumerator), 616
 dv::io::WriteOnlyFile (C++ class), 566
 dv::io::WriteOnlyFile::~~WriteOnlyFile (C++ function), 567
 dv::io::WriteOnlyFile::emptyWriteBuffer (C++ function), 567
 dv::io::WriteOnlyFile::mMutex (C++ member), 567
 dv::io::WriteOnlyFile::mOutputInfo (C++ member), 567
 dv::io::WriteOnlyFile::mStopRequested (C++ member), 567
 dv::io::WriteOnlyFile::mWriteBuffer (C++ member), 567
 dv::io::WriteOnlyFile::mWriter (C++ member), 567
 dv::io::WriteOnlyFile::mWriteThread (C++ member), 567
 dv::io::WriteOnlyFile::pushFileDataTable (C++ function), 567
 dv::io::WriteOnlyFile::pushHeader (C++ function), 567
 dv::io::WriteOnlyFile::pushPacket (C++ function), 567
 dv::io::WriteOnlyFile::pushVersion (C++ function), 567
 dv::io::WriteOnlyFile::stop (C++ function), 567
 dv::io::WriteOnlyFile::write (C++ function), 567
 dv::io::WriteOnlyFile::writeFileDataTable (C++ function), 567
 dv::io::WriteOnlyFile::writeHeader (C++ function), 567
 dv::io::WriteOnlyFile::WriteOnlyFile (C++ function), 567
 dv::io::WriteOnlyFile::writePacket (C++ function), 567
 dv::io::WriteOnlyFile::writeThread (C++ function), 567
 dv::io::WriteOnlyFile::writeVersion (C++ function), 567
 dv::io::Writer (C++ class), 568
 dv::io::Writer::~~Writer (C++ function), 569
 dv::io::Writer::compressData (C++ function), 570
 dv::io::Writer::encodeAadat4Version (C++ function), 569
 dv::io::Writer::encodeFileDataTable (C++ function), 569
 dv::io::Writer::encodeFileHeader (C++ function), 569
 dv::io::Writer::encodePacketBody (C++ function), 569
 dv::io::Writer::encodePacketHeader (C++ function), 569
 dv::io::Writer::getCompressionType (C++ function), 569
 dv::io::Writer::mByteOffset (C++ member), 570
 dv::io::Writer::mCompressionSupport (C++ member), 570
 dv::io::Writer::mFileDataTable (C++ member), 570
 dv::io::Writer::mStats (C++ member), 570
 dv::io::Writer::operator= (C++ function), 569
 dv::io::Writer::updateFileDataTable (C++ function), 570
 dv::io::Writer::writeAadatVersion (C++ function), 569
 dv::io::Writer::writeFileDataTable (C++ function), 569
 dv::io::Writer::WriteHandler (C++ type), 569
 dv::io::Writer::writeHeader (C++ function), 569
 dv::io::Writer::writePacket (C++ function), 569
 dv::io::Writer::Writer (C++ function), 569
 dv::io::Writer::writeToDestination (C++ function), 570
 dv::IOHeader (C++ struct), 375
 dv::IOHeader::compression (C++ member), 376
 dv::IOHeader::dataTablePosition (C++ member), 376
 dv::IOHeader::GetFullyQualifiedName (C++ function), 376
 dv::IOHeader::infoNode (C++ member), 376
 dv::IOHeader::IOHeader (C++ function), 376
 dv::IOHeader::TableType (C++ type), 376
 dv::IOHeaderBufferHasIdentifier (C++ function), 599
 dv::IOHeaderBuilder (C++ struct), 376
 dv::IOHeaderBuilder::add_compression (C++ function), 376
 dv::IOHeaderBuilder::add_dataTablePosition (C++ function), 376
 dv::IOHeaderBuilder::add_infoNode (C++ function), 376
 dv::IOHeaderBuilder::fbb_ (C++ member), 376
 dv::IOHeaderBuilder::Finish (C++ function), 376
 dv::IOHeaderBuilder::IOHeaderBuilder (C++ function), 376
 dv::IOHeaderBuilder::operator= (C++ function), 376
 dv::IOHeaderBuilder::start_ (C++ member), 376
 dv::IOHeaderFlatbuffer (C++ struct), 376
 dv::IOHeaderFlatbuffer::compression (C++ function), 377
 dv::IOHeaderFlatbuffer::dataTablePosition (C++ function), 377

dv::IOHeaderFlatbuffer::GetFullyQualified-
 Name (C++ function), 377
 dv::IOHeaderFlatbuffer::identifier (C++
 member), 377
 dv::IOHeaderFlatbuffer::infoNode (C++ func-
 tion), 377
 dv::IOHeaderFlatbuffer::MiniReflectType-
 Table (C++ function), 377
 dv::IOHeaderFlatbuffer::NativeTableType
 (C++ type), 376
 dv::IOHeaderFlatbuffer::Pack (C++ function),
 377
 dv::IOHeaderFlatbuffer::UnPack (C++ function),
 377
 dv::IOHeaderFlatbuffer::UnPackTo (C++ func-
 tion), 377
 dv::IOHeaderFlatbuffer::UnPackToFrom (C++
 function), 377
 dv::IOHeaderFlatbuffer::Verify (C++ function),
 377
 dv::IOHeaderIdentifier (C++ function), 599
 dv::IOHeaderTypeTable (C++ function), 599
 dv::isWithinDimensions (C++ function), 585
 dv::kinematics (C++ type), 626
 dv::kinematics::LinearTransformer (C++ class),
 390
 dv::kinematics::LinearTransformer::begin
 (C++ function), 391
 dv::kinematics::LinearTransformer::buffer-
 LowerBound (C++ function), 393
 dv::kinematics::LinearTransformer::buffer-
 UpperBound (C++ function), 393
 dv::kinematics::LinearTransformer::cbegin
 (C++ function), 391
 dv::kinematics::LinearTransformer::cend
 (C++ function), 391
 dv::kinematics::LinearTransformer::clear
 (C++ function), 391
 dv::kinematics::LinearTrans-
 former::const_iterator (C++ type),
 391
 dv::kinematics::LinearTransformer::ear-
 liestTransformation (C++ function),
 392
 dv::kinematics::LinearTransformer::empty
 (C++ function), 391
 dv::kinematics::LinearTransformer::end (C++
 function), 391
 dv::kinematics::LinearTransformer::get-
 TransformAt (C++ function), 391
 dv::kinematics::LinearTransformer::get-
 TransformsBetween (C++ function), 392
 dv::kinematics::LinearTransformer::inter-
 polateComponentwise (C++ function),
 393
 dv::kinematics::LinearTransformer::isWith-
 inTimeRange (C++ function), 392
 dv::kinematics::LinearTransformer::itera-
 tor (C++ type), 391
 dv::kinematics::LinearTransformer::latest-
 Transformation (C++ function), 392
 dv::kinematics::LinearTransformer::Lin-
 earTransformer (C++ function), 391
 dv::kinematics::LinearTransformer::mTrans-
 forms (C++ member), 393
 dv::kinematics::LinearTransformer::push-
 Transformation (C++ function), 391
 dv::kinematics::LinearTransformer::resam-
 pleTransforms (C++ function), 392
 dv::kinematics::LinearTransformer::setCa-
 pacity (C++ function), 392
 dv::kinematics::LinearTransformer::size
 (C++ function), 392
 dv::kinematics::LinearTransformer::Trans-
 formationBuffer (C++ type), 393
 dv::kinematics::LinearTransformer::Trans-
 formationType (C++ type), 393
 dv::kinematics::LinearTransformerd (C++
 type), 626
 dv::kinematics::LinearTransformerf (C++
 type), 626
 dv::kinematics::MotionCompensator (C++ class),
 433
 dv::kinematics::MotionCompensator::accept
 (C++ function), 433
 dv::kinematics::MotionCompensator::accumu-
 lator (C++ member), 436
 dv::kinematics::MotionCompensator::compen-
 sateEvents (C++ function), 435
 dv::kinematics::MotionCompensator::con-
 stantDepth (C++ member), 436
 dv::kinematics::MotionCompensator::depths
 (C++ member), 436
 dv::kinematics::MotionCompensator::event-
 Buffer (C++ member), 436
 dv::kinematics::MotionCompensator::genera-
 teEvents (C++ function), 433
 dv::kinematics::MotionCompensator::genera-
 teEventsAt (C++ function), 435
 dv::kinematics::MotionCompensator::gener-
 ateFrame (C++ function), 433
 dv::kinematics::MotionCompensator::gener-
 ateFrameAt (C++ function), 435
 dv::kinematics::MotionCompensator::gener-
 ateTransforms (C++ function), 435
 dv::kinematics::MotionCompensator::getCon-
 stantDepth (C++ function), 434
 dv::kinematics::MotionCompensator::getInfo

- (C++ function), 433
- dv::kinematics::MotionCompensator::info (C++ member), 436
- dv::kinematics::MotionCompensator::Info (C++ struct), 372
- dv::kinematics::MotionCompensator::Info::accumulatedEventCount (C++ member), 372
- dv::kinematics::MotionCompensator::Info::depthAvailable (C++ member), 372
- dv::kinematics::MotionCompensator::Info::depthTime (C++ member), 372
- dv::kinematics::MotionCompensator::Info::generationTime (C++ member), 372
- dv::kinematics::MotionCompensator::Info::imageCompensated (C++ member), 372
- dv::kinematics::MotionCompensator::Info::inputEventCount (C++ member), 372
- dv::kinematics::MotionCompensator::Info::transformsAvailable (C++ member), 372
- dv::kinematics::MotionCompensator::MotionCompensator (C++ function), 434
- dv::kinematics::MotionCompensator::operator<< (C++ function), 434
- dv::kinematics::MotionCompensator::operator>> (C++ function), 434, 435
- dv::kinematics::MotionCompensator::predictor (C++ member), 436
- dv::kinematics::MotionCompensator::reset (C++ function), 434
- dv::kinematics::MotionCompensator::samplingPeriod (C++ member), 436
- dv::kinematics::MotionCompensator::setConstantDepth (C++ function), 435
- dv::kinematics::MotionCompensator::storageDuration (C++ member), 436
- dv::kinematics::MotionCompensator::transformer (C++ member), 436
- dv::kinematics::PixelMotionPredictor (C++ class), 476
- dv::kinematics::PixelMotionPredictor::~PixelMotionPredictor (C++ function), 477
- dv::kinematics::PixelMotionPredictor::camera (C++ member), 478
- dv::kinematics::PixelMotionPredictor::isUseDistortion (C++ function), 477
- dv::kinematics::PixelMotionPredictor::PixelMotionPredictor (C++ function), 477
- dv::kinematics::PixelMotionPredictor::predict (C++ function), 477
- dv::kinematics::PixelMotionPredictor::predictEvents (C++ function), 477
- dv::kinematics::PixelMotionPredictor::predictSequence (C++ function), 477
- dv::kinematics::PixelMotionPredictor::setUseDistortion (C++ function), 477
- dv::kinematics::PixelMotionPredictor::SharedPtr (C++ type), 476
- dv::kinematics::PixelMotionPredictor::UniquePtr (C++ type), 476
- dv::kinematics::PixelMotionPredictor::useDistortion (C++ member), 478
- dv::kinematics::Transformation (C++ class), 548
- dv::kinematics::Transformation::delta (C++ function), 550
- dv::kinematics::Transformation::fromNonHomogenous (C++ function), 550
- dv::kinematics::Transformation::getQuaternion (C++ function), 549
- dv::kinematics::Transformation::getRotationMatrix (C++ function), 549
- dv::kinematics::Transformation::getTimeStamp (C++ function), 549
- dv::kinematics::Transformation::getTransform (C++ function), 549
- dv::kinematics::Transformation::getTranslation (C++ function), 549
- dv::kinematics::Transformation::inverse (C++ function), 550
- dv::kinematics::Transformation::mT (C++ member), 550
- dv::kinematics::Transformation::mTimestamp (C++ member), 550
- dv::kinematics::Transformation::operator== (C++ function), 550
- dv::kinematics::Transformation::rotatePoint (C++ function), 549
- dv::kinematics::Transformation::Transformation (C++ function), 548, 549
- dv::kinematics::Transformation::transformPoint (C++ function), 549
- dv::kinematics::Transformationd (C++ type), 626
- dv::kinematics::Transformationf (C++ type), 626
- dv::Landmark (C++ struct), 386
- dv::Landmark::covariance (C++ member), 387
- dv::Landmark::descriptor (C++ member), 387
- dv::Landmark::descriptorType (C++ member), 387

`dv::Landmark::GetFullyQualifiedName` (C++ *function*), 387

`dv::Landmark::id` (C++ *member*), 387

`dv::Landmark::Landmark` (C++ *function*), 387

`dv::Landmark::observations` (C++ *member*), 387

`dv::Landmark::pt` (C++ *member*), 387

`dv::Landmark::TableType` (C++ *type*), 387

`dv::Landmark::timestamp` (C++ *member*), 387

`dv::LandmarkBuilder` (C++ *struct*), 387

`dv::LandmarkBuilder::add_covariance` (C++ *function*), 387

`dv::LandmarkBuilder::add_descriptor` (C++ *function*), 387

`dv::LandmarkBuilder::add_descriptorType` (C++ *function*), 387

`dv::LandmarkBuilder::add_id` (C++ *function*), 387

`dv::LandmarkBuilder::add_observations` (C++ *function*), 387

`dv::LandmarkBuilder::add_pt` (C++ *function*), 387

`dv::LandmarkBuilder::add_timestamp` (C++ *function*), 387

`dv::LandmarkBuilder::fbb_` (C++ *member*), 388

`dv::LandmarkBuilder::Finish` (C++ *function*), 388

`dv::LandmarkBuilder::LandmarkBuilder` (C++ *function*), 388

`dv::LandmarkBuilder::operator=` (C++ *function*), 388

`dv::LandmarkBuilder::start_` (C++ *member*), 388

`dv::LandmarkFlatbuffer` (C++ *struct*), 388

`dv::LandmarkFlatbuffer::covariance` (C++ *function*), 388

`dv::LandmarkFlatbuffer::descriptor` (C++ *function*), 388

`dv::LandmarkFlatbuffer::descriptorType` (C++ *function*), 388

`dv::LandmarkFlatbuffer::GetFullyQualifiedName` (C++ *function*), 389

`dv::LandmarkFlatbuffer::id` (C++ *function*), 388

`dv::LandmarkFlatbuffer::MiniReflectTypeTable` (C++ *function*), 389

`dv::LandmarkFlatbuffer::NativeTableType` (C++ *type*), 388

`dv::LandmarkFlatbuffer::observations` (C++ *function*), 388

`dv::LandmarkFlatbuffer::Pack` (C++ *function*), 389

`dv::LandmarkFlatbuffer::pt` (C++ *function*), 388

`dv::LandmarkFlatbuffer::timestamp` (C++ *function*), 388

`dv::LandmarkFlatbuffer::UnPack` (C++ *function*), 388

`dv::LandmarkFlatbuffer::UnPackTo` (C++ *function*), 388

`dv::LandmarkFlatbuffer::UnPackToFrom` (C++ *function*), 389

`dv::LandmarkFlatbuffer::Verify` (C++ *function*), 389

`dv::LandmarkFlatbuffer::Verify` (C++ *function*), 388

`dv::LandmarksPacket` (C++ *struct*), 389

`dv::LandmarksPacket::elements` (C++ *member*), 389

`dv::LandmarksPacket::GetFullyQualifiedName` (C++ *function*), 389

`dv::LandmarksPacket::LandmarksPacket` (C++ *function*), 389

`dv::LandmarksPacket::operator<<` (C++ *function*), 389

`dv::LandmarksPacket::referenceFrame` (C++ *member*), 389

`dv::LandmarksPacket::TableType` (C++ *type*), 389

`dv::LandmarksPacketBufferHasIdentifier` (C++ *function*), 594

`dv::LandmarksPacketBuilder` (C++ *struct*), 389

`dv::LandmarksPacketBuilder::add_elements` (C++ *function*), 389

`dv::LandmarksPacketBuilder::add_referenceFrame` (C++ *function*), 389

`dv::LandmarksPacketBuilder::fbb_` (C++ *member*), 390

`dv::LandmarksPacketBuilder::Finish` (C++ *function*), 389

`dv::LandmarksPacketBuilder::LandmarksPacketBuilder` (C++ *function*), 389

`dv::LandmarksPacketBuilder::operator=` (C++ *function*), 389

`dv::LandmarksPacketBuilder::start_` (C++ *member*), 390

`dv::LandmarksPacketFlatbuffer` (C++ *struct*), 390

`dv::LandmarksPacketFlatbuffer::elements` (C++ *function*), 390

`dv::LandmarksPacketFlatbuffer::GetFullyQualifiedName` (C++ *function*), 390

`dv::LandmarksPacketFlatbuffer::identifier` (C++ *member*), 390

`dv::LandmarksPacketFlatbuffer::MiniReflectTypeTable` (C++ *function*), 390

`dv::LandmarksPacketFlatbuffer::NativeTableType` (C++ *type*), 390

`dv::LandmarksPacketFlatbuffer::Pack` (C++ *function*), 390

`dv::LandmarksPacketFlatbuffer::referenceFrame` (C++ *function*), 390

`dv::LandmarksPacketFlatbuffer::UnPack` (C++ *function*), 390

`dv::LandmarksPacketFlatbuffer::UnPackTo` (C++ *function*), 390

`dv::LandmarksPacketFlatbuffer::UnPackToFrom` (C++ *function*), 390

`dv::LandmarksPacketFlatbuffer::Verify` (C++ *function*), 390

function), 390
 dv::LandmarksPacketIdentifier (C++ *function*), 594
 dv::LandmarksPacketTypeTable (C++ *function*), 593
 dv::LandmarkTypeTable (C++ *function*), 593
 dv::mallocConstructor (C++ *function*), 585
 dv::mallocConstructorSize (C++ *function*), 585
 dv::mallocDestructor (C++ *function*), 585
 dv::maskFilter (C++ *function*), 583
 dv::measurements (C++ *type*), 626
 dv::measurements::Depth (C++ *struct*), 271
 dv::measurements::Depth::Depth (C++ *function*), 271
 dv::measurements::Depth::mDepth (C++ *member*), 271
 dv::measurements::Depth::mTimestamp (C++ *member*), 271
 dv::MultiStreamSlicer (C++ *class*), 436
 dv::MultiStreamSlicer::accept (C++ *function*), 437
 dv::MultiStreamSlicer::addStream (C++ *function*), 437
 dv::MultiStreamSlicer::doEveryNumberOfElements (C++ *function*), 438
 dv::MultiStreamSlicer::doEveryTimeInterval (C++ *function*), 437, 438
 dv::MultiStreamSlicer::eraseUpTo (C++ *function*), 442
 dv::MultiStreamSlicer::eraseUpToIterable (C++ *function*), 442
 dv::MultiStreamSlicer::evaluate (C++ *function*), 441
 dv::MultiStreamSlicer::getMinEvaluatedJobTime (C++ *function*), 440
 dv::MultiStreamSlicer::getMinLastBufferTimestamps (C++ *function*), 440
 dv::MultiStreamSlicer::getPacketTimeWindow (C++ *function*), 442
 dv::MultiStreamSlicer::hasJob (C++ *function*), 439
 dv::MultiStreamSlicer::InputType (C++ *type*), 437
 dv::MultiStreamSlicer::isPacketEmpty (C++ *function*), 442
 dv::MultiStreamSlicer::MainType (C++ *type*), 440
 dv::MultiStreamSlicer::MapOfVariants (C++ *class*), 396
 dv::MultiStreamSlicer::MapOfVariants::get (C++ *function*), 396
 dv::MultiStreamSlicer::mBuffer (C++ *member*), 440
 dv::MultiStreamSlicer::mergePackets (C++ *function*), 441
 dv::MultiStreamSlicer::mHashCounter (C++ *member*), 440
 dv::MultiStreamSlicer::mLastReceivedBufferTimestamps (C++ *member*), 440
 dv::MultiStreamSlicer::mMainBufferSeekTime (C++ *member*), 440
 dv::MultiStreamSlicer::mMainSlicer (C++ *member*), 440
 dv::MultiStreamSlicer::mMainStreamName (C++ *member*), 440
 dv::MultiStreamSlicer::mMapFromSliceJobsToMainSlicerIds (C++ *member*), 440
 dv::MultiStreamSlicer::modifyNumberInterval (C++ *function*), 439
 dv::MultiStreamSlicer::modifyTimeInterval (C++ *function*), 439
 dv::MultiStreamSlicer::mSliceJobs (C++ *member*), 440
 dv::MultiStreamSlicer::MultiStreamSlicer (C++ *function*), 437
 dv::MultiStreamSlicer::removeJob (C++ *function*), 439
 dv::MultiStreamSlicer::setStreamSeekTime (C++ *function*), 439
 dv::MultiStreamSlicer::SliceJob (C++ *class*), 507
 dv::MultiStreamSlicer::SliceJob::JobCallback (C++ *type*), 507
 dv::MultiStreamSlicer::SliceJob::mCallback (C++ *member*), 507
 dv::MultiStreamSlicer::SliceJob::mInterval (C++ *member*), 508
 dv::MultiStreamSlicer::SliceJob::mLastEvaluatedTimestamp (C++ *member*), 508
 dv::MultiStreamSlicer::SliceJob::mNumberOfElements (C++ *member*), 508
 dv::MultiStreamSlicer::SliceJob::mTimeSlicing (C++ *member*), 508
 dv::MultiStreamSlicer::SliceJob::mType (C++ *member*), 507
 dv::MultiStreamSlicer::SliceJob::run (C++ *function*), 507
 dv::MultiStreamSlicer::SliceJob::SliceJob (C++ *function*), 507
 dv::MultiStreamSlicer::SliceJob::SliceType (C++ *enum*), 507
 dv::MultiStreamSlicer::SliceJob::SliceType::NUMBER (C++ *enumerator*), 507
 dv::MultiStreamSlicer::SliceJob::SliceType::TIME (C++ *enumerator*), 507
 dv::MultiStreamSlicer::slicePacket (C++ *function*), 441

dv::MultiStreamSlicer::slicePacketSpecific
(C++ function), 441

dv::MultiStreamSlicer::sliceVector (C++
function), 441

dv::NAME_STRING (C++ member), 600

dv::noise (C++ type), 626

dv::noise::BackgroundActivityNoiseFilter
(C++ class), 199

dv::noise::BackgroundActivityNoise-
Filter::BackgroundActivityNoise-
Filter (C++ function), 199

dv::noise::BackgroundActivityNoise-
Filter::doBackgroundActivityLookup
(C++ function), 200

dv::noise::BackgroundActivityNoise-
Filter::doBackgroundActivity-
Lookup_unsafe (C++ function), 200

dv::noise::BackgroundActivityNoise-
Filter::getBackgroundActivityDu-
ration (C++ function), 200

dv::noise::BackgroundActivityNoise-
Filter::mBackgroundActivityDuration
(C++ member), 200

dv::noise::BackgroundActivityNoise-
Filter::mResolutionLimits (C++
member), 200

dv::noise::BackgroundActivityNoise-
Filter::mTimeSurface (C++ member),
200

dv::noise::BackgroundActivityNoise-
Filter::operator<< (C++ function),
200

dv::noise::BackgroundActivityNoise-
Filter::retain (C++ function), 200

dv::noise::BackgroundActivityNoise-
Filter::setBackgroundActivityDu-
ration (C++ function), 200

dv::noise::BandCutFilter (C++ class), 201

dv::noise::BandCutFilter::BandCutFilter
(C++ function), 201

dv::noise::BandCutFilter::getLowerCutOff-
Frequency (C++ function), 201

dv::noise::BandCutFilter::getUpperCutOff-
Frequency (C++ function), 201

dv::noise::BandCutFilter::operator<< (C++
function), 201

dv::noise::BandCutFilter::setLowerCutOff-
Frequency (C++ function), 201

dv::noise::BandCutFilter::setUpperCutOff-
Frequency (C++ function), 201

dv::noise::BandPassFilter (C++ class), 201

dv::noise::BandPassFilter::BandPassFilter
(C++ function), 202

dv::noise::BandPassFilter::getLowerCutOff-
Frequency (C++ function), 202

dv::noise::BandPassFilter::getUpperCutOff-
Frequency (C++ function), 202

dv::noise::BandPassFilter::operator<< (C++
function), 202

dv::noise::BandPassFilter::setLowerCutOff-
Frequency (C++ function), 202

dv::noise::BandPassFilter::setUpperCutOff-
Frequency (C++ function), 202

dv::noise::BaseFrequencyFilter (C++ class), 202

dv::noise::BaseFrequencyFilter::BaseFre-
quencyFilter (C++ function), 203

dv::noise::BaseFrequencyFilter::mFilter-
Type (C++ member), 203

dv::noise::BaseFrequencyFilter::mHighCut-
OffPeriod (C++ member), 203

dv::noise::BaseFrequencyFilter::mLowCut-
OffPeriod (C++ member), 203

dv::noise::BaseFrequencyFilter::mTimeSur-
face (C++ member), 203

dv::noise::BaseFrequencyFilter::retain (C++
function), 203

dv::noise::FastDecayNoiseFilter (C++ class),
341

dv::noise::FastDecayNoiseFilter::FastDe-
cayNoiseFilter (C++ function), 341

dv::noise::FastDecayNoiseFilter::getH-
alfLife (C++ function), 342

dv::noise::FastDecayNoiseFilter::get-
NoiseThreshold (C++ function), 342

dv::noise::FastDecayNoiseFilter::mDecayLUT
(C++ member), 342

dv::noise::FastDecayNoise-
Filter::mHalfLifeMicros (C++ member),
343

dv::noise::FastDecayNoise-
Filter::mNoiseThreshold (C++ member),
343

dv::noise::FastDecayNoiseFilter::mSubdivi-
sionFactor (C++ member), 342

dv::noise::FastDecayNoiseFilter::mTimeSur-
face (C++ member), 342

dv::noise::FastDecayNoiseFilter::opera-
tor<< (C++ function), 342

dv::noise::FastDecayNoiseFilter::retain
(C++ function), 342

dv::noise::FastDecayNoise-
Filter::setHalfLife (C++ function),
342

dv::noise::FastDecayNoiseFilter::set-
NoiseThreshold (C++ function), 342

dv::noise::FrequencyFilterType (C++ enum),
627

dv::noise::FrequencyFilterType::CUT (C++

enumerator), 627
 dv::noise::FrequencyFilterType::PASS (C++ *enumerator*), 627
 dv::noise::HighPassFilter (C++ *class*), 360
 dv::noise::HighPassFilter::getCutOffFrequency (C++ *function*), 361
 dv::noise::HighPassFilter::HighPassFilter (C++ *function*), 361
 dv::noise::HighPassFilter::operator<< (C++ *function*), 361
 dv::noise::HighPassFilter::setCutOffFrequency (C++ *function*), 361
 dv::noise::KNoiseFilter (C++ *class*), 385
 dv::noise::KNoiseFilter::checkColumnSupport (C++ *function*), 386
 dv::noise::KNoiseFilter::checkRowSupport (C++ *function*), 386
 dv::noise::KNoiseFilter::doKNoiseLookup (C++ *function*), 386
 dv::noise::KNoiseFilter::doKNoiseLookup_unsafe (C++ *function*), 386
 dv::noise::KNoiseFilter::getTemporalWindowDuration (C++ *function*), 385
 dv::noise::KNoiseFilter::KMemCell (C++ *struct*), 384
 dv::noise::KNoiseFilter::KMemCell::mOtherAddress (C++ *member*), 385
 dv::noise::KNoiseFilter::KMemCell::mPolarity (C++ *member*), 385
 dv::noise::KNoiseFilter::KMemCell::mTimestamp (C++ *member*), 385
 dv::noise::KNoiseFilter::KNoiseFilter (C++ *function*), 385
 dv::noise::KNoiseFilter::mColumnCells (C++ *member*), 386
 dv::noise::KNoiseFilter::mResolutionLimits (C++ *member*), 386
 dv::noise::KNoiseFilter::mRowCells (C++ *member*), 386
 dv::noise::KNoiseFilter::mTimeDelta (C++ *member*), 386
 dv::noise::KNoiseFilter::operator<< (C++ *function*), 385
 dv::noise::KNoiseFilter::retain (C++ *function*), 385
 dv::noise::KNoiseFilter::setTemporalWindowDuration (C++ *function*), 385
 dv::noise::LowPassFilter (C++ *class*), 394
 dv::noise::LowPassFilter::getCutOffFrequency (C++ *function*), 394
 dv::noise::LowPassFilter::LowPassFilter (C++ *function*), 394
 dv::noise::LowPassFilter::operator<< (C++ *function*), 394
 dv::noise::LowPassFilter::setCutOffFrequency (C++ *function*), 394
 dv::noise::operator<< (C++ *function*), 627
 dv::now (C++ *function*), 585
 dv::Observation (C++ *struct*), 451
 dv::Observation::cameraId (C++ *member*), 451
 dv::Observation::cameraName (C++ *member*), 451
 dv::Observation::GetFullyQualified_name (C++ *function*), 452
 dv::Observation::Observation (C++ *function*), 451
 dv::Observation::TableType (C++ *type*), 451
 dv::Observation::timestamp (C++ *member*), 451
 dv::Observation::trackId (C++ *member*), 451
 dv::ObservationBuilder (C++ *struct*), 452
 dv::ObservationBuilder::add_cameraId (C++ *function*), 452
 dv::ObservationBuilder::add_cameraName (C++ *function*), 452
 dv::ObservationBuilder::add_timestamp (C++ *function*), 452
 dv::ObservationBuilder::add_trackId (C++ *function*), 452
 dv::ObservationBuilder::fbb_ (C++ *member*), 452
 dv::ObservationBuilder::Finish (C++ *function*), 452
 dv::ObservationBuilder::ObservationBuilder (C++ *function*), 452
 dv::ObservationBuilder::operator= (C++ *function*), 452
 dv::ObservationBuilder::start_ (C++ *member*), 452
 dv::ObservationFlatbuffer (C++ *struct*), 452
 dv::ObservationFlatbuffer::cameraId (C++ *function*), 452
 dv::ObservationFlatbuffer::cameraName (C++ *function*), 452
 dv::ObservationFlatbuffer::GetFullyQualified_name (C++ *function*), 453
 dv::ObservationFlatbuffer::MiniReflectTypeTable (C++ *function*), 453
 dv::ObservationFlatbuffer::NativeTableType (C++ *type*), 452
 dv::ObservationFlatbuffer::Pack (C++ *function*), 453
 dv::ObservationFlatbuffer::timestamp (C++ *function*), 452
 dv::ObservationFlatbuffer::trackId (C++ *function*), 452
 dv::ObservationFlatbuffer::UnPack (C++ *function*), 452
 dv::ObservationFlatbuffer::UnPackTo (C++

- function*), 452
- dv::ObservationFlatbuffer::UnPackToFrom (C++ *function*), 453
- dv::ObservationFlatbuffer::Verify (C++ *function*), 452
- dv::ObservationTypeTable (C++ *function*), 593
- dv::operator<< (C++ *function*), 584
- dv::operator== (C++ *function*), 586599
- dv::optimization (C++ *type*), 627
- dv::optimization::ContrastMaximizationWrapper (C++ *class*), 230
- dv::optimization::ContrastMaximizationWrapper::ContrastMaximizationWrapper (C++ *function*), 230
- dv::optimization::ContrastMaximizationWrapper::mFunctor (C++ *member*), 231
- dv::optimization::ContrastMaximizationWrapper::mParams (C++ *member*), 231
- dv::optimization::ContrastMaximizationWrapper::optimizationOutput (C++ *struct*), 454
- dv::optimization::ContrastMaximizationWrapper::optimizationOutput::iter (C++ *member*), 454
- dv::optimization::ContrastMaximizationWrapper::optimizationOutput::optimizationSuccessful (C++ *member*), 454
- dv::optimization::ContrastMaximizationWrapper::optimizationOutput::optimizedVariable (C++ *member*), 454
- dv::optimization::ContrastMaximizationWrapper::optimizationParameters (C++ *struct*), 454
- dv::optimization::ContrastMaximizationWrapper::optimizationParameters::epsfcn (C++ *member*), 455
- dv::optimization::ContrastMaximizationWrapper::optimizationParameters::ftol (C++ *member*), 455
- dv::optimization::ContrastMaximizationWrapper::optimizationParameters::gtol (C++ *member*), 455
- dv::optimization::ContrastMaximizationWrapper::optimizationParameters::learningRate (C++ *member*), 455
- dv::optimization::ContrastMaximizationWrapper::optimizationParameters::maxfev (C++ *member*), 455
- dv::optimization::ContrastMaximizationWrapper::optimizationParameters::xtol (C++ *member*), 455
- dv::optimization::ContrastMaximizationWrapper::optimize (C++ *function*), 230
- dv::optimization::OptimizationFunctor (C++ *class*), 453
- dv::optimization::OptimizationFunctor::inputs (C++ *function*), 454
- dv::optimization::OptimizationFunctor::InputType (C++ *type*), 453
- dv::optimization::OptimizationFunctor::JacobianType (C++ *type*), 453
- dv::optimization::OptimizationFunctor::mInputs (C++ *member*), 454
- dv::optimization::OptimizationFunctor::mValues (C++ *member*), 454
- dv::optimization::OptimizationFunctor::operator() (C++ *function*), 454
- dv::optimization::OptimizationFunctor::OptimizationFunctor (C++ *function*), 454
- dv::optimization::OptimizationFunctor::PhonyNameDueToError::InputsAtCompileTime (C++ *enumerator*), 453
- dv::optimization::OptimizationFunctor::PhonyNameDueToError::ValuesAtCompileTime (C++ *enumerator*), 453
- dv::optimization::OptimizationFunctor::Scalar (C++ *type*), 453
- dv::optimization::OptimizationFunctor::values (C++ *function*), 454
- dv::optimization::OptimizationFunctor::ValueType (C++ *type*), 453
- dv::optimization::RotationLossFunctor (C++ *class*), 499
- dv::optimization::RotationLossFunctor::mCamera (C++ *member*), 500
- dv::optimization::RotationLossFunctor::mContribution (C++ *member*), 500
- dv::optimization::RotationLossFunctor::mEvents (C++ *member*), 500
- dv::optimization::RotationLossFunctor::mImuSamples (C++ *member*), 500
- dv::optimization::RotationLossFunctor::mImuToTargetTimeOffsetUs (C++ *member*), 500
- dv::optimization::RotationLossFunctor::mT_S_target (C++ *member*), 500
- dv::optimization::RotationLossFunctor::operator() (C++ *function*), 500
- dv::optimization::RotationLossFunctor::RotationLossFunctor (C++ *function*), 499
- dv::optimization::TranslationLossFunctor (C++ *class*), 550
- dv::optimization::TranslationLossFunctor::mCamera (C++ *member*), 551
- dv::optimization::TranslationLossFunctor::optimize (C++ *function*), 230

tor::mContribution (C++ member), 551
 dv::optimization::TranslationLossFunc-
 tor::mEvents (C++ member), 551
 dv::optimization::TranslationLossFunc-
 tor::operator() (C++ function), 551
 dv::optimization::TranslationLossFunc-
 tor::TranslationLossFunc (C++
 function), 551
 dv::PacketHeaderTypeTable (C++ function), 598
 dv::packets (C++ type), 627
 dv::packets::getPacketBegin (C++ function), 628
 dv::packets::getPacketEnd (C++ function), 628
 dv::packets::getPacketSize (C++ function), 628
 dv::packets::getPacketTimestamp (C++ func-
 tion), 628
 dv::packets::getPacketTimeWindow (C++ func-
 tion), 628
 dv::packets::getTimestamp (C++ function), 627
 dv::packets::isPacketEmpty (C++ function), 627
 dv::packets::Timestamp (C++ enum), 627
 dv::packets::Timestamp::END (C++ enumerator),
 627
 dv::packets::Timestamp::START (C++ enumera-
 tor), 627
 dv::PartialEventData (C++ class), 470
 dv::PartialEventData::_unsafe_addEvent (C++
 function), 472
 dv::PartialEventData::_unsafe_moveEvent
 (C++ function), 472
 dv::PartialEventData::availableCapacity
 (C++ function), 473
 dv::PartialEventData::back (C++ function), 473
 dv::PartialEventData::begin (C++ function), 471
 dv::PartialEventData::canStoreMoreEvents
 (C++ function), 473
 dv::PartialEventData::capacity_ (C++ mem-
 ber), 474
 dv::PartialEventData::data_ (C++ member), 474
 dv::PartialEventData::end (C++ function), 471
 dv::PartialEventData::front (C++ function), 473
 dv::PartialEventData::getHighestTime (C++
 function), 473
 dv::PartialEventData::getLength (C++ func-
 tion), 473
 dv::PartialEventData::getLowestTime (C++
 function), 473
 dv::PartialEventData::highestTime_ (C++
 member), 474
 dv::PartialEventData::iterator (C++ type), 474
 dv::PartialEventData::iteratorAtTime (C++
 function), 471
 dv::PartialEventData::length_ (C++ member),
 474
 dv::PartialEventData::lowestTime_ (C++ mem-
 ber), 474
 dv::PartialEventData::merge (C++ function), 474
 dv::PartialEventData::modifiableDataPtr_
 (C++ member), 474
 dv::PartialEventData::operator[] (C++ func-
 tion), 473
 dv::PartialEventData::PartialEventData (C++
 function), 471
 dv::PartialEventData::referencesConstData_
 (C++ member), 474
 dv::PartialEventData::sliceBack (C++ func-
 tion), 471
 dv::PartialEventData::sliceFront (C++ func-
 tion), 471
 dv::PartialEventData::sliceTimeBack (C++
 function), 472
 dv::PartialEventData::sliceTimeFront (C++
 function), 472
 dv::PartialEventData::start_ (C++ member),
 474
 dv::PartialEventDataTimeComparator (C++
 class), 474
 dv::PartialEventDataTimeComparator::lower_
 (C++ member), 475
 dv::PartialEventDataTimeComparator::opera-
 tor() (C++ function), 475
 dv::PartialEventDataTimeComparator::Par-
 tialEventDataTimeComparator (C++
 function), 475
 dv::pathResolveExisting (C++ function), 585
 dv::pathResolveNonExisting (C++ function), 585
 dv::PIXEL_COLOR_KEYS (C++ member), 600
 dv::PixelArrangement (C++ enum), 578
 dv::PixelArrangement::BGRG (C++ enumerator),
 578
 dv::PixelArrangement::GBGR (C++ enumerator),
 578
 dv::PixelArrangement::GRGB (C++ enumerator),
 578
 dv::PixelArrangement::MONO (C++ enumerator),
 578
 dv::PixelArrangement::RGBG (C++ enumerator),
 578
 dv::Point2fTypeTable (C++ function), 591
 dv::Point3fTypeTable (C++ function), 591
 dv::polarityFilter (C++ function), 583
 dv::Pose (C++ struct), 478
 dv::Pose::GetFullyQualifiedName (C++ func-
 tion), 478
 dv::Pose::operator<< (C++ function), 478
 dv::Pose::Pose (C++ function), 478
 dv::Pose::referenceFrame (C++ member), 478
 dv::Pose::rotation (C++ member), 478

`dv::Pose::TableType (C++ type)`, 478
`dv::Pose::targetFrame (C++ member)`, 478
`dv::Pose::timestamp (C++ member)`, 478
`dv::Pose::translation (C++ member)`, 478
`dv::PoseBufferHasIdentifier (C++ function)`, 595
`dv::PoseBuilder (C++ struct)`, 478
`dv::PoseBuilder::add_referenceFrame (C++ function)`, 479
`dv::PoseBuilder::add_rotation (C++ function)`, 479
`dv::PoseBuilder::add_targetFrame (C++ function)`, 479
`dv::PoseBuilder::add_timestamp (C++ function)`, 479
`dv::PoseBuilder::add_translation (C++ function)`, 479
`dv::PoseBuilder::fbb_ (C++ member)`, 479
`dv::PoseBuilder::Finish (C++ function)`, 479
`dv::PoseBuilder::operator= (C++ function)`, 479
`dv::PoseBuilder::PoseBuilder (C++ function)`, 479
`dv::PoseBuilder::start_ (C++ member)`, 479
`dv::PoseFlatbuffer (C++ struct)`, 479
`dv::PoseFlatbuffer::GetFullyQualifiedName (C++ function)`, 480
`dv::PoseFlatbuffer::identifier (C++ member)`, 480
`dv::PoseFlatbuffer::MiniReflectTypeTable (C++ function)`, 480
`dv::PoseFlatbuffer::NativeTableType (C++ type)`, 479
`dv::PoseFlatbuffer::Pack (C++ function)`, 480
`dv::PoseFlatbuffer::referenceFrame (C++ function)`, 479
`dv::PoseFlatbuffer::rotation (C++ function)`, 479
`dv::PoseFlatbuffer::targetFrame (C++ function)`, 479
`dv::PoseFlatbuffer::timestamp (C++ function)`, 479
`dv::PoseFlatbuffer::translation (C++ function)`, 479
`dv::PoseFlatbuffer::UnPack (C++ function)`, 479
`dv::PoseFlatbuffer::UnPackTo (C++ function)`, 479
`dv::PoseFlatbuffer::UnPackToFrom (C++ function)`, 480
`dv::PoseFlatbuffer::Verify (C++ function)`, 479
`dv::PoseIdentifier (C++ function)`, 595
`dv::PoseTypeTable (C++ function)`, 594
`dv::QuaternionTypeTable (C++ function)`, 591
`dv::roiFilter (C++ function)`, 583
`dv::runtime_assert (C++ function)`, 582
`dv::scale (C++ function)`, 583
`dv::SemiDenseStereoMatcher (C++ class)`, 500
`dv::SemiDenseStereoMatcher::compute (C++ function)`, 502
`dv::SemiDenseStereoMatcher::computeDisparity (C++ function)`, 501
`dv::SemiDenseStereoMatcher::estimateDepth (C++ function)`, 502
`dv::SemiDenseStereoMatcher::estimateDepthFrame (C++ function)`, 503
`dv::SemiDenseStereoMatcher::getLeftFrame (C++ function)`, 502
`dv::SemiDenseStereoMatcher::getRightFrame (C++ function)`, 502
`dv::SemiDenseStereoMatcher::mLeftAccumulator (C++ member)`, 503
`dv::SemiDenseStereoMatcher::mLeftFrame (C++ member)`, 503
`dv::SemiDenseStereoMatcher::mMatcher (C++ member)`, 503
`dv::SemiDenseStereoMatcher::mRightAccumulator (C++ member)`, 503
`dv::SemiDenseStereoMatcher::mRightFrame (C++ member)`, 503
`dv::SemiDenseStereoMatcher::mStereoGeometry (C++ member)`, 503
`dv::SemiDenseStereoMatcher::SemiDenseStereoMatcher (C++ function)`, 500, 501
`dv::SemiDenseStereoMatcher::validateStereoGeometry (C++ function)`, 503
`dv::SparseEventBlockMatcher (C++ class)`, 512
`dv::SparseEventBlockMatcher::computeDisparitySparse (C++ function)`, 512
`dv::SparseEventBlockMatcher::getLeftFrame (C++ function)`, 513
`dv::SparseEventBlockMatcher::getLeftMask (C++ function)`, 513
`dv::SparseEventBlockMatcher::getMaxDisparity (C++ function)`, 513
`dv::SparseEventBlockMatcher::getMinDisparity (C++ function)`, 513
`dv::SparseEventBlockMatcher::getMinScore (C++ function)`, 514
`dv::SparseEventBlockMatcher::getPointRoi (C++ function)`, 514
`dv::SparseEventBlockMatcher::getRightFrame (C++ function)`, 513
`dv::SparseEventBlockMatcher::getRightMask (C++ function)`, 513
`dv::SparseEventBlockMatcher::getWindowSize (C++ function)`, 513
`dv::SparseEventBlockMatcher::initializeMaskPoint (C++ function)`, 514
`dv::SparseEventBlockMatcher::mHalfWindowSize (C++ member)`, 515

dv::SparseEventBlockMatcher::mLeftAcc (C++ member), 514
 dv::SparseEventBlockMatcher::mLeftFrame (C++ member), 514
 dv::SparseEventBlockMatcher::mLeftMask (C++ member), 514
 dv::SparseEventBlockMatcher::mMaxDisparity (C++ member), 515
 dv::SparseEventBlockMatcher::mMinDisparity (C++ member), 515
 dv::SparseEventBlockMatcher::mMinScore (C++ member), 515
 dv::SparseEventBlockMatcher::mRightAcc (C++ member), 514
 dv::SparseEventBlockMatcher::mRightFrame (C++ member), 514
 dv::SparseEventBlockMatcher::mRightMask (C++ member), 514
 dv::SparseEventBlockMatcher::mStereoGeometry (C++ member), 515
 dv::SparseEventBlockMatcher::mWindowSize (C++ member), 515
 dv::SparseEventBlockMatcher::PixelDisparity (C++ struct), 475
 dv::SparseEventBlockMatcher::PixelDisparity::coordinates (C++ member), 476
 dv::SparseEventBlockMatcher::PixelDisparity::correlation (C++ member), 476
 dv::SparseEventBlockMatcher::PixelDisparity::disparity (C++ member), 476
 dv::SparseEventBlockMatcher::PixelDisparity::matchedPosition (C++ member), 476
 dv::SparseEventBlockMatcher::PixelDisparity::PixelDisparity (C++ function), 475
 dv::SparseEventBlockMatcher::PixelDisparity::score (C++ member), 476
 dv::SparseEventBlockMatcher::PixelDisparity::templatePosition (C++ member), 476
 dv::SparseEventBlockMatcher::PixelDisparity::valid (C++ member), 476
 dv::SparseEventBlockMatcher::setMaxDisparity (C++ function), 513
 dv::SparseEventBlockMatcher::setMinDisparity (C++ function), 514
 dv::SparseEventBlockMatcher::setMinScore (C++ function), 514
 dv::SparseEventBlockMatcher::setWindowSize (C++ function), 513
 dv::SparseEventBlockMatcher::SparseEventBlockMatcher (C++ function), 512
 dv::SpeedInvariantTimeSurface (C++ type), 577
 dv::SpeedInvariantTimeSurfaceBase (C++ class), 515
 dv::SpeedInvariantTimeSurfaceBase::accept (C++ function), 516
 dv::SpeedInvariantTimeSurfaceBase::BaseClassType (C++ type), 516
 dv::SpeedInvariantTimeSurfaceBase::mLatestPixelValue (C++ member), 516
 dv::SpeedInvariantTimeSurfaceBase::operator<< (C++ function), 515
 dv::SpeedInvariantTimeSurfaceBase::SpeedInvariantTimeSurfaceBase (C++ function), 515
 dv::std_function_exact (C++ struct), 516
 dv::std_function_exact<R (Args...)> (C++ struct), 516
 dv::std_function_exact<R (Args...)>::std_function_exact (C++ function), 516
 dv::StereoEventStreamSlicer (C++ type), 577
 dv::StreamSlicer (C++ class), 530
 dv::StreamSlicer::accept (C++ function), 530, 531
 dv::StreamSlicer::doEveryNumberOfElements (C++ function), 531
 dv::StreamSlicer::doEveryTimeInterval (C++ function), 531
 dv::StreamSlicer::evaluate (C++ function), 532
 dv::StreamSlicer::hasJob (C++ function), 532
 dv::StreamSlicer::mHashCounter (C++ member), 533
 dv::StreamSlicer::modifyNumberInterval (C++ function), 532
 dv::StreamSlicer::modifyTimeInterval (C++ function), 532
 dv::StreamSlicer::mSliceJobs (C++ member), 533
 dv::StreamSlicer::mStorePacket (C++ member), 533
 dv::StreamSlicer::removeJob (C++ function), 532
 dv::StreamSlicer::SliceJob (C++ class), 508
 dv::StreamSlicer::SliceJob::mCallback (C++ member), 509
 dv::StreamSlicer::SliceJob::mLastCallEnd (C++ member), 509
 dv::StreamSlicer::SliceJob::mLastCallEndTime (C++ member), 509
 dv::StreamSlicer::SliceJob::mNumberInterval (C++ member), 509
 dv::StreamSlicer::SliceJob::mTimeInterval (C++ member), 509
 dv::StreamSlicer::SliceJob::mType (C++ member), 509
 dv::StreamSlicer::SliceJob::run (C++ function), 509

tion), 508
 dv::StreamSlicer::SliceJob::setNumberInterval (C++ function), 509
 dv::StreamSlicer::SliceJob::setTimeInterval (C++ function), 508
 dv::StreamSlicer::SliceJob::sliceByNumber (C++ function), 509
 dv::StreamSlicer::SliceJob::sliceByTime (C++ function), 509
 dv::StreamSlicer::SliceJob::SliceJob (C++ function), 508
 dv::StreamSlicer::SliceJob::SliceType (C++ enum), 508
 dv::StreamSlicer::SliceJob::SliceType::NUMBER (C++ enumerator), 508
 dv::StreamSlicer::SliceJob::SliceType::TIME (C++ enumerator), 508
 dv::StreamSlicer::StreamSlicer (C++ function), 530
 dv::TimedKeyPoint (C++ struct), 537
 dv::TimedKeyPoint::angle (C++ member), 537
 dv::TimedKeyPoint::class_id (C++ member), 537
 dv::TimedKeyPoint::GetFullyQualifiedName (C++ function), 538
 dv::TimedKeyPoint::octave (C++ member), 537
 dv::TimedKeyPoint::pt (C++ member), 537
 dv::TimedKeyPoint::response (C++ member), 537
 dv::TimedKeyPoint::size (C++ member), 537
 dv::TimedKeyPoint::TableType (C++ type), 537
 dv::TimedKeyPoint::TimedKeyPoint (C++ function), 537
 dv::TimedKeyPoint::timestamp (C++ member), 537
 dv::TimedKeyPointBuilder (C++ struct), 538
 dv::TimedKeyPointBuilder::add_angle (C++ function), 538
 dv::TimedKeyPointBuilder::add_class_id (C++ function), 538
 dv::TimedKeyPointBuilder::add_octave (C++ function), 538
 dv::TimedKeyPointBuilder::add_pt (C++ function), 538
 dv::TimedKeyPointBuilder::add_response (C++ function), 538
 dv::TimedKeyPointBuilder::add_size (C++ function), 538
 dv::TimedKeyPointBuilder::add_timestamp (C++ function), 538
 dv::TimedKeyPointBuilder::fbb_ (C++ member), 538
 dv::TimedKeyPointBuilder::Finish (C++ function), 538
 dv::TimedKeyPointBuilder::operator= (C++ function), 538
 dv::TimedKeyPointBuilder::start_ (C++ member), 538
 dv::TimedKeyPointBuilder::TimedKeyPointBuilder (C++ function), 538
 dv::TimedKeyPointFlatbuffer (C++ struct), 538
 dv::TimedKeyPointFlatbuffer::angle (C++ function), 538
 dv::TimedKeyPointFlatbuffer::class_id (C++ function), 539
 dv::TimedKeyPointFlatbuffer::GetFullyQualifiedName (C++ function), 539
 dv::TimedKeyPointFlatbuffer::MiniReflectTypeTable (C++ function), 539
 dv::TimedKeyPointFlatbuffer::NativeTableType (C++ type), 538
 dv::TimedKeyPointFlatbuffer::octave (C++ function), 539
 dv::TimedKeyPointFlatbuffer::Pack (C++ function), 539
 dv::TimedKeyPointFlatbuffer::pt (C++ function), 538
 dv::TimedKeyPointFlatbuffer::response (C++ function), 538
 dv::TimedKeyPointFlatbuffer::size (C++ function), 538
 dv::TimedKeyPointFlatbuffer::timestamp (C++ function), 539
 dv::TimedKeyPointFlatbuffer::UnPack (C++ function), 539
 dv::TimedKeyPointFlatbuffer::UnPackTo (C++ function), 539
 dv::TimedKeyPointFlatbuffer::UnPackToFrom (C++ function), 539
 dv::TimedKeyPointFlatbuffer::Verify (C++ function), 539
 dv::TimedKeyPointPacket (C++ struct), 539
 dv::TimedKeyPointPacket::elements (C++ member), 539
 dv::TimedKeyPointPacket::GetFullyQualifiedName (C++ function), 539
 dv::TimedKeyPointPacket::operator<< (C++ function), 540
 dv::TimedKeyPointPacket::TableType (C++ type), 539
 dv::TimedKeyPointPacket::TimedKeyPointPacket (C++ function), 539
 dv::TimedKeyPointPacketBufferHasIdentifier (C++ function), 596
 dv::TimedKeyPointPacketBuilder (C++ struct), 540
 dv::TimedKeyPointPacketBuilder::add_elements (C++ function), 540
 dv::TimedKeyPointPacketBuilder::fbb_ (C++ member), 540

dv::TimedKeyPointPacketBuilder::Finish (C++ function), 540
 dv::TimedKeyPointPacketBuilder::operator= (C++ function), 540
 dv::TimedKeyPointPacketBuilder::start_ (C++ member), 540
 dv::TimedKeyPointPacketBuilder::TimedKeyPointPacketBuilder (C++ function), 540
 dv::TimedKeyPointPacketFlatbuffer (C++ struct), 540
 dv::TimedKeyPointPacketFlatbuffer::elements (C++ function), 540
 dv::TimedKeyPointPacketFlatbuffer::GetFullyQualifiedName (C++ function), 540
 dv::TimedKeyPointPacketFlatbuffer::Identifier (C++ member), 541
 dv::TimedKeyPointPacketFlatbuffer::MiniReflectTypeTable (C++ function), 540
 dv::TimedKeyPointPacketFlatbuffer::NativeTableType (C++ type), 540
 dv::TimedKeyPointPacketFlatbuffer::Pack (C++ function), 540
 dv::TimedKeyPointPacketFlatbuffer::UnPack (C++ function), 540
 dv::TimedKeyPointPacketFlatbuffer::UnPackTo (C++ function), 540
 dv::TimedKeyPointPacketFlatbuffer::UnPackToFrom (C++ function), 540
 dv::TimedKeyPointPacketFlatbuffer::Verify (C++ function), 540
 dv::TimedKeyPointPacketIdentifier (C++ function), 596
 dv::TimedKeyPointPacketTypeTable (C++ function), 595
 dv::TimedKeyPointTypeTable (C++ function), 595
 dv::TimePoint (C++ type), 577
 dv::TimeSlicingApproach (C++ enum), 578
 dv::TimeSlicingApproach::BACKWARD (C++ enumerator), 578
 dv::TimeSlicingApproach::FORWARD (C++ enumerator), 578
 dv::TimestampClock (C++ type), 577
 dv::TimestampResolution (C++ type), 577
 dv::TimeSurface (C++ type), 577
 dv::TimeSurfaceBase (C++ class), 541
 dv::TimeSurfaceBase::~~TimeSurfaceBase (C++ function), 542
 dv::TimeSurfaceBase::accept (C++ function), 542
 dv::TimeSurfaceBase::addImpl (C++ function), 546
 dv::TimeSurfaceBase::at (C++ function), 542, 543
 dv::TimeSurfaceBase::block (C++ function), 543
 dv::TimeSurfaceBase::cols (C++ function), 546
 dv::TimeSurfaceBase::generateFrame (C++ function), 544
 dv::TimeSurfaceBase::getOCVMat (C++ function), 544
 dv::TimeSurfaceBase::getOCVMatScaled (C++ function), 544
 dv::TimeSurfaceBase::isEmpty (C++ function), 546
 dv::TimeSurfaceBase::mData (C++ member), 546
 dv::TimeSurfaceBase::operator- (C++ function), 545
 dv::TimeSurfaceBase::operator() (C++ function), 543
 dv::TimeSurfaceBase::operator+ (C++ function), 545
 dv::TimeSurfaceBase::operator+= (C++ function), 545
 dv::TimeSurfaceBase::operator<< (C++ function), 542
 dv::TimeSurfaceBase::operator= (C++ function), 545
 dv::TimeSurfaceBase::operator-= (C++ function), 545
 dv::TimeSurfaceBase::operator>> (C++ function), 542
 dv::TimeSurfaceBase::reset (C++ function), 545
 dv::TimeSurfaceBase::rows (C++ function), 546
 dv::TimeSurfaceBase::Scalar (C++ type), 541
 dv::TimeSurfaceBase::size (C++ function), 546
 dv::TimeSurfaceBase::TimeSurfaceBase (C++ function), 541, 542
 dv::TimeWindow (C++ struct), 546
 dv::TimeWindow::duration (C++ function), 546
 dv::TimeWindow::endTime (C++ member), 547
 dv::TimeWindow::startTime (C++ member), 547
 dv::TimeWindow::TimeWindow (C++ function), 546
 dv::toTimePoint (C++ function), 584
 dv::Trigger (C++ struct), 551
 dv::Trigger::GetFullyQualifiedName (C++ function), 552
 dv::Trigger::TableType (C++ type), 551
 dv::Trigger::timestamp (C++ member), 552
 dv::Trigger::Trigger (C++ function), 552
 dv::Trigger::type (C++ member), 552
 dv::TriggerBuilder (C++ struct), 552
 dv::TriggerBuilder::add_timestamp (C++ function), 552
 dv::TriggerBuilder::add_type (C++ function), 552
 dv::TriggerBuilder::fbb_ (C++ member), 552
 dv::TriggerBuilder::Finish (C++ function), 552
 dv::TriggerBuilder::operator= (C++ function), 552
 dv::TriggerBuilder::start_ (C++ member), 552

dv::TriggerBuilder::TriggerBuilder (C++ function), 552
 dv::TriggerFlatbuffer (C++ struct), 552
 dv::TriggerFlatbuffer::GetFullyQualifiedName (C++ function), 553
 dv::TriggerFlatbuffer::MiniReflectTypeTable (C++ function), 553
 dv::TriggerFlatbuffer::NativeTableType (C++ type), 552
 dv::TriggerFlatbuffer::Pack (C++ function), 553
 dv::TriggerFlatbuffer::timestamp (C++ function), 552
 dv::TriggerFlatbuffer::type (C++ function), 552
 dv::TriggerFlatbuffer::UnPack (C++ function), 552
 dv::TriggerFlatbuffer::UnPackTo (C++ function), 553
 dv::TriggerFlatbuffer::UnPackToFrom (C++ function), 553
 dv::TriggerFlatbuffer::Verify (C++ function), 552
 dv::TriggerPacket (C++ struct), 553
 dv::TriggerPacket::elements (C++ member), 553
 dv::TriggerPacket::GetFullyQualifiedName (C++ function), 553
 dv::TriggerPacket::operator<< (C++ function), 553
 dv::TriggerPacket::TableType (C++ type), 553
 dv::TriggerPacket::TriggerPacket (C++ function), 553
 dv::TriggerPacketBufferHasIdentifier (C++ function), 597
 dv::TriggerPacketBuilder (C++ struct), 553
 dv::TriggerPacketBuilder::add_elements (C++ function), 553
 dv::TriggerPacketBuilder::fbb_ (C++ member), 554
 dv::TriggerPacketBuilder::Finish (C++ function), 553
 dv::TriggerPacketBuilder::operator= (C++ function), 553
 dv::TriggerPacketBuilder::start_ (C++ member), 554
 dv::TriggerPacketBuilder::TriggerPacketBuilder (C++ function), 553
 dv::TriggerPacketFlatbuffer (C++ struct), 554
 dv::TriggerPacketFlatbuffer::elements (C++ function), 554
 dv::TriggerPacketFlatbuffer::GetFullyQualifiedName (C++ function), 554
 dv::TriggerPacketFlatbuffer::identifier (C++ member), 554
 dv::TriggerPacketFlatbuffer::MiniReflectTypeTable (C++ function), 554
 dv::TriggerPacketFlatbuffer::NativeTableType (C++ type), 554
 dv::TriggerPacketFlatbuffer::Pack (C++ function), 554
 dv::TriggerPacketFlatbuffer::UnPack (C++ function), 554
 dv::TriggerPacketFlatbuffer::UnPackTo (C++ function), 554
 dv::TriggerPacketFlatbuffer::UnPackToFrom (C++ function), 554
 dv::TriggerPacketFlatbuffer::Verify (C++ function), 554
 dv::TriggerPacketIdentifier (C++ function), 597
 dv::TriggerPacketTypeTable (C++ function), 596
 dv::TriggerStreamSlicer (C++ type), 577
 dv::TriggerType (C++ enum), 581
 dv::TriggerType::APS_EXPOSURE_END (C++ enumerator), 582
 dv::TriggerType::APS_EXPOSURE_START (C++ enumerator), 582
 dv::TriggerType::APS_FRAME_END (C++ enumerator), 581
 dv::TriggerType::APS_FRAME_START (C++ enumerator), 581
 dv::TriggerType::EXTERNAL_GENERATOR_FALLING_EDGE (C++ enumerator), 581
 dv::TriggerType::EXTERNAL_GENERATOR_RISING_EDGE (C++ enumerator), 581
 dv::TriggerType::EXTERNAL_SIGNAL_FALLING_EDGE (C++ enumerator), 581
 dv::TriggerType::EXTERNAL_SIGNAL_PULSE (C++ enumerator), 581
 dv::TriggerType::EXTERNAL_SIGNAL_RISING_EDGE (C++ enumerator), 581
 dv::TriggerType::MAX (C++ enumerator), 582
 dv::TriggerType::MIN (C++ enumerator), 582
 dv::TriggerType::TIMESTAMP_RESET (C++ enumerator), 581
 dv::TriggerTypeTable (C++ function), 596
 dv::TriggerTypeTypeTable (C++ function), 597
 dv::types (C++ type), 629
 dv::types::ConstructPtr (C++ type), 629
 dv::types::DestructPtr (C++ type), 629
 dv::types::IdentifierStringToId (C++ function), 629
 dv::types::IdToIdentifierString (C++ function), 629
 dv::types::makeTypeDefinition (C++ function), 629
 dv::types::Packer (C++ function), 629
 dv::types::PackFuncPtr (C++ type), 629
 dv::types::TimeElementExtractor (C++ struct),

- 541
- dv::types::TimeElementExtractor::~~TimeElementExtractor (C++ function), 541
- dv::types::TimeElementExtractor::endTimeStamp (C++ member), 541
- dv::types::TimeElementExtractor::numElements (C++ member), 541
- dv::types::TimeElementExtractor::operator= (C++ function), 541
- dv::types::TimeElementExtractor::operator!= (C++ function), 541
- dv::types::TimeElementExtractor::operator== (C++ function), 541
- dv::types::TimeElementExtractor::startTimestamp (C++ member), 541
- dv::types::TimeElementExtractor::TimeElementExtractor (C++ function), 541
- dv::types::TimeElementExtractorDefault (C++ function), 629
- dv::types::TimeElementExtractorPtr (C++ type), 629
- dv::types::TimeRangeExtractorDefault (C++ function), 629
- dv::types::TimeRangeExtractorPtr (C++ type), 629
- dv::types::Type (C++ struct), 554
- dv::types::Type::~~Type (C++ function), 554
- dv::types::Type::construct (C++ member), 555
- dv::types::Type::destruct (C++ member), 555
- dv::types::Type::id (C++ member), 555
- dv::types::Type::operator= (C++ function), 555
- dv::types::Type::operator!= (C++ function), 555
- dv::types::Type::operator== (C++ function), 555
- dv::types::Type::pack (C++ member), 555
- dv::types::Type::sizeofType (C++ member), 555
- dv::types::Type::timeElementExtractor (C++ member), 555
- dv::types::Type::timeRangeExtractor (C++ member), 555
- dv::types::Type::Type (C++ function), 554, 555
- dv::types::Type::unpack (C++ member), 555
- dv::types::TypedObject (C++ struct), 555
- dv::types::TypedObject::~~TypedObject (C++ function), 555
- dv::types::TypedObject::moveToSharedPtr (C++ function), 555
- dv::types::TypedObject::obj (C++ member), 556
- dv::types::TypedObject::operator= (C++ function), 555
- dv::types::TypedObject::operator!= (C++ function), 555
- dv::types::TypedObject::operator== (C++ function), 555
- dv::types::TypedObject::type (C++ member),
- 556
- dv::types::TypedObject::TypedObject (C++ function), 555
- dv::types::Unpacker (C++ function), 629
- dv::types::UnpackFuncPtr (C++ type), 629
- dv::UnPackBoundingBoxPacket (C++ function), 587
- dv::UnPackDepthEventPacket (C++ function), 588
- dv::UnPackDepthFrame (C++ function), 589
- dv::UnPackEventPacket (C++ function), 590
- dv::UnPackFileDataTable (C++ function), 599
- dv::UnPackFrame (C++ function), 591
- dv::UnPackIMUPacket (C++ function), 593
- dv::UnPackIOHeader (C++ function), 600
- dv::UnPackLandmarksPacket (C++ function), 594
- dv::UnPackPose (C++ function), 595
- dv::UnPackTimedKeyPointPacket (C++ function), 596
- dv::UnPackTriggerPacket (C++ function), 597
- dv::Vec2fTypeTable (C++ function), 591
- dv::Vec3fTypeTable (C++ function), 591
- dv::vectorContains (C++ function), 585
- dv::vectorContainsIf (C++ function), 585
- dv::vectorRemove (C++ function), 585
- dv::vectorRemoveIf (C++ function), 585
- dv::vectorSortUnique (C++ function), 585
- dv::VerifyBoundingBoxPacketBuffer (C++ function), 587
- dv::VerifyDepthEventPacketBuffer (C++ function), 588
- dv::VerifyDepthFrameBuffer (C++ function), 589
- dv::VerifyEventPacketBuffer (C++ function), 589
- dv::VerifyFileDataTableBuffer (C++ function), 598
- dv::VerifyFrameBuffer (C++ function), 591
- dv::VerifyIMUPacketBuffer (C++ function), 592
- dv::VerifyIOHeaderBuffer (C++ function), 600
- dv::VerifyLandmarksPacketBuffer (C++ function), 594
- dv::VerifyPoseBuffer (C++ function), 595
- dv::VerifySizePrefixedBoundingBoxPacketBuffer (C++ function), 587
- dv::VerifySizePrefixedDepthEventPacketBuffer (C++ function), 588
- dv::VerifySizePrefixedDepthFrameBuffer (C++ function), 589
- dv::VerifySizePrefixedEventPacketBuffer (C++ function), 589
- dv::VerifySizePrefixedFileDataTableBuffer (C++ function), 599
- dv::VerifySizePrefixedFrameBuffer (C++ function), 591
- dv::VerifySizePrefixedIMUPacketBuffer (C++ function), 592

dv::VerifySizePrefixedIOHeaderBuffer (C++ function), 600

dv::VerifySizePrefixedLandmarksPacketBuffer (C++ function), 594

dv::VerifySizePrefixedPoseBuffer (C++ function), 595

dv::VerifySizePrefixedTimedKeyPointPacketBuffer (C++ function), 596

dv::VerifySizePrefixedTriggerPacketBuffer (C++ function), 597

dv::VerifyTimedKeyPointPacketBuffer (C++ function), 596

dv::VerifyTriggerPacketBuffer (C++ function), 597

dv::VERSION (C++ member), 600

dv::VERSION_MAJOR (C++ member), 600

dv::VERSION_MINOR (C++ member), 600

dv::VERSION_PATCH (C++ member), 600

dv::VERSION_STRING (C++ member), 600

dv::visualization (C++ type), 629

dv::visualization::colors (C++ type), 629

dv::visualization::colors::black (C++ member), 630

dv::visualization::colors::blue (C++ member), 630

dv::visualization::colors::cyan (C++ member), 630

dv::visualization::colors::darkGray (C++ member), 630

dv::visualization::colors::gray (C++ member), 630

dv::visualization::colors::green (C++ member), 630

dv::visualization::colors::iniBlue (C++ member), 630

dv::visualization::colors::lime (C++ member), 630

dv::visualization::colors::magenta (C++ member), 630

dv::visualization::colors::maroon (C++ member), 630

dv::visualization::colors::navy (C++ member), 630

dv::visualization::colors::neonPalette (C++ member), 630

dv::visualization::colors::red (C++ member), 630

dv::visualization::colors::silver (C++ member), 630

dv::visualization::colors::someNeonColor (C++ function), 630

dv::visualization::colors::white (C++ member), 630

dv::visualization::colors::yellow (C++ member), 630

dv::visualization::EventVisualizer (C++ class), 338

dv::visualization::EventVisualizer::backgroundColor (C++ member), 340

dv::visualization::EventVisualizer::EventVisualizer (C++ function), 338

dv::visualization::EventVisualizer::generateImage (C++ function), 339

dv::visualization::EventVisualizer::getBackgroundColor (C++ function), 339

dv::visualization::EventVisualizer::getNegativeColor (C++ function), 339

dv::visualization::EventVisualizer::getPositiveColor (C++ function), 339

dv::visualization::EventVisualizer::negativeColor (C++ member), 340

dv::visualization::EventVisualizer::positiveColor (C++ member), 340

dv::visualization::EventVisualizer::resolution (C++ member), 340

dv::visualization::EventVisualizer::setBackgroundColor (C++ function), 339

dv::visualization::EventVisualizer::setNegativeColor (C++ function), 339

dv::visualization::EventVisualizer::setPositiveColor (C++ function), 339

dv::visualization::PoseVisualizer (C++ class), 480

dv::visualization::PoseVisualizer::accept (C++ function), 482, 483

dv::visualization::PoseVisualizer::addText (C++ function), 488

dv::visualization::PoseVisualizer::clearAllObjects (C++ function), 483

dv::visualization::PoseVisualizer::clearLandmarks (C++ function), 483

dv::visualization::PoseVisualizer::clearLandmarksInInterval (C++ function), 483

dv::visualization::PoseVisualizer::clearLandmarkWithId (C++ function), 483

dv::visualization::PoseVisualizer::clearObject (C++ function), 483

dv::visualization::PoseVisualizer::clearObjectPosesInInterval (C++ function), 483

dv::visualization::PoseVisualizer::computeSpan (C++ function), 488

dv::visualization::PoseVisualizer::DEFAULT_CAMERA_HEIGHT (C++ member), 491

dv::visualization::PoseVisual-
 izer::DEG_TO_RAD (C++ *member*), 492
 dv::visualization::PoseVisualizer::drawCo-
 ordinateFrame (C++ *function*), 489
 dv::visualization::PoseVisualizer::draw-
 Grid (C++ *function*), 489
 dv::visualization::PoseVisualizer::draw-
 Landmarks (C++ *function*), 490
 dv::visualization::PoseVisualizer::drawLe-
 gend (C++ *function*), 490
 dv::visualization::PoseVisualizer::draw-
 Line (C++ *function*), 489
 dv::visualization::PoseVisualizer::drawOb-
 jectPoses (C++ *function*), 490
 dv::visualization::PoseVisualizer::draw-
 Point (C++ *function*), 489
 dv::visualization::PoseVisualizer::draw-
 WorldCoordinateSystem (C++ *function*),
 490
 dv::visualization::PoseVisualizer::gener-
 ateFrame (C++ *function*), 483
 dv::visualization::PoseVisualizer::get-
 BackgroundColor (C++ *function*), 486
 dv::visualization::PoseVisualizer::getCam-
 eraIntrinsics (C++ *function*), 484
 dv::visualization::PoseVisualizer::getCam-
 eraOrientation (C++ *function*), 484
 dv::visualization::PoseVisualizer::getCam-
 eraPose (C++ *function*), 485
 dv::visualization::PoseVisualizer::getCam-
 eraPosition (C++ *function*), 484
 dv::visualization::PoseVisualizer::get-
 GridColor (C++ *function*), 487
 dv::visualization::PoseVisualizer::get-
 GridHalfWidth (C++ *function*), 488
 dv::visualization::PoseVisualizer::get-
 GridPlane (C++ *function*), 485
 dv::visualization::PoseVisualizer::get-
 GridSpacing (C++ *function*), 486
 dv::visualization::PoseVisualizer::get-
 LandmarkLimit (C++ *function*), 487
 dv::visualization::PoseVisualizer::get-
 LineThickness (C++ *function*), 486
 dv::visualization::PoseVisualizer::getNum-
 berOfLandmarks (C++ *function*), 487
 dv::visualization::PoseVisualizer::getNum-
 berOfObjects (C++ *function*), 487
 dv::visualization::PoseVisualizer::getO-
 bjectCoordinateSize (C++ *function*),
 486
 dv::visualization::PoseVisualizer::getRes-
 olution (C++ *function*), 483
 dv::visualization::PoseVisualizer::get-
 TextSize (C++ *function*), 488
 dv::visualization::PoseVisualizer::get-
 Timestamp (C++ *function*), 483
 dv::visualization::PoseVisual-
 izer::getViewMode (C++ *function*),
 485
 dv::visualization::PoseVisualizer::get-
 WorldCoordinateSize (C++ *function*),
 486
 dv::visualization::PoseVisual-
 izer::GRID_SPACING_GROWTH_FACTOR
 (C++ *member*), 491
 dv::visualization::PoseVisualizer::Grid-
 Plane (C++ *enum*), 480
 dv::visualization::PoseVisualizer::Grid-
 Plane::PLANE_NONE (C++ *enumerator*),
 480
 dv::visualization::PoseVisualizer::Grid-
 Plane::PLANE_XY (C++ *enumerator*), 481
 dv::visualization::PoseVisualizer::Grid-
 Plane::PLANE_YZ (C++ *enumerator*), 481
 dv::visualization::PoseVisualizer::Grid-
 Plane::PLANE_ZX (C++ *enumerator*), 481
 dv::visualization::PoseVisual-
 izer::isAutoScalingEnabled (C++
 function), 487
 dv::visualization::PoseVisualizer::isLe-
 gendVisualizationEnabled (C++ *func-*
 tion), 487
 dv::visualization::PoseVisualizer::isWith-
 inGridBounds (C++ *function*), 488
 dv::visualization::PoseVisual-
 izer::LIMIT_NUM_GRID_LINES (C++
 member), 492
 dv::visualization::PoseVisualizer::mBack-
 groundColor (C++ *member*), 490
 dv::visualization::PoseVisualizer::mBound-
 sSet (C++ *member*), 491
 dv::visualization::PoseVisualizer::mCamer-
 aPosition (C++ *member*), 490
 dv::visualization::PoseVisual-
 izer::mEnableAutoScale (C++ *member*),
 490
 dv::visualization::PoseVisual-
 izer::mEnableLegend (C++ *member*),
 491
 dv::visualization::PoseVisualizer::mGrid-
 Color (C++ *member*), 490
 dv::visualization::PoseVisualizer::mGrid-
 Plane (C++ *member*), 490
 dv::visualization::PoseVisual-
 izer::mGridSpacing (C++ *member*),
 490
 dv::visualization::PoseVisualizer::mLand-
 markLimit (C++ *member*), 491

dv::visualization::PoseVisualizer::mLandmarks (C++ member), 491

dv::visualization::PoseVisualizer::mLineThickness (C++ member), 491

dv::visualization::PoseVisualizer::mMaxBoundX (C++ member), 491

dv::visualization::PoseVisualizer::mMaxBoundY (C++ member), 491

dv::visualization::PoseVisualizer::mMaxBoundZ (C++ member), 491

dv::visualization::PoseVisualizer::mMinBoundX (C++ member), 491

dv::visualization::PoseVisualizer::mMinBoundY (C++ member), 491

dv::visualization::PoseVisualizer::mMinBoundZ (C++ member), 491

dv::visualization::PoseVisualizer::mObjectCoordinateFrameSize (C++ member), 491

dv::visualization::PoseVisualizer::mObjectTrajectories (C++ member), 491

dv::visualization::PoseVisualizer::mT_C_W (C++ member), 490

dv::visualization::PoseVisualizer::mTrajectoryLength (C++ member), 491

dv::visualization::PoseVisualizer::mViewingCameraGeometry (C++ member), 490

dv::visualization::PoseVisualizer::mViewMode (C++ member), 490

dv::visualization::PoseVisualizer::mWorldCoordinateFrameSize (C++ member), 491

dv::visualization::PoseVisualizer::operator<< (C++ function), 492

dv::visualization::PoseVisualizer::PIXEL_PITCH (C++ member), 491

dv::visualization::PoseVisualizer::project (C++ function), 489

dv::visualization::PoseVisualizer::RAD_TO_DEG (C++ member), 492

dv::visualization::PoseVisualizer::reset (C++ function), 483

dv::visualization::PoseVisualizer::setAutoScaling (C++ function), 487

dv::visualization::PoseVisualizer::setBackgroundColors (C++ function), 486

dv::visualization::PoseVisualizer::setCameraExtrinsicsFromViewMode (C++ function), 488

dv::visualization::PoseVisualizer::setCameraIntrinsics (C++ function), 484

dv::visualization::PoseVisualizer::setCameraOrientation (C++ function), 485

dv::visualization::PoseVisualizer::setCameraPose (C++ function), 485

dv::visualization::PoseVisualizer::setCameraPosition (C++ function), 484

dv::visualization::PoseVisualizer::setGridColor (C++ function), 487

dv::visualization::PoseVisualizer::setGridPlane (C++ function), 485

dv::visualization::PoseVisualizer::setGridSpacing (C++ function), 486

dv::visualization::PoseVisualizer::setLandmarkLimit (C++ function), 487

dv::visualization::PoseVisualizer::setLegendVisualization (C++ function), 487

dv::visualization::PoseVisualizer::setLineThickness (C++ function), 486

dv::visualization::PoseVisualizer::setObjectCoordinateSize (C++ function), 486

dv::visualization::PoseVisualizer::setResolution (C++ function), 484

dv::visualization::PoseVisualizer::setViewMode (C++ function), 485

dv::visualization::PoseVisualizer::setWorldCoordinateSize (C++ function), 486

dv::visualization::PoseVisualizer::updateBounds (C++ function), 488

dv::visualization::PoseVisualizer::updateGridSpacing (C++ function), 488

dv::visualization::PoseVisualizer::ViewMode (C++ enum), 480

dv::visualization::PoseVisualizer::ViewMode::CUSTOM (C++ enumerator), 480

dv::visualization::PoseVisualizer::ViewMode::VIEW_XY (C++ enumerator), 480

dv::visualization::PoseVisualizer::ViewMode::VIEW_XZ (C++ enumerator), 480

dv::visualization::PoseVisualizer::ViewMode::VIEW_YX (C++ enumerator), 480

dv::visualization::PoseVisualizer::ViewMode::VIEW_YZ (C++ enumerator), 480

dv::visualization::PoseVisualizer::ViewMode::VIEW_ZX (C++ enumerator), 480

dv::visualization::PoseVisualizer::ViewMode::VIEW_ZY (C++ enumerator), 480

dv_capture_node (C++ type), 630

DV_LIBUSB_VERSION_1_0_23 (C macro), 641

DV_LIBUSB_VERSION_1_0_24 (C macro), 641

DV_LIBUSB_VERSION_1_0_25_26 (C macro), 641

DV_LIBUSB_VERSION_1_0_27_28 (C macro), 641

DV_LIBUSB_VERSION_1_0_29 (C macro), 641

DV_PROCESSING_NAME_STRING (C macro), 646

DV_PROCESSING_VERSION (*C macro*), 646
DV_PROCESSING_VERSION_MAJOR (*C macro*), 646
DV_PROCESSING_VERSION_MINOR (*C macro*), 646
DV_PROCESSING_VERSION_PATCH (*C macro*), 646
DV_PROCESSING_VERSION_STRING (*C macro*), 646
dv_runtime (*C++ type*), 630

F

flatbuffers (*C++ type*), 630
fmt (*C++ type*), 631
fmt::formatter<cv::Mat> (*C++ struct*), 355
fmt::formatter<cv::Mat>::format (*C++ function*), 355
fmt::formatter<cv::Point> (*C++ struct*), 355
fmt::formatter<cv::Rect> (*C++ struct*), 355
fmt::formatter<cv::Size> (*C++ struct*), 355
fmt::formatter<dv::Accumulator::Decay> (*C++ struct*), 355
fmt::formatter<dv::BoundingBoxPacket> (*C++ struct*), 355
fmt::formatter<dv::camera::CameraGeometry::FunctionImplementation> (*C++ struct*), 355
fmt::formatter<dv::camera::DistortionModel> (*C++ struct*), 355
fmt::formatter<dv::camera::StereoGeometry::CameraPosition> (*C++ struct*), 355
fmt::formatter<dv::camera::StereoGeometry::FunctionImplementation> (*C++ struct*), 355
fmt::formatter<dv::DepthEventPacket> (*C++ struct*), 355
fmt::formatter<dv::DepthFrame> (*C++ struct*), 355
fmt::formatter<dv::EdgeMapAccumulator::DecayMode> (*C++ struct*), 355
fmt::formatter<dv::Event> (*C++ struct*), 355
fmt::formatter<dv::EventColor> (*C++ struct*), 356
fmt::formatter<dv::EventPacket> (*C++ struct*), 356
fmt::formatter<dv::EventStore> (*C++ struct*), 356
fmt::formatter<dv::Frame> (*C++ struct*), 356
fmt::formatter<dv::IMUPacket> (*C++ struct*), 356
fmt::formatter<dv::io::camera::CameraInputBase::Flatten> (*C++ struct*), 356
fmt::formatter<dv::io::camera::CameraModel> (*C++ struct*), 356
fmt::formatter<dv::io::camera::DVXplorer::ReadoutFPS> (*C++ struct*), 356
fmt::formatter<dv::io::camera::DVXplorer::SubSample> (*C++ struct*), 356
fmt::formatter<dv::io::camera::imu::ImuModel> (*C++ struct*), 356
fmt::formatter<dv::io::camera::era::parser::DAVIS::ColorMode> (*C++ struct*), 356
fmt::formatter<dv::io::camera::era::parser::DAVIS::SensorModel> (*C++ struct*), 356
fmt::formatter<dv::io::camera::USBDevice::LogLevel> (*C++ struct*), 356
fmt::formatter<dv::io::camera::USBDeviceType> (*C++ struct*), 356
fmt::formatter<dv::io::support::VariantValueOwning> (*C++ class*), 356
fmt::formatter<dv::io::support::VariantValueOwning>::format (*C++ function*), 357
fmt::formatter<dv::io::support::VariantValueOwning>::FORMATTER_MAX_LEN (*C++ member*), 357
fmt::formatter<dv::io::support::VariantValueOwning>::mFmtForward (*C++ member*), 357
fmt::formatter<dv::io::support::VariantValueOwning>::parse (*C++ function*), 357
fmt::formatter<dv::LandmarksPacket> (*C++ struct*), 357
fmt::formatter<dv::noise::FrequencyFilterType> (*C++ struct*), 357
fmt::formatter<dv::PixelArrangement> (*C++ struct*), 357
fmt::formatter<dv::Pose> (*C++ struct*), 357
fmt::formatter<dv::TimedKeyPointPacket> (*C++ struct*), 357
fmt::formatter<dv::TimeSlicingApproach> (*C++ struct*), 357
fmt::formatter<dv::TriggerPacket> (*C++ struct*), 357
fmt::formatter<dv::visualization::PoseVisualizer::GridPlane> (*C++ struct*), 357
fmt::formatter<dv::visualization::PoseVisualizer::ViewMode> (*C++ struct*), 357
L
LZ4F_HEADER_SIZE_MAX (*C macro*), 641, 642
O
operator<< (*C++ function*), 632

P

PACKED_STRUCT (*C macro*), [644](#)

S

std (*C++ type*), [631](#)

Z

ZSTD_CLEVEL_DEFAULT (*C macro*), [641](#), [642](#)