
DepthAI API Docs

Release

2.2.1.0.dev+d99fa42233320b513fe8d4a4f87b8d7f5e63ab29

Luxonis

Apr 27, 2021

CONTENT:

1	What is Gen2?	3
2	Basic glossary	5
3	Getting started	7
3.1	Installation	7
3.2	Overview	13
3.3	Device	14
3.4	Bootloader	28
3.5	Pipeline	30
3.6	Nodes	34
3.7	Messages	99
3.8	Hello World	130
3.9	Multiple DepthAI per Host	135
3.10	Local OpenVINO Model Conversion	138
3.11	Code samples	145
3.12	Python API Reference	247
3.13	C++ API Reference	360
	Python Module Index	417
	Index	419

On this page you can find the details regarding the Gen2 DepthAI API that will allow you to interact with the DepthAI device. We support both *Python API* and *C++ API*

WHAT IS GEN2?

Gen2 is a step forward in DepthAI integration, allowing users to define their own flow of data using pipelines, nodes and connections. Gen2 was created based on user's feedback from Gen1 and from raising capabilities of both DepthAI and supporting software like OpenVINO.

BASIC GLOSSARY

- **Host side** is the device, like PC or RPi, to which the DepthAI is connected to. If something is happening on the host side, it means that this device is involved in it, not DepthAI itself
- **Device side** is the DepthAI itself. If something is happening on the device side, it means that the DepthAI is responsible for it
- **Pipeline** is a complete workflow on the device side, consisting of nodes and connections between them - these cannot exist outside of pipeline.
- **Node** is a single functionality of the DepthAI. It have either inputs or outputs or both, together with properties to be defined (like resolution on the camera node or blob path in neural network node)
- **Connection** is a link between one node's output and another one's input. In order to define the pipeline dataflow, the connections define where to send data in order to achieve an expected result
- **XLink** is a middleware that is capable to exchange data between device and host. XLinkIn node allows to send the data from host to device, XLinkOut does the opposite.

GETTING STARTED

To help you get started with Gen2 API, we have prepared multiple examples of it's usage, with more yet to come, together with some insightful tutorials.

Before running the example, install the DepthAI Python library using the command below

```
python3 -m pip install -U --force-reinstall depthai
```

Now, pick a tutorial or code sample and start utilizing Gen2 capabilities

3.1 Installation

Please *install the necessary dependencies* for your platform by referring to the table below. Once installed you can *install the DepthAI library*.

We are constantly striving to improve how we release our software to keep up with countless platforms and the numerous ways to package it. If you do not see a particular platform or package format listed below please reach out to us on [Discord](#) or on [Github](#).

3.1.1 Supported Platforms

We keep up-to-date, pre-compiled, libraries for the following platforms. Note that a new change is that for Ubuntu now also work unchanged for the Jetson/Xavier series:

Platform	Instructions	Support
Windows 10	Platform dependencies	Discord
macOS	Platform dependencies	Discord
Ubuntu & Jetson/Xavier	Platform dependencies	Discord
Raspberry Pi OS	Platform dependencies	Discord

And the following platforms are also supported by a combination of the community and Luxonis.

Platform	Instructions	Support
Fedora		Discord
Robot Operating System		Discord
Windows 7	WinUSB driver	Discord
Docker	Pull and run official images	Discord
Kernel Virtual Machine	Run on KVM	Discord

macOS

```
bash -c "$(curl -fL http://docs.luxonis.com/_static/install_dependencies.sh)"
```

Close and re-open the terminal window after this command.

The script also works on M1 Macs, Homebrew being installed under Rosetta 2, as some Python packages are still missing native M1 support. In case you already have Homebrew installed natively and things don't work, see [here](#) for some additional troubleshooting steps.

Note that if the video streaming window does not appear consider running the following:

```
python3 -m pip install opencv-python --force-reinstall --no-cache-dir
```

See the [Video preview window fails to appear on macOS](#) thread on our forum for more information.

Raspberry Pi OS

```
sudo curl -fL http://docs.luxonis.com/_static/install_dependencies.sh | bash
```

Ubuntu

These Ubuntu instructions also work for the **Jetson** and **Xavier** series.

```
sudo wget -qO- http://docs.luxonis.com/_static/install_dependencies.sh | bash
```

Note! If opencv fails with illegal instruction after installing from PyPi, add:

```
echo "export OPENBLAS_CORETYPE=ARMV8" >> ~/.bashrc
source ~/.bashrc
```

openSUSE

For openSUSE, available in [this official article](#) how to install the OAK device on the openSUSE platform.

Windows

We recommend using the Chocolatey package manager to install DepthAI's dependencies on Windows. Chocolatey is very similar to Homebrew for macOS.

To [install Chocolatey](#) and use it to install DepthAI's dependencies do the following:

- Right click on *Start*
- Choose *Windows PowerShell (Admin)* and run the following:

```
Set-ExecutionPolicy Bypass -Scope Process -Force; [System.Net.
↪ServicePointManager]::SecurityProtocol = [System.Net.
↪ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object System.Net.
↪WebClient).DownloadString('https://chocolatey.org/install.ps1'))
```

- Close the PowerShell and then re-open another PowerShell (Admin) by repeating the first two steps.
- Install Python and PyCharm

```
choco install cmake git python pycharm-community -y
```

Windows 7

Although we do not officially support Windows 7, members of the community [have had success](#) manually installing WinUSB using [Zadig](#). After connecting your DepthAI device look for a device with USB ID: 03E7 2485 and install the WinUSB driver by selecting *WinUSB(v6.1.7600.16385)* and then *Install WCID Driver*.

Docker

We maintain a Docker image containing DepthAI, its dependencies and helpful tools in the [luxonis/depthai-library](#) repository on Docker Hub. It builds upon the [luxonis/depthai-base](#) image.

Run the `01_rgb_preview.py` example inside a Docker container on a Linux host (with the X11 windowing system):

```
docker pull luxonis/depthai-library
docker run --rm \
  --privileged \
  -v /dev/bus/usb:/dev/bus/usb \
  --device-cgroup-rule='c 189:* rmw' \
  -e DISPLAY=$DISPLAY \
  -v /tmp/.X11-unix:/tmp/.X11-unix \
  luxonis/depthai-library:latest \
  python3 /depthai-python/examples/01_rgb_preview.py
```

To allow the container to update X11 you may need to run `xhost local:root` on the host.

KVM

To access the OAK-D camera in the [Kernel Virtual Machine](#), there is a need to attach and detach USB devices on the fly when the host machine detects changes in the USB bus.

OAK-D camera changes the USB device type when it is used by DepthAI API. This happens in background when the camera is used natively. But when the camera is used in a virtual environment the situation is different.

On your host machine, use the following code:

```
SUBSYSTEM=="usb", ACTION=="bind", ENV{ID_VENDOR_ID}=="03e7", MODE="0666", RUN+="/usr/
↳ local/bin/movidius_usb_hotplug.sh depthai-vm"
SUBSYSTEM=="usb", ACTION=="remove", ENV{PRODUCT}=="3e7/2485/1", ENV{DEVTYPE}=="usb_
↳ device", MODE="0666", RUN+="/usr/local/bin/movidius_usb_hotplug.sh depthai-vm"
SUBSYSTEM=="usb", ACTION=="remove", ENV{PRODUCT}=="3e7/f63b/100", ENV{DEVTYPE}=="usb_
↳ device", MODE="0666", RUN+="/usr/local/bin/movidius_usb_hotplug.sh depthai-vm"
```

The script that the udev rule is calling (`movidius_usb_hotplug.sh`) should then attach/detach the USB device to the virtual machine. In this case we need to call `virsh` command. For example, the script could do the following:

```
#!/bin/bash
# Abort script execution on errors
set -e
if [ "${ACTION}" == 'bind' ]; then
  COMMAND='attach-device'
elif [ "${ACTION}" == 'remove' ]; then
```

(continues on next page)

(continued from previous page)

```

COMMAND='detach-device'
if [ "${PRODUCT}" == '3e7/2485/1' ]; then
    ID_VENDOR_ID=03e7
    ID_MODEL_ID=2485
fi
if [ "${PRODUCT}" == '3e7/f63b/100' ]; then
    ID_VENDOR_ID=03e7
    ID_MODEL_ID=f63b
fi
else
    echo "Invalid udev ACTION: ${ACTION}" >&2
    exit 1
fi
echo "Running virsh ${COMMAND} ${DOMAIN} for ${ID_VENDOR_ID}." >&2
virsh "${COMMAND}" "${DOMAIN}" /dev/stdin <<END
<hostdev mode='subsystem' type='usb'>
  <source>
    <vendor id='0x${ID_VENDOR_ID}'/>
    <product id='0x${ID_MODEL_ID}'/>
  </source>
</hostdev>
END
exit 0

```

Note that when the device is disconnected from the USB bus, some udev environmental variables are not available (ID_VENDOR_ID or ID_MODEL_ID), that is why you need to use PRODUCT environmental variable to identify which device has been disconnected.

The virtual machine where DepthAI API application is running should have defined a udev rules that identify the OAK-D camera. The udev rule is described [here](#)

Solution provided by [Manuel Segarra-Abad](#)

3.1.2 Install from PyPI

Our packages are distributed [via PyPI](#), to install it in your environment use

```
python3 -m pip install depthai
```

For other installation options, see [other installation options](#).

3.1.3 Test installation

We have [a set of examples](#) that should help you verify if your setup was correct.

First, clone the [depthai-python](#) repository and change directory into this repo:

```
git clone https://github.com/luxonis/depthai-python.git
cd depthai-python
```

Next install the requirements for this repository. Note that we recommend installing the dependencies in a virtual environment, so that they don't interfere with other Python tools/environments on your system.

- For development machines like Mac/Windows/Ubuntu/etc., we recommend the [PyCharm](#) IDE, as it automatically makes/manages virtual environments for you, along with a bunch of other benefits. Alternatively, `conda`, `pipenv`, or `virtualenv` could be used directly (and/or with your preferred IDE).

- For installations on resource-constrained systems, such as the Raspberry Pi or other small Linux systems, we recommend `conda`, `pipenv`, or `virtualenv`. To set up a virtual environment with `virtualenv`, run `virtualenv venv && source venv/bin/activate`.

Using a virtual environment (or system-wide, if you prefer), run the following to install the requirements for this example repository:

```
cd examples
python3 install_requirements.py
```

Now, run the `01_rgb_preview.py` script from within `examples` directory to make sure everything is working:

```
python3 01_rgb_preview.py
```

If all goes well a small window video display should appear. An example is shown below:

3.1.4 Run Other Examples

After you have run this example, you can run other examples to learn about DepthAI possibilities. You can also proceed to:

- Our tutorials, starting with a Hello World tutorial explaining the API usage step by step ([here](#))
- Our experiments, containing implementations of various user use cases on DepthAI ([here](#))

You can also proceed below to learn how to convert your own neural network to run on DepthAI.

And we also have online model training below, which shows you how to train and convert models for DepthAI:

- Online ML Training and model Conversion: [HERE](#)

3.1.5 Other installation methods

To get the latest and yet unreleased features from our source code, you can go ahead and compile `depthai` package manually.

Dependencies to build from source

- CMake > 3.2.0
- Generation tool (Ninja, make, ...)
- C/C++ compiler
- `libusb1` development package

Ubuntu, Raspberry Pi OS, ... (Debian based systems)

On Debian based systems (Raspberry Pi OS, Ubuntu, ...) these can be acquired by running:

```
sudo apt-get -y install cmake libusb-1.0-0-dev build-essential
```

macOS (Mac OS X)

Assuming a stock Mac OS X install, `depthai-python` library needs following dependencies

- Homebrew (If it's not installed already)

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/  
↪master/install.sh)"
```

- Python, libusb, CMake, wget

```
brew install coreutils python3 cmake libusb wget
```

And now you're ready to clone the `depthai-python` from Github and build it for Mac OS X.

Install using GitHub commit

Pip allows users to install the packages from specific commits, even if they are not yet released on PyPi.

To do so, use the command below - and be sure to replace the `<commit_sha>` with the correct commit hash [from here](#)

```
python3 -m pip install git+https://github.com/luxonis/depthai-python.git@<commit_sha>
```

Using/Testing a Specific Branch/PR

From time to time, it may be of interest to use a specific branch. This may occur, for example, because we have listened to your feature request and implemented a quick implementation in a branch. Or it could be to get early access to a feature that is soaking in our `develop` for stability purposes before being merged into `main` (`develop` is the branch we use to soak new features before merging them into `main`):

So when working in the `depthai-python` repository, using a branch can be accomplished with the following commands.

Prior to running the following, you can either clone the repository independently (for not over-writing any of your local changes) or simply do a `git pull` first.

```
git checkout <branch>  
git submodule update --init --recursive  
python3 setup.py develop
```

Install from source

If desired, you can also install the package from the source code itself - it will allow you to make the changes to the API and see them live in action.

To do so, first download the repository and then add the package to your python interpreter in development mode

```
git clone https://github.com/luxonis/depthai-python.git  
cd depthai-python  
git submodule update --init --recursive  
python3 setup.py develop # you may need to add sudo if using system interpreter_  
↪instead of virtual environment
```

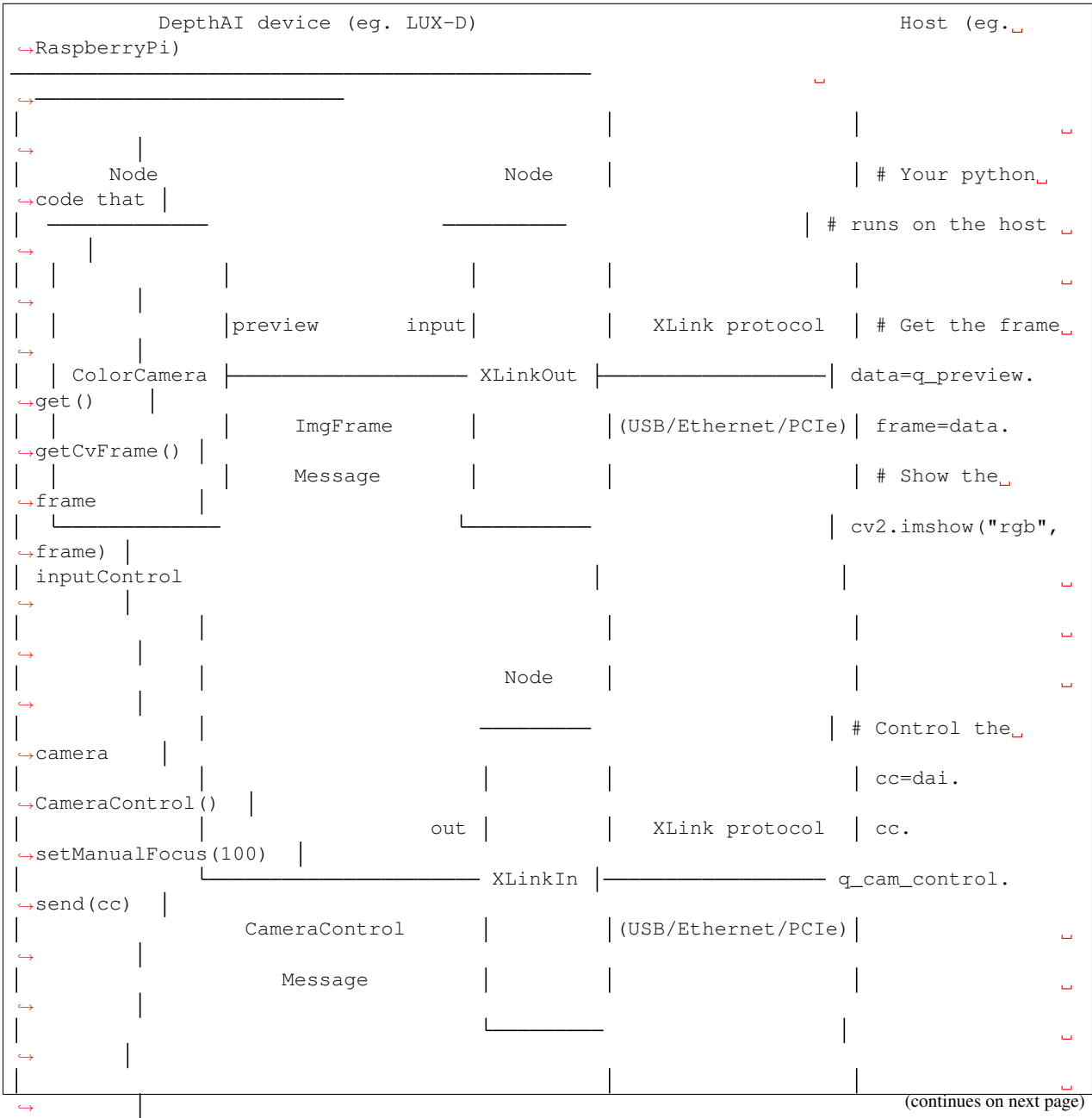
If you want to use other branch (e.g. `develop`) than default (`main`), you can do so by typing


```
git checkout develop # replace the "develop" with a desired branch name
git submodule update --recursive
python3 setup.py develop
```

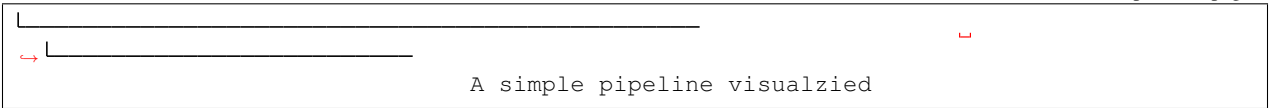
Or, if you want to checkout a specific commit, type

```
git checkout <commit_sha>
git submodule update --recursive
python3 setup.py develop
```

3.2 Overview



(continued from previous page)



3.2.1 Device

Device is the [DepthAI module](#) itself. On the device there is a powerful vision processing unit (VPU) from Intel, called [Myriad X](#) (MX for short). The VPU is optimized for performing AI inference algorithms and for processing sensory inputs (eg. calculating stereo disparity from two cameras).

For more details, click [here](#)

3.2.2 Pipeline

The upper flowchart is a simple pipeline visualized. So a **pipeline is collection of nodes and links** between them.

For more details, click [here](#)

3.2.3 Nodes

Each node provides a specific functionality on the DepthAI, a set of configurable properties and inputs/outputs. On the flowchart above, we have 3 nodes; ColorCamera, XLinkOut and XLinkIn.

For more details, click [here](#)

3.2.4 Messages

Messages are sent between linked nodes. On the flowchart above, there are two links - visualized as arrows that are inside the device. There are a few different types of messages, on the chart we have `ImgFrame` and `CameraControl`.

For more details, click [here](#)

We're always happy to help with code or other questions you might have.

3.3 Device

Device is a [DepthAI module](#). After the [Pipeline](#) is defined, it can be uploaded to the device. When you create the device in the code, firmware is uploaded together with the pipeline.

```
pipeline = depthai.Pipeline()

# Create nodes, configure them and link them together

# Upload the pipeline to the device
with depthai.Device(pipeline) as device:
    # Start the pipeline that is now on the device
    device.startPipeline()

    # Input queue, to send message from the host to the device (you can receive the
    # message on the device with XLinkIn)
```

(continues on next page)

(continued from previous page)

```

input_q = device.getInputQueue("input_name", maxSize=4, blocking=False)

# Output queue, to receive message on the host from the device (you can send the_
↪message on the device with XLinkOut)
output_q = device.getOutputQueue("output_name", maxSize=4, blocking=False)

while True:
    # Get the message from the queue
    output_q.get() # Or output_q.tryGet() for non-blocking

    # Send a message to the device
    cfg = depthai.ImageManipConfig()
    input_q.send(cfg)

```

3.3.1 Multiple devices

If you want to use multiple devices on a host, check [Multiple DepthAI per Host](#).

3.3.2 Device queues

After initializing the device, one has to initialize the input/output queues as well. You can create an input queue with `device.getInputQueue("input_name")` and output queue with `device.getOutputQueue("output_name")`.

When you define an output queue, the device can write to it at any point in time, and the host can read from it at any point in time. There might be a cases when the host is reading very fast from the queue, and the queue, no matter its size, will stay empty most of the time. But as we add things on the host side (additional processing, analysis etc), it may happen that the device will be writing to the queue faster than the host can read from it. And then the packets in the queue will start to add up - and both `maxSize` and `blocking` flags determine the behavior of the queue in this case.

By default, the queue is blocking and its size is 30, so the device will put 30 packets at most, and when the limit is reached, it will hang on queue put call and wait until it can successfully complete this call (so, waits for the host to consume the packet before putting a new one). Making the queue non-blocking will change its behavior in this situation - instead of waiting, it will discard the oldest packet and add the new one, and then continue its processing loop (so it won't get blocked). `maxSize` determines the size of the queue and also helps to control the memory usage - if the packet have 5MB of data, and the queue size is 30, this queue effectively stores 150MB of data in memory (the packets can also get really big, for instance a single 4K NV12 encoded frame takes about ~12MB). Decreasing the queue size to 1 and setting non-blocking behavior will effectively mean "I only want the latest packet from the queue".

The size and behavior of the queue can be modified after the initialization by calling `queue.setBlocking()` and `queue.setMaxSize()`.

3.3.3 Reference

Python

class `depthai.Device`

Represents the DepthAI device with the methods to interact with it.

addLogCallback (*self: depthai.Device, callback: std::function<void (dai::LogMessage)>*) → *int*

Add a callback for device logging. The callback will be called from a separate thread with the `LogMessage` being passed.

Parameter callback:

- Callback to call whenever a log message arrives

Returns Id which can be used to later remove the callback

close (*self*: [depthai.Device](#)) → [None](#)

Closes the connection to device. Better alternative is the usage of context manager: *with depthai.Device(pipeline) as device:*

static getAllAvailableDevices () → List[[depthai.DeviceInfo](#)]

Returns all connected devices

Returns vector of connected devices

static getAnyAvailableDevice (*args, **kwargs)

Overloaded function.

1. `getAnyAvailableDevice(timeout: datetime.timedelta) -> Tuple[bool, depthai.DeviceInfo]`

Waits for any available device with a timeout

Parameter timeout:

- duration of time to wait for the any device

Returns a tuple of bool and DeviceInfo. Bool specifies if device was found. DeviceInfo specifies the found device

2. `getAnyAvailableDevice() -> Tuple[bool, depthai.DeviceInfo]`

Gets any available device

Returns a tuple of bool and DeviceInfo. Bool specifies if device was found. DeviceInfo specifies the found device

getChipTemperature (*self*: [depthai.Device](#)) → [dai::ChipTemperature](#)

Retrieves current chip temperature as measured by device

Returns Temperature of various onboard sensors

getCmxMemoryUsage (*self*: [depthai.Device](#)) → [dai::MemoryInfo](#)

Retrieves current CMX memory information from device

Returns Used, remaining and total cmx memory

getDdrMemoryUsage (*self*: [depthai.Device](#)) → [dai::MemoryInfo](#)

Retrieves current DDR memory information from device

Returns Used, remaining and total ddr memory

static getDeviceByMxId (*mxId*: [str](#)) → Tuple[bool, [depthai.DeviceInfo](#)]

Finds a device by MX ID. Example: 14442C10D13EABCE00

Parameter mxId:

- MyraidX ID which uniquely specifies a device

Returns a tuple of bool and DeviceInfo. Bool specifies if device was found. DeviceInfo specifies the found device

static getEmbeddedDeviceBinary (*usb2Mode*: bool, *version*: [depthai.OpenVINO.Version](#) = [<Version.VERSION_2021_3: 6>](#)) → List[int]

Gets device firmware binary for a specific OpenVINO version

Parameter usb2Mode:

- USB2 mode firmware

Parameter version:

- Version of OpenVINO which firmware will support

Returns firmware binary

static getFirstAvailableDevice () → Tuple[bool, *depthai.DeviceInfo*]

Gets first available device. Device can be either in XLINK_UNBOOTED or XLINK_BOOTLOADER state

Returns a tuple of bool and DeviceInfo. Bool specifies if device was found. DeviceInfo specifies the found device

getInputQueue (*args, **kwargs)

Overloaded function.

1. getInputQueue(self: depthai.Device, name: str) -> dai::DataInputQueue

Gets an input queue corresponding to stream name. If it doesn't exist it throws

Parameter name: Queue/stream name, set in XLinkIn node

Returns Smart pointer to DataInputQueue

2. getInputQueue(self: depthai.Device, name: str, maxSize: int, blocking: bool = True) -> dai::DataInputQueue

Gets an input queue corresponding to stream name. If it doesn't exist it throws. Also sets queue options

Parameter name: Queue/stream name, set in XLinkOut node

Parameter maxSize: Maximum number of messages in queue

Parameter blocking: Queue behavior once full. True: blocking, false: overwriting of oldest messages. Default: true

Returns Smart pointer to DataInputQueue

getInputQueueNames (self: *depthai.Device*) → List[str]

Get all available input queue names

Returns Vector of input queue names

getLeonCssCpuUsage (self: *depthai.Device*) → dai::CpuUsage

Retrieves average CSS Leon CPU usage

Returns Average CPU usage and sampling duration

getLeonCssHeapUsage (self: *depthai.Device*) → dai::MemoryInfo

Retrieves current CSS Leon CPU heap information from device

Returns Used, remaining and total heap memory

getLeonMssCpuUsage (self: *depthai.Device*) → dai::CpuUsage

Retrieves average MSS Leon CPU usage

Returns Average CPU usage and sampling duration

getLeonMssHeapUsage (self: *depthai.Device*) → dai::MemoryInfo

Retrieves current MSS Leon CPU heap information from device

Returns Used, remaining and total heap memory

getLogLevel (*self*: [depthai.Device](#)) → [dai::LogLevel](#)

Gets current logging severity level of the device.

Returns Logging severity level

getLogLevel (*self*: [depthai.Device](#)) → [dai::LogLevel](#)

Gets logging level which decides printing level to standard output.

Returns Standard output printing severity

getOutputQueue (**args*, ***kwargs*)

Overloaded function.

1. `getOutputQueue(self: depthai.Device, name: str) -> dai::DataOutputQueue`

Gets an output queue corresponding to stream name. If it doesn't exist it throws

Parameter name: Queue/stream name, created by XLinkOut node

Returns Smart pointer to [DataOutputQueue](#)

2. `getOutputQueue(self: depthai.Device, name: str, maxSize: int, blocking: bool = True) -> dai::DataOutputQueue`

Gets a queue corresponding to stream name, if it exists, otherwise it throws. Also sets queue options

Parameter name: Queue/stream name, set in XLinkOut node

Parameter maxSize: Maximum number of messages in queue

Parameter blocking: Queue behavior once full. True specifies blocking and false overwriting of oldest messages. Default: true

Returns Smart pointer to [DataOutputQueue](#)

getOutputQueueNames (*self*: [depthai.Device](#)) → [List\[str\]](#)

Get all available output queue names

Returns Vector of output queue names

getQueueEvent (**args*, ***kwargs*)

Overloaded function.

1. `getQueueEvent(self: depthai.Device, queueNames: List\[str\], timeout: datetime.timedelta = datetime.timedelta\(days=-1, seconds=86399, microseconds=999999\)) -> str`

Gets or waits until any of specified queues has received a message

Parameter queueNames: Names of queues for which to wait for

Parameter timeout: Timeout after which return regardless. If negative then wait is indefinite. Default is -1

Returns Queue name which received a message first

2. `getQueueEvent(self: depthai.Device, queueName: str, timeout: datetime.timedelta = datetime.timedelta\(days=-1, seconds=86399, microseconds=999999\)) -> str`

Gets or waits until specified queue has received a message

Parameter queueNames: Name of queues for which to wait for

Parameter timeout: Timeout after which return regardless. If negative then wait is indefinite. Default is -1

Returns Queue name which received a message

3. `getQueueEvent(self: depthai.Device, timeout: datetime.timedelta = datetime.timedelta(days=-1, seconds=86399, microseconds=999999)) -> str`

Gets or waits until any queue has received a message

Parameter timeout: Timeout after which return regardless. If negative then wait is indefinite. Default is -1

Returns Queue name which received a message

getQueueEvents (*args, **kwargs)

Overloaded function.

1. `getQueueEvents(self: depthai.Device, queueNames: List[str], maxNumEvents: int = 18446744073709551615, timeout: datetime.timedelta = datetime.timedelta(days=-1, seconds=86399, microseconds=999999)) -> List[str]`

Gets or waits until any of specified queues has received a message

Parameter queueNames: Names of queues for which to block

Parameter maxNumEvents: Maximum number of events to remove from queue - Default is unlimited

Parameter timeout: Timeout after which return regardless. If negative then wait is indefinite - Default is -1

Returns Names of queues which received messages first

2. `getQueueEvents(self: depthai.Device, queueName: str, maxNumEvents: int = 18446744073709551615, timeout: datetime.timedelta = datetime.timedelta(days=-1, seconds=86399, microseconds=999999)) -> List[str]`

Gets or waits until specified queue has received a message

Parameter queueName: Name of queues for which to wait for

Parameter maxNumEvents: Maximum number of events to remove from queue. Default is unlimited

Parameter timeout: Timeout after which return regardless. If negative then wait is indefinite. Default is -1

Returns Names of queues which received messages first

3. `getQueueEvents(self: depthai.Device, maxNumEvents: int = 18446744073709551615, timeout: datetime.timedelta = datetime.timedelta(days=-1, seconds=86399, microseconds=999999)) -> List[str]`

Gets or waits until any queue has received a message

Parameter maxNumEvents: Maximum number of events to remove from queue. Default is unlimited

Parameter timeout: Timeout after which return regardless. If negative then wait is indefinite. Default is -1

Returns Names of queues which received messages first

getSystemInformationLoggingRate (*self*: [depthai.Device](#)) → [float](#)

Gets current rate of system information logging (“info” severity) in Hz.

Returns Logging rate in Hz

isPipelineRunning (*self*: [depthai.Device](#)) → [bool](#)

Checks if devices pipeline is already running

Returns true if running, false otherwise

removeLogCallback (*self*: [depthai.Device](#), *callbackId*: [int](#)) → [bool](#)

Removes a callback

Parameter callbackId: Id of callback to be removed

Returns true if callback was removed, false otherwise

setLogLevel (*self*: [depthai.Device](#), *level*: [dai::LogLevel](#)) → [None](#)

Sets the devices logging severity level. This level affects which logs are transferred from device to host.

Parameter level: Logging severity

setLogOutputLevel (*self*: [depthai.Device](#), *level*: [dai::LogLevel](#)) → [None](#)

Sets logging level which decides printing level to standard output. If lower than setLogLevel, no messages will be printed

Parameter level:

- Standard output printing severity

setSystemInformationLoggingRate (*self*: [depthai.Device](#), *rateHz*: [float](#)) → [None](#)

Sets rate of system information logging (“info” severity). Default 1Hz If parameter is less or equal to zero, then system information logging will be disabled

Parameter rateHz: Logging rate in Hz

startPipeline (*self*: [depthai.Device](#)) → [bool](#)

Starts the execution of the devices pipeline

Returns true if pipeline started, false otherwise

C++

class [dai::Device](#)

Represents the DepthAI device with the methods to interact with it.

Public Functions

Device (**const** [Pipeline](#) &*pipeline*)

Connects to any available device with a DEFAULT_SEARCH_TIME timeout.

Parameters

- *pipeline*: - [Pipeline](#) to be executed on the device

Device (**const** [Pipeline](#) &*pipeline*, **bool** *usb2Mode*)

Connects to any available device with a DEFAULT_SEARCH_TIME timeout.

Parameters

- *pipeline*: - [Pipeline](#) to be executed on the device
- *usb2Mode*: - Boot device using USB2 mode firmware

Device (**const** *Pipeline* &pipeline, **const** char *pathToCmd)

Connects to any available device with a DEFAULT_SEARCH_TIME timeout.

Parameters

- pipeline: - *Pipeline* to be executed on the device
- pathToCmd: - Path to custom device firmware

Device (**const** *Pipeline* &pipeline, **const** std::string &pathToCmd)

Connects to any available device with a DEFAULT_SEARCH_TIME timeout.

Parameters

- pipeline: - *Pipeline* to be executed on the device
- pathToCmd: - Path to custom device firmware

Device (**const** *Pipeline* &pipeline, **const** *DeviceInfo* &devInfo, bool usb2Mode = false)

Connects to device specified by devInfo.

Parameters

- pipeline: - *Pipeline* to be executed on the device
- devInfo: - *DeviceInfo* which specifies which device to connect to
- usb2Mode: - Boot device using USB2 mode firmware

Device (**const** *Pipeline* &pipeline, **const** *DeviceInfo* &devInfo, **const** char *pathToCmd)

Connects to device specified by devInfo.

Parameters

- pipeline: - *Pipeline* to be executed on the device
- devInfo: - *DeviceInfo* which specifies which device to connect to
- pathToCmd: - Path to custom device firmware

Device (**const** *Pipeline* &pipeline, **const** *DeviceInfo* &devInfo, **const** std::string &pathToCmd)

Connects to device specified by devInfo.

Parameters

- pipeline: - *Pipeline* to be executed on the device
- devInfo: - *DeviceInfo* which specifies which device to connect to
- usb2Mode: - Path to custom device firmware

~Device ()

Device destructor. Closes the connection and data queues.

bool **isPipelineRunning** ()

Checks if devices pipeline is already running

Return true if running, false otherwise

bool **startPipeline** ()

Starts the execution of the devices pipeline

Return true if pipeline started, false otherwise

void **setLogLevel** (*LogLevel* level)

Sets the devices logging severity level. This level affects which logs are transferred from device to host.

Parameters

- `level`: Logging severity

LogLevel **getLogLevel** ()

Gets current logging severity level of the device.

Return Logging severity level

void **setLogLevel** (*LogLevel level*)

Sets logging level which decides printing level to standard output. If lower than `setLogLevel`, no messages will be printed

Parameters

- `level`: - Standard output printing severity

LogLevel **getLogOutputLevel** ()

Gets logging level which decides printing level to standard output.

Return Standard output printing severity

int **addLogCallback** (std::function<void> *LogMessage*

> *callback*) Add a callback for device logging. The callback will be called from a separate thread with the *LogMessage* being passed.

Return Id which can be used to later remove the callback

Parameters

- `callback`: - Callback to call whenever a log message arrives

bool **removeLogCallback** (int *callbackId*)

Removes a callback

Return true if callback was removed, false otherwise

Parameters

- `callbackId`: Id of callback to be removed

void **setSystemInformationLoggingRate** (float *rateHz*)

Sets rate of system information logging (“info” severity). Default 1Hz If parameter is less or equal to zero, then system information logging will be disabled

Parameters

- `rateHz`: Logging rate in Hz

float **getSystemInformationLoggingRate** ()

Gets current rate of system information logging (“info” severity) in Hz.

Return Logging rate in Hz

std::shared_ptr<*DataOutputQueue*> **getOutputQueue** (const std::string &*name*)

Gets an output queue corresponding to stream name. If it doesn’t exist it throws

Return Smart pointer to *DataOutputQueue*

Parameters

- name: Queue/stream name, created by XLinkOut node

`std::shared_ptr<DataOutputQueue> getOutputQueue (const std::string &name, unsigned int maxSize, bool blocking = true)`

Gets a queue corresponding to stream name, if it exists, otherwise it throws. Also sets queue options

Return Smart pointer to *DataOutputQueue*

Parameters

- name: Queue/stream name, set in XLinkOut node
- maxSize: Maximum number of messages in queue
- blocking: Queue behavior once full. True specifies blocking and false overwriting of oldest messages. Default: true

`std::vector<std::string> getOutputQueueNames () const`

Get all available output queue names

Return Vector of output queue names

`std::shared_ptr<DataInputQueue> getInputQueue (const std::string &name)`

Gets an input queue corresponding to stream name. If it doesn't exist it throws

Return Smart pointer to *DataInputQueue*

Parameters

- name: Queue/stream name, set in XLinkIn node

`std::shared_ptr<DataInputQueue> getInputQueue (const std::string &name, unsigned int maxSize, bool blocking = true)`

Gets an input queue corresponding to stream name. If it doesn't exist it throws. Also sets queue options

Return Smart pointer to *DataInputQueue*

Parameters

- name: Queue/stream name, set in XLinkOut node
- maxSize: Maximum number of messages in queue
- blocking: Queue behavior once full. True: blocking, false: overwriting of oldest messages. Default: true

`std::vector<std::string> getInputQueueNames () const`

Get all available input queue names

Return Vector of input queue names

`std::vector<std::string> getQueueEvents (const std::vector<std::string> &queueNames, std::size_t maxNumEvents = std::numeric_limits<std::size_t>::max(), std::chrono::microseconds timeout = std::chrono::microseconds(-1))`

Gets or waits until any of specified queues has received a message

Return Names of queues which received messages first

Parameters

- `queueNames`: Names of queues for which to block
- `maxNumEvents`: Maximum number of events to remove from queue - Default is unlimited
- `timeout`: Timeout after which return regardless. If negative then wait is indefinite - Default is -1

```
std::vector<std::string> getQueueEvents (const std::initializer_list<std::string>
                                         &queueNames, std::size_t maxNumEvents
                                         = std::numeric_limits<std::size_t>::max(),
                                         std::chrono::microseconds timeout =
                                         std::chrono::microseconds(-1))
```

```
std::vector<std::string> getQueueEvents (std::string queueName, std::size_t maxNumEvents
                                         = std::numeric_limits<std::size_t>::max(),
                                         std::chrono::microseconds timeout =
                                         std::chrono::microseconds(-1))
```

Gets or waits until specified queue has received a message

Return Names of queues which received messages first

Parameters

- `queueName`: Name of queues for which to wait for
- `maxNumEvents`: Maximum number of events to remove from queue. Default is unlimited
- `timeout`: Timeout after which return regardless. If negative then wait is indefinite. Default is -1

```
std::vector<std::string> getQueueEvents (std::size_t maxNumEvents
                                         = std::numeric_limits<std::size_t>::max(),
                                         std::chrono::microseconds timeout =
                                         std::chrono::microseconds(-1))
```

Gets or waits until any any queue has received a message

Return Names of queues which received messages first

Parameters

- `maxNumEvents`: Maximum number of events to remove from queue. Default is unlimited
- `timeout`: Timeout after which return regardless. If negative then wait is indefinite. Default is -1

```
std::string getQueueEvent (const std::vector<std::string> &queueNames, std::chrono::microseconds
                           timeout = std::chrono::microseconds(-1))
```

Gets or waits until any of specified queues has received a message

Return Queue name which received a message first

Parameters

- `queueNames`: Names of queues for which to wait for
- `timeout`: Timeout after which return regardless. If negative then wait is indefinite. Default is -1

```
std::string getQueueEvent (const          std::initializer_list<std::string>      &queueNames,  
                           std::chrono::microseconds timeout = std::chrono::microseconds(-1))
```

```
std::string getQueueEvent (std::string  queueName,      std::chrono::microseconds  timeout  =  
                           std::chrono::microseconds(-1))
```

Gets or waits until specified queue has received a message

Return Queue name which received a message

Parameters

- *queueNames*: Name of queues for which to wait for
- *timeout*: Timeout after which return regardless. If negative then wait is indefinite. Default is -1

```
std::string getQueueEvent (std::chrono::microseconds timeout = std::chrono::microseconds(-1))
```

Gets or waits until any queue has received a message

Return Queue name which received a message

Parameters

- *timeout*: Timeout after which return regardless. If negative then wait is indefinite. Default is -1

MemoryInfo **getDdrMemoryUsage** ()

Retrieves current DDR memory information from device

Return Used, remaining and total ddr memory

MemoryInfo **getCmxMemoryUsage** ()

Retrieves current CMX memory information from device

Return Used, remaining and total cmx memory

MemoryInfo **getLeonCssHeapUsage** ()

Retrieves current CSS Leon CPU heap information from device

Return Used, remaining and total heap memory

MemoryInfo **getLeonMssHeapUsage** ()

Retrieves current MSS Leon CPU heap information from device

Return Used, remaining and total heap memory

ChipTemperature **getChipTemperature** ()

Retrieves current chip temperature as measured by device

Return Temperature of various onboard sensors

CpuUsage **getLeonCssCpuUsage** ()

Retrieves average CSS Leon CPU usage

Return Average CPU usage and sampling duration

CpuUsage **getLeonMssCpuUsage** ()

Retrieves average MSS Leon CPU usage

Return Average CPU usage and sampling duration

void **close** ()

Explicitly closes connection to device.

Note This function does not need to be explicitly called as destructor closes the device automatically

bool **isClosed** () **const**

Is the device already closed (or disconnected)

Public Static Functions

template<typename **Rep**, typename **Period**>

std::tuple<bool, *DeviceInfo*> **getAnyAvailableDevice** (std::chrono::duration<*Rep*, *Period*> *timeout*)

Waits for any available device with a timeout

Return a tuple of bool and *DeviceInfo*. Bool specifies if device was found. *DeviceInfo* specifies the found device

Parameters

- *timeout*: - duration of time to wait for the any device

std::tuple<bool, *DeviceInfo*> **getAnyAvailableDevice** ()

Gets any available device

Return a tuple of bool and *DeviceInfo*. Bool specifies if device was found. *DeviceInfo* specifies the found device

std::tuple<bool, *DeviceInfo*> **getFirstAvailableDevice** ()

Gets first available device. *Device* can be either in XLINK_UNBOOTED or XLINK_BOOTLOADER state

Return a tuple of bool and *DeviceInfo*. Bool specifies if device was found. *DeviceInfo* specifies the found device

std::tuple<bool, *DeviceInfo*> **getDeviceByMxId** (std::string *mxId*)

Finds a device by MX ID. Example: 14442C10D13EABCE00

Return a tuple of bool and *DeviceInfo*. Bool specifies if device was found. *DeviceInfo* specifies the found device

Parameters

- *mxId*: - MyraidX ID which uniquely specifies a device

std::vector<*DeviceInfo*> **getAllAvailableDevices** ()

Returns all connected devices

Return vector of connected devices

std::vector<std::uint8_t> **getEmbeddedDeviceBinary** (bool *usb2Mode*, *OpenVINO::Version* *version* = *Pipeline::DEFAULT_OPENVINO_VERSION*)

Gets device firmware binary for a specific *OpenVINO* version

Return firmware binary

Parameters

- `usb2Mode`: - USB2 mode firmware
- `version`: - Version of *OpenVINO* which firmware will support

Public Static Attributes

constexpr std::chrono::seconds **DEFAULT_SEARCH_TIME** = {3}

Default search time for constructors which discover devices.

constexpr std::size_t **EVENT_QUEUE_MAXIMUM_SIZE** = {2048}

Maximum number of elements in event queue.

constexpr float **DEFAULT_SYSTEM_INFORMATION_LOGGING_RATE_HZ** = {1.0f}

Default rate at which system information is logged.

Private Functions

void **init**(const *Pipeline* &pipeline, bool *embeddedMvcm*, bool *usb2Mode*, const std::string &pathToMvcm)

void **checkClosed**() const

Private Members

std::shared_ptr<*XLinkConnection*> **connection**

std::unique_ptr<nanorpc::core::client<nanorpc::packer::nlohmann_msgpack>> **client**

std::mutex **rpcMutex**

std::vector<uint8_t> **patchedCmd**

DeviceInfo **deviceInfo** = {}

std::unordered_map<std::string, std::shared_ptr<*DataOutputQueue*>> **outputQueueMap**

std::unordered_map<std::string, std::shared_ptr<*DataInputQueue*>> **inputQueueMap**

std::unordered_map<std::string, *DataOutputQueue::CallbackId*> **callbackIdMap**

int **uniqueCallbackId** = 0

std::mutex **logCallbackMapMtx**

std::unordered_map<int, std::function<void (*LogMessage*)>> **logCallbackMap**

std::mutex **eventMtx**

std::condition_variable **eventCv**

std::deque<std::string> **eventQueue**

std::thread **watchdogThread**

std::atomic<bool> **watchdogRunning** = {true}

std::thread **timesyncThread**

std::atomic<bool> **timesyncRunning** = {true}

```
std::thread loggingThread
std::atomic<bool> loggingRunning = {true}
std::unique_ptr<XLinkStream> rpcStream
std::atomic<bool> closed = {false}
Pimpl<Impl> pimpl
PipelineSchema schema
Assets assets
std::vector<std::uint8_t> assetStorage
OpenVINO::Version version
```

We're always happy to help with code or other questions you might have.

3.4 Bootloader

Depthai bootloader is a small program which aids in booting and updating bootloader or depthai application packages.

To be able to run hostless, the Depthai bootloader must be first flashed to the devices flash. This step is required only once.

Plug USB to the board Flash bootloader using DeviceBootloader::flashBootloader (Check Example at the bottom) Disconnect the board and switch the boot mode GPIO to the following settings: BOOT[4:0] : 01000 (see attached images for reference) Reassemble the board

Once the device has the bootloader flashed, it will perform the same as before. Running pipelines with a host connected doesn't require any changes.

Suggested workflow is to perform as much of development as possible with the host connected as the iteration cycle is greatly improved.

Once desired pipeline is created, use the following function to flash: DeviceBootloader::flash

3.4.1 API

DeviceBootloader is a class to communicate with the bootloader. It is used to flash created *Pipeline*, depthai application package or update the bootloader itself.

progressCb parameter takes a callback function, which will be called each time an progress update occurs (rate limited to 1 second). This is mainly used to inform the user of the current flashing progress.

3.4.2 DepthAI Application Package (.dap)

Depthai application package is a binary file format which stores sections of data. The purpose of this format is to be able to extract individual sections and do OTA updates without requiring to update all data. Example: Between update 1 and 2 of users application, Depthai firmware, Asset storage (50MiB neural network) and asset structure remained the same, but some additional processing nodes were added to the pipeline. Instead of transferring the whole package only Pipeline description can be sent and updated.

Depthai application package (**.dap**) consists of:

- SBR (512B header which describes sections of data)
- Depthai device firmware (section “__firmware”)

- Pipeline description (section “pipeline”)
- Assets structure (section “assets”)
- Asset storage (section “asset_storage”)

3.4.3 Example

Following section will show an example of: Flashing bootloader (needed only once) and flashing a created Pipeline “myExamplePipeline” to the device (The example is written in Python, similar steps apply to C++)

1. Flashing bootloader

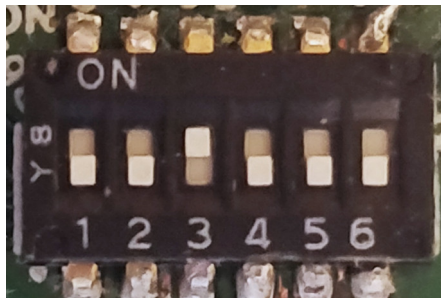
```
import depthai as dai
(f, bl) = dai.DeviceBootloader.getFirstAvailableDevice()
bootloader = dai.DeviceBootloader(bl)
progress = lambda p : print(f'Flashing progress: {p*100:.1f}%')
bootloader.flashBootloader(progress)
```

Note: Make sure to switch GPIO BOOT mode settings (See image below for more details)

2. Flashing created pipeline

```
import depthai as dai
# ...
# Create Pipeline 'myExamplePipeline'
# ...
(f, bl) = dai.DeviceBootloader.getFirstAvailableDevice()
bootloader = dai.DeviceBootloader(bl)
progress = lambda p : print(f'Flashing progress: {p*100:.1f}%')
bootloader.flash(progress, myExamplePipeline)
```

GPIO boot settings. Boot settings must be set as following: BOOT[4:0] : 01000 and GPIO58 (WAKEUP): 0



We’re always happy to help with code or other questions you might have.

3.5 Pipeline

Pipeline is a collection of *nodes* and links between them. This flow provides extensive flexibility that users get for their DepthAI device.

3.5.1 Pipeline first steps

To get DepthAI up and running, one has to define a pipeline, populate it with nodes, configure the nodes and link them together. After that, the pipeline can be loaded onto the *Device* and be started.

```
pipeline = depthai.Pipeline()

# Create nodes, configure them and link them together

# Upload the pipeline to the device
with depthai.Device(pipeline) as device:
    # Start the pipeline that is now on the device
    device.startPipeline()

    # Set input/output queues to configure device/host communication through the XLink..
    → .
```

3.5.2 Using multiple devices

If user has multiple DepthAI devices, each device can run a separate pipeline or the same pipeline ([demo here](#)). To use different pipeline for each device, you can create multiple pipelines and pass the desired pipeline to the desired device on initialization.

3.5.3 Specifying OpenVINO version

When using a NN blob that was not compiled with the latest OpenVINO (that DepthAI supports), you have to specify the OpenVINO version of the pipeline. The reason behind this is that OpenVINO doesn't provide version inside the blob.

```
pipeline = depthai.Pipeline()
# Set the correct version:
pipeline.setOpenVINOVersion(depthai.OpenVINO.Version.VERSION_2020_1)
```

3.5.4 How to place it

Python

```
pipeline = dai.Pipeline()
```

C++

```
dai::Pipeline pipeline;
```

3.5.5 Reference

Python

class `depthai.Pipeline`

Represents the pipeline, set of nodes and connections between them

createColorCamera (*self*: `depthai.Pipeline`) → *depthai.ColorCamera*

createImageManip (*self*: `depthai.Pipeline`) → *depthai.ImageManip*

createMobileNetDetectionNetwork (*self*: `depthai.Pipeline`) → *depthai.MobileNetDetectionNetwork*

createMobileNetSpatialDetectionNetwork (*self*: `depthai.Pipeline`) → *depthai.MobileNetSpatialDetectionNetwork*

createMonoCamera (*self*: `depthai.Pipeline`) → *depthai.MonoCamera*

createNeuralNetwork (*self*: `depthai.Pipeline`) → *depthai.NeuralNetwork*

createObjectTracker (*self*: `depthai.Pipeline`) → *depthai.ObjectTracker*

createSPIOut (*self*: `depthai.Pipeline`) → *depthai.SPIOut*

createSpatialLocationCalculator (*self*: `depthai.Pipeline`) → *depthai.SpatialLocationCalculator*

createStereoDepth (*self*: `depthai.Pipeline`) → *depthai.StereoDepth*

createSystemLogger (*self*: `depthai.Pipeline`) → *depthai.SystemLogger*

createVideoEncoder (*self*: `depthai.Pipeline`) → *depthai.VideoEncoder*

createXLinkIn (*self*: `depthai.Pipeline`) → *depthai.XLinkIn*

createXLinkOut (*self*: `depthai.Pipeline`) → *depthai.XLinkOut*

createYoloDetectionNetwork (*self*: `depthai.Pipeline`) → *depthai.YoloDetectionNetwork*

createYoloSpatialDetectionNetwork (*self*: `depthai.Pipeline`) → *depthai.YoloSpatialDetectionNetwork*

getAllAssets (*self*: `depthai.Pipeline`) → *depthai.AssetManager*

Get assets on the pipeline includes nodes assets

getAllNodes (**args*, ***kwargs*)

Overloaded function.

1. `getAllNodes(self: depthai.Pipeline) -> List[depthai.Node]`

Get a vector of all nodes

2. `getAllNodes(self: depthai.Pipeline) -> List[depthai.Node]`

Get a vector of all nodes

getAssetManager (**args*, ***kwargs*)

Overloaded function.

1. `getAssetManager(self: depthai.Pipeline) -> depthai.AssetManager`

Get pipelines AssetManager as reference

2. `getAssetManager(self: depthai.Pipeline) -> depthai.AssetManager`

Get pipelines AssetManager as reference

getConnectionMap (*self*: [depthai.Pipeline](#)) → Dict[int, Set[*depthai.Node.Connection*]]
Get a reference to internal connection representation

getConnections (*self*: [depthai.Pipeline](#)) → List[*depthai.Node.Connection*]
Get all connections

getGlobalProperties (*self*: [depthai.Pipeline](#)) → *depthai.GlobalProperties*
Returns Global properties of current pipeline

getNode (**args*, ***kwargs*)
Overloaded function.

1. **getNode**(*self*: [depthai.Pipeline](#), *arg0*: int) -> *depthai.Node*
Get node with id if it exists, nullptr otherwise
2. **getNode**(*self*: [depthai.Pipeline](#), *arg0*: int) -> *depthai.Node*
Get node with id if it exists, nullptr otherwise

getNodeMap (*self*: [depthai.Pipeline](#)) → Dict[int, *depthai.Node*]
Get a reference to internal node map

link (*self*: [depthai.Pipeline](#), *arg0*: *depthai.Node.Output*, *arg1*: *depthai.Node.Input*) → None
Link output to an input. Both nodes must be on the same pipeline
Throws an error if they aren't or cannot be connected
Parameter out: Nodes output to connect from
Parameter in: Nodes input to connect to

remove (*self*: [depthai.Pipeline](#), *node*: [depthai.Node](#)) → None
Removes a node from pipeline

setOpenVINOVersion (*self*: [depthai.Pipeline](#), *version*: *depthai.OpenVINO.Version* = <*Version.VERSION_2021_3*: 6>) → None
Set a specific OpenVINO version to use with this pipeline

unlink (*self*: [depthai.Pipeline](#), *arg0*: *depthai.Node.Output*, *arg1*: *depthai.Node.Input*) → None
Unlink output from an input.
Throws an error if link doesn't exists
Parameter out: Nodes output to unlink from
Parameter in: Nodes input to unlink to

C++

class *dai::Pipeline*
Represents the pipeline, set of nodes and connections between them.

Public Types

```
using NodeConnectionMap = PipelineImpl::NodeConnectionMap
using NodeMap = PipelineImpl::NodeMap
```

Public Functions

```
Pipeline ()
    Constructs a new pipeline

Pipeline (const std::shared_ptr<PipelineImpl> &pimpl)

GlobalProperties getGlobalProperties () const
    Return Global properties of current pipeline

PipelineSchema getPipelineSchema ()
    Return Pipeline schema

void serialize (PipelineSchema &schema, Assets &assets, std::vector<std::uint8_t> &assetStorage,
               OpenVINO::Version &version) const

template<class N>
std::shared_ptr<N> create ()
    Adds a node to pipeline.
    Node is specified by template argument N

void remove (std::shared_ptr<Node> node)
    Removes a node from pipeline.

std::vector<std::shared_ptr<const Node>> getAllNodes () const
    Get a vector of all nodes.

std::vector<std::shared_ptr<Node>> getAllNodes ()
    Get a vector of all nodes.

std::shared_ptr<const Node> getNode (Node::Id id) const
    Get node with id if it exists, nullptr otherwise.

std::shared_ptr<Node> getNode (Node::Id id)
    Get node with id if it exists, nullptr otherwise.

std::vector<Node::Connection> getConnections () const
    Get all connections.

const NodeConnectionMap &getConnectionMap () const
    Get a reference to internal connection representation.

const NodeMap &getNodeMap () const
    Get a reference to internal node map.

void link (const Node::Output &out, const Node::Input &in)
    Link output to an input. Both nodes must be on the same pipeline
    Throws an error if they aren't or cannot be connected
```

Parameters

- out: Nodes output to connect from
- in: Nodes input to connect to

void **unlink** (const *Node*::Output &out, const *Node*::Input &in)

Unlink output from an input.

Throws an error if link doesn't exists

Parameters

- out: Nodes output to unlink from
- in: Nodes input to unlink to

AssetManager **getAllAssets** () const

Get assets on the pipeline includes nodes assets.

const *AssetManager* &**getAssetManager** () const

Get pipelines *AssetManager* as reference.

AssetManager &**getAssetManager** ()

Get pipelines *AssetManager* as reference.

void **setOpenVINOVersion** (*OpenVINO*::Version version)

Set a specific *OpenVINO* version to use with this pipeline.

Public Static Attributes

constexpr auto **DEFAULT_OPENVINO_VERSION** = *PipelineImpl*::DEFAULT_OPENVINO_VERSION

Default *Pipeline* openvino version.

Private Functions

PipelineImpl ***impl** ()

const *PipelineImpl* ***impl** () const

Private Members

std::shared_ptr<*PipelineImpl*> **pimpl**

We're always happy to help with code or other questions you might have.

3.6 Nodes

Nodes are the building blocks when populating the *Pipeline*. Each node provides a specific functionality on the DepthAI, a set of configurable properties and inputs/outputs. After you create a node on a pipeline, you can also configure it as desired and link it to other nodes.

Outputs and inputs

Each node can have zero, one or multiple inputs and outputs. For example *SystemLogger* node has no inputs and 1 output and *StereoDepth* has 2 inputs and 6 outputs.

Node input

Node input queue is a queue for the *Messages*. It can be linked with other node's output (that's how you link up nodes). Node inputs are configurable - with `input.setBlocking(bool)` and `input.setQueueSize(num)`. Default behaviour is blocking and a queue size of 8 messages. If the input queue fills up, behavior of the input depends on blocking attribute. If blocking is enabled, new messages will be discarded until user gets the old messages. If blocking is disabled, new messages will push out old messages.

Node output

Node outputs *Messages*. There is no output queue per se, but some nodes do have a configurable output message pool. Output message pool is a reserved memory region (to reduce memory fragmentation) that holds output messages. After the node creates an output message (for example *ImgFrame*), it will send it to other nodes as specified when linking the inputs/outputs of the node. Currently, some nodes (*VideoEncoder*, *NeuralNetwork*, *ImageManip*, *XLinkIn*) can have the pool size configured. The size of the pool specifies how many messages can be created and sent out while other messages are already somewhere in the pipeline. When all the messages from pool are sent out and none yet returned, that is when the node will block and wait until a message is returned to the pool (not used by any node in the pipeline anymore)

On the table of contents (left side of the page) all nodes are listed under the Node entry. You can click on them to find out more.

3.6.1 ColorCamera

ColorCamera node is a source of *image frames*. You can control in at runtime with the `InputControl` and `InputConfig`.

How to place it

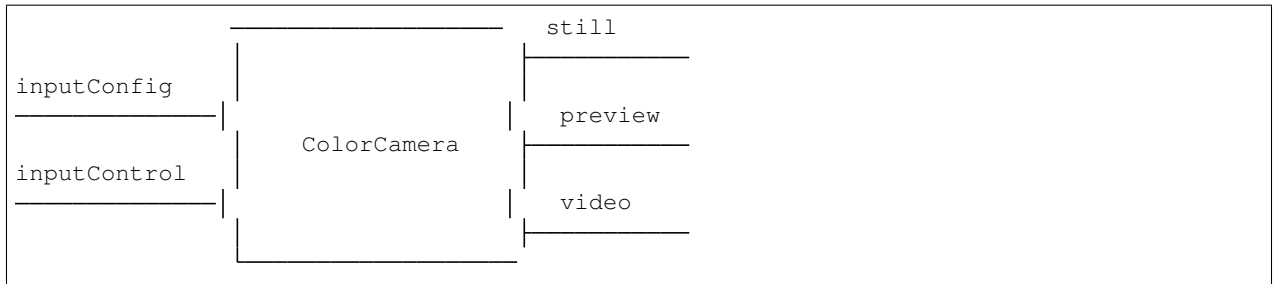
Python

```
pipeline = dai.Pipeline()
cam = pipeline.createColorCamera()
```

C++

```
dai::Pipeline pipeline;
auto cam = pipeline.create<dai::node::ColorCamera>();
```

Inputs and Outputs



Message types

- inputConfig - *ImageManipConfig*
- inputControl - *CameraControl*
- still - *ImgFrame*
- preview - *ImgFrame*
- video - *ImgFrame*

Preview is RGB (or BGR planar/interleaved if configured) and is mostly suited for small size previews and to feed the image into *NeuralNetwork*. video and still are both NV12, so are suitable for bigger sizes. still image gets created when a capture event is sent to the ColorCamera, so it's like taking a photo.

Usage

Python

```

pipeline = dai.Pipeline()
cam = pipeline.createColorCamera()
cam.setPreviewSize(300, 300)
cam.setBoardSocket(dai.CameraBoardSocket.RGB)
cam.setResolution(dai.ColorCameraProperties.SensorResolution.THE_1080_P)
cam.setInterleaved(False)
cam.setColorOrder(dai.ColorCameraProperties.ColorOrder.RGB)

```

C++

```

dai::Pipeline pipeline;
auto cam = pipeline.create<dai::node::createColorCamera>();
cam->setPreviewSize(300, 300);
cam->setBoardSocket(dai::CameraBoardSocket::RGB);
cam->setResolution(dai::ColorCameraProperties::SensorResolution::THE_1080_P);
cam->setInterleaved(false);
cam->setColorOrder(dai::ColorCameraProperties::ColorOrder::RGB);

```


Examples of functionality

- *01 - RGB Preview*
- *14.1 - Color Camera Control*
- *28 - Camera video high resolution*

Reference

Python

```
class depthai.ColorCamera
    ColorCamera node. For use with color sensors.

    class Connection
        Connection between an Input and Output

    class Id
        Node identificator. Unique for every node on a single Pipeline

    Properties
        alias of depthai.ColorCameraProperties

    getAssets (self: depthai.Node) → List[depthai.Asset]
        Retrieves all nodes assets

    getBoardSocket (self: depthai.ColorCamera) → dai::CameraBoardSocket
        Retrieves which board socket to use

        Returns Board socket to use

    getCamId (self: depthai.ColorCamera) → int

    getColorOrder (self: depthai.ColorCamera) → dai::ColorCameraProperties::ColorOrder
        Get color order of preview output frames. RGB or BGR

    getFp16 (self: depthai.ColorCamera) → bool
        Get fp16 (0..255) data of preview output frames

    getFps (self: depthai.ColorCamera) → float
        Get rate at which camera should produce frames

        Returns Rate in frames per second

    getImageOrientation (self: depthai.ColorCamera) → dai::CameraImageOrientation
        Get camera image orientation

    getInputs (self: depthai.Node) → List[dai::Node::Input]
        Retrieves all nodes inputs

    getInterleaved (self: depthai.ColorCamera) → bool
        Get planar or interleaved data of preview output frames

    getName (self: depthai.Node) → str
        Retrieves nodes name

    getOutputs (self: depthai.Node) → List[dai::Node::Output]
        Retrieves all nodes outputs

    getPreviewHeight (self: depthai.ColorCamera) → int
        Get preview height
```

getPreviewKeepAspectRatio (*self*: depthai.ColorCamera) → bool

See also:

setPreviewKeepAspectRatio

Returns Preview keep aspect ratio option

getPreviewSize (*self*: depthai.ColorCamera) → Tuple[int, int]

Get preview size as tuple

getPreviewWidth (*self*: depthai.ColorCamera) → int

Get preview width

getResolution (*self*: depthai.ColorCamera) → dai::ColorCameraProperties::SensorResolution

Get sensor resolution

getResolutionHeight (*self*: depthai.ColorCamera) → int

Get sensor resolution height

getResolutionSize (*self*: depthai.ColorCamera) → Tuple[int, int]

Get sensor resolution as size

getResolutionWidth (*self*: depthai.ColorCamera) → int

Get sensor resolution width

getSensorCrop (*self*: depthai.ColorCamera) → Tuple[float, float]

Returns Sensor top left crop coordinates

getSensorCropX (*self*: depthai.ColorCamera) → float

Get sensor top left x crop coordinate

getSensorCropY (*self*: depthai.ColorCamera) → float

Get sensor top left y crop coordinate

getStillHeight (*self*: depthai.ColorCamera) → int

Get still height

getStillSize (*self*: depthai.ColorCamera) → Tuple[int, int]

Get still size as tuple

getStillWidth (*self*: depthai.ColorCamera) → int

Get still width

getVideoHeight (*self*: depthai.ColorCamera) → int

Get video height

getVideoSize (*self*: depthai.ColorCamera) → Tuple[int, int]

Get video size as tuple

getVideoWidth (*self*: depthai.ColorCamera) → int

Get video width

getWaitForConfigInput (*self*: depthai.ColorCamera) → bool

See also:

setWaitForConfigInput

Returns True if wait for inputConfig message, false otherwise

property id

Id of node

property initialControl

Initial control options to apply to sensor

property inputConfig

Input for ImageManipConfig message, which can modify crop paremeters in runtime

Default queue is non-blocking with size 8

property inputControl

Input for CameraControl message, which can modify camera parameters in runtime

Default queue is blocking with size 8

property preview

Outputs ImgFrame message that carries BGR/RGB planar/interleaved encoded frame data.

Suitable for use with NeuralNetwork node

sensorCenterCrop (*self: depthai.ColorCamera*) → *None*

Specify sensor center crop. Resolution size / video size

setBoardSocket (*self: depthai.ColorCamera, boardSocket: dai::CameraBoardSocket*) → *None*

Specify which board socket to use

Parameter boardSocket: Board socket to use**setCamId** (*self: depthai.ColorCamera, arg0: int*) → *None***setColorOrder** (*self: depthai.ColorCamera, colorOrder: dai::ColorCameraProperties::ColorOrder*) → *None*

Set color order of preview output images. RGB or BGR

setFp16 (*self: depthai.ColorCamera, fp16: bool*) → *None*

Set fp16 (0..255) data type of preview output frames

setFps (*self: depthai.ColorCamera, fps: float*) → *None*

Set rate at which camera should produce frames

Parameter fps: Rate in frames per second**setImageOrientation** (*self: depthai.ColorCamera, boardSocket: dai::CameraImageOrientation*) → *None*

Set camera image orientation

setInterleaved (*self: depthai.ColorCamera, interleaved: bool*) → *None*

Set planar or interleaved data of preview output frames

setPreviewKeepAspectRatio (*self: depthai.ColorCamera, keep: bool*) → *None*

Specifies whether preview output should preserve aspect ratio, after downscaling from video size or not.

Parameter keep: If true, a larger crop region will be considered to still be able to create the final image in the specified aspect ratio. Otherwise video size is resized to fit preview size**setPreviewSize** (*self: depthai.ColorCamera, width: int, height: int*) → *None*

Set preview output size

setResolution (*self: depthai.ColorCamera, resolution: dai::ColorCameraProperties::SensorResolution*) → *None*

Set sensor resolution

setSensorCrop (*self: depthai.ColorCamera, x: float, y: float*) → *None*

Specifies sensor crop rectangle

Parameter x: Top left X coordinate

Parameter y: Top left Y coordinate

setStillSize (*self*: [depthai.ColorCamera](#), *width*: *int*, *height*: *int*) → *None*
Set still output size

setVideoSize (*self*: [depthai.ColorCamera](#), *width*: *int*, *height*: *int*) → *None*
Set video output size

setWaitForConfigInput (*self*: [depthai.ColorCamera](#), *wait*: *bool*) → *None*
Specify to wait until inputConfig receives a configuration message, before sending out a frame.

Parameter wait: True to wait for inputConfig message, false otherwise

property still

Outputs ImgFrame message that carries NV12 encoded (YUV420, UV plane interleaved) frame data.

The message is sent only when a CameraControl message arrives to inputControl with captureStill command set.

property video

Outputs ImgFrame message that carries NV12 encoded (YUV420, UV plane interleaved) frame data.

Suitable for use with VideoEncoder node

C++

```
class dai::node::ColorCamera : public dai::Node  
ColorCamera node. For use with color sensors.
```

Public Types

```
using Properties = dai::ColorCameraProperties
```

Public Functions

ColorCamera (**const** std::shared_ptr<[PipelineImpl](#)> &par, int64_t nodeId)
Constructs *ColorCamera* node.

void **setBoardSocket** ([CameraBoardSocket](#) boardSocket)
Specify which board socket to use

Parameters

- boardSocket: Board socket to use

[CameraBoardSocket](#) **getBoardSocket** () **const**
Retrieves which board socket to use

Return Board socket to use

void **setCamId** (int64_t id)
Set which color camera to use.

int64_t **getCamId** () **const**
Get which color camera to use.

void **setImageOrientation** ([CameraImageOrientation](#) imageOrientation)
Set camera image orientation.

[CameraImageOrientation](#) **getImageOrientation** () **const**
Get camera image orientation.

void **setColorOrder** (*ColorCameraProperties::ColorOrder colorOrder*)
Set color order of preview output images. RGB or BGR.

ColorCameraProperties::ColorOrder **getColorOrder** () const
Get color order of preview output frames. RGB or BGR.

void **setInterleaved** (bool *interleaved*)
Set planar or interleaved data of preview output frames.

bool **getInterleaved** () const
Get planar or interleaved data of preview output frames.

void **setFp16** (bool *fp16*)
Set fp16 (0..255) data type of preview output frames.

bool **getFp16** () const
Get fp16 (0..255) data of preview output frames.

void **setPreviewSize** (int *width*, int *height*)
Set preview output size.

void **setVideoSize** (int *width*, int *height*)
Set video output size.

void **setStillSize** (int *width*, int *height*)
Set still output size.

void **setResolution** (*Properties::SensorResolution resolution*)
Set sensor resolution.

Properties::SensorResolution **getResolution** () const
Get sensor resolution.

void **setFps** (float *fps*)
Set rate at which camera should produce frames

Parameters

- *fps*: Rate in frames per second

float **getFps** () const
Get rate at which camera should produce frames

Return Rate in frames per second

std::tuple<int, int> **getPreviewSize** () const
Get preview size as tuple.

int **getPreviewWidth** () const
Get preview width.

int **getPreviewHeight** () const
Get preview height.

std::tuple<int, int> **getVideoSize** () const
Get video size as tuple.

int **getVideoWidth** () const
Get video width.

int **getVideoHeight** () const
Get video height.

std::tuple<int, int> **getStillSize** () const
Get still size as tuple.

int **getStillWidth** () const
Get still width.

int **getStillHeight** () const
Get still height.

std::tuple<int, int> **getResolutionSize** () const
Get sensor resolution as size.

int **getResolutionWidth** () const
Get sensor resolution width.

int **getResolutionHeight** () const
Get sensor resolution height.

void **sensorCenterCrop** ()
Specify sensor center crop. Resolution size / video size

void **setSensorCrop** (float x, float y)
Specifies sensor crop rectangle

Parameters

- x: Top left X coordinate
- y: Top left Y coordinate

std::tuple<float, float> **getSensorCrop** () const

Return Sensor top left crop coordinates

float **getSensorCropX** () const
Get sensor top left x crop coordinate.

float **getSensorCropY** () const
Get sensor top left y crop coordinate.

void **setWaitForConfigInput** (bool wait)
Specify to wait until inputConfig receives a configuration message, before sending out a frame.

Parameters

- wait: True to wait for inputConfig message, false otherwise

bool **getWaitForConfigInput** ()

See [*setWaitForConfigInput*](#)

Return True if wait for inputConfig message, false otherwise

void **setPreviewKeepAspectRatio** (bool keep)
Specifies whether preview output should preserve aspect ratio, after downscaling from video size or not.

Parameters

- keep: If true, a larger crop region will be considered to still be able to create the final image in the specified aspect ratio. Otherwise video size is resized to fit preview size

bool **getPreviewKeepAspectRatio** ()

See [*setPreviewKeepAspectRatio*](#)

Return Preview keep aspect ratio option

Public Members

CameraControl **initialControl**

Initial control options to apply to sensor

Input **inputConfig** = { *this, "inputConfig", Input::Type::SReceiver, false, 8, { { *DatatypeEnum::ImageManipConfig*, false } } }

Input for *ImageManipConfig* message, which can modify crop paremeters in runtime

Default queue is non-blocking with size 8

Input **inputControl** = { *this, "inputControl", Input::Type::SReceiver, true, 8, { { *DatatypeEnum::CameraControl*, false } } }

Input for *CameraControl* message, which can modify camera parameters in runtime

Default queue is blocking with size 8

Output **video** = { *this, "video", Output::Type::MSender, { { *DatatypeEnum::ImgFrame*, false } } }

Outputs *ImgFrame* message that carries NV12 encoded (YUV420, UV plane interleaved) frame data.

Suitable for use with *VideoEncoder* node

Output **preview** = { *this, "preview", Output::Type::MSender, { { *DatatypeEnum::ImgFrame*, false } } }

Outputs *ImgFrame* message that carries BGR/RGB planar/interleaved encoded frame data.

Suitable for use with *NeuralNetwork* node

Output **still** = { *this, "still", Output::Type::MSender, { { *DatatypeEnum::ImgFrame*, false } } }

Outputs *ImgFrame* message that carries NV12 encoded (YUV420, UV plane interleaved) frame data.

The message is sent only when a *CameraControl* message arrives to inputControl with captureStill command set.

Private Functions

std::string **getName** () **const override**

Retrieves nodes name.

std::vector<Output> **getOutputs** () **override**

Retrieves all nodes outputs.

std::vector<Input> **getInputs** () **override**

Retrieves all nodes inputs.

nlohmann::json **getProperties** () **override**

std::shared_ptr<*Node*> **clone** () **override**

Private Members

Properties **properties**

std::shared_ptr<*RawCameraControl*> **rawControl**

We're always happy to help with code or other questions you might have.

3.6.2 ImageManip

ImageManip node can be used to crop, rotate rectangle area or perform various image transforms: rotate, mirror, flip, perspective transform.

How to place it

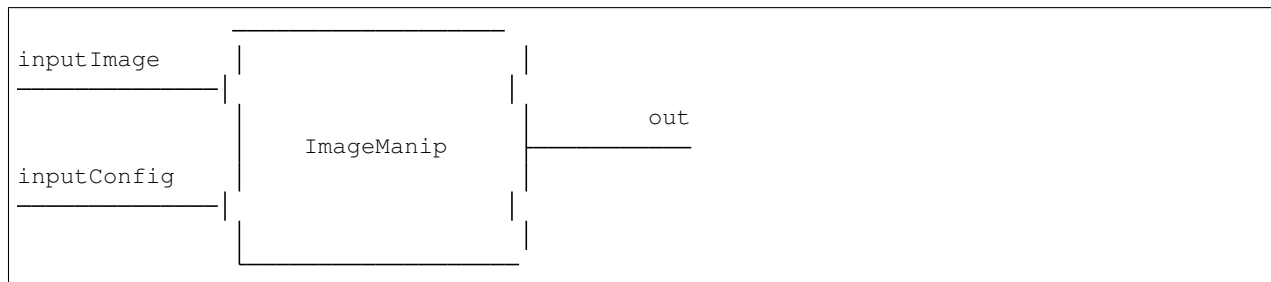
Python

```
pipeline = dai.Pipeline()
manip = pipeline.createImageManip()
```

C++

```
dai::Pipeline pipeline;
auto manip = pipeline.create<dai::node::ImageManip>();
```

Inputs and Outputs



Message types

- inputImage - *ImgFrame*
- inputConfig - *ImageManipConfig*
- out - *ImgFrame*

Usage

Python

```
pipeline = dai.Pipeline()
manip = pipeline.createImageManip()

manip.initialConfig.setResize(300, 300)
manip.initialConfig.setFrameType(dai.ImgFrame.Type.BGR888p)
```

C++

```
dai::Pipeline pipeline;
auto manip = pipeline.create<dai::node::ImageManip>();

manip->initialConfig.setResize(300, 300);
manip->initialConfig.setFrameType(dai::ImgFrame::Type::BGR888p);
```


Examples of functionality

- *09 - Mono & MobilenetSSD*
- *11 - RGB & Encoding & Mono & MobilenetSSD*
- *14.1 - Color Camera Control*

Reference

Python

class `depthai.ImageManip`

ImageManip node. Capability to crop, resize, warp, ... incoming image frames

class `Connection`

Connection between an Input and Output

class `Id`

Node identifier. Unique for every node on a single Pipeline

getAssets (*self*: `depthai.Node`) → `List[depthai.Asset]`

Retrieves all nodes assets

getInputs (*self*: `depthai.Node`) → `List[dai::Node::Input]`

Retrieves all nodes inputs

getName (*self*: `depthai.Node`) → `str`

Retrieves nodes name

getOutputs (*self*: `depthai.Node`) → `List[dai::Node::Output]`

Retrieves all nodes outputs

property `id`

Id of node

property `initialConfig`

Initial config to use when manipulating frames

property `inputConfig`

Input ImageManipConfig message with ability to modify parameters in runtime Default queue is blocking with size 8

property `inputImage`

Input image to be modified Default queue is blocking with size 8

property `out`

Outputs ImgFrame message that carries modified image.

setCenterCrop (*self*: `depthai.ImageManip`, *arg0*: `float`, *arg1*: `float`) → `None`

setCropRect (*self*: `depthai.ImageManip`, *arg0*: `float`, *arg1*: `float`, *arg2*: `float`, *arg3*: `float`) → `None`

setFrameType (*self*: `depthai.ImageManip`, *arg0*: `dai::RawImgFrame::Type`) → `None`

setHorizontalFlip (*self*: `depthai.ImageManip`, *arg0*: `bool`) → `None`

setKeepAspectRatio (*self*: `depthai.ImageManip`, *arg0*: `bool`) → `None`

setMaxOutputFrameSize (*self*: `depthai.ImageManip`, *arg0*: `int`) → `None`

Specify maximum size of output image.

Parameter maxFrameSize: Maximum frame size in bytes

setNumFramesPool (*self*: depthai.ImageManip, *arg0*: int) → None

Specify number of frames in pool.

Parameter numFramesPool: How many frames should the pool have

setResize (*self*: depthai.ImageManip, *arg0*: int, *arg1*: int) → None

setResizeThumbnail (*self*: depthai.ImageManip, *arg0*: int, *arg1*: int, *arg2*: int, *arg3*: int, *arg4*: int) → None

setWaitForConfigInput (*self*: depthai.ImageManip, *arg0*: bool) → None

Specify whether or not wait until configuration message arrives to inputConfig Input.

Parameter wait: True to wait for configuration message, false otherwise

C++

class *dai::node::ImageManip*: **public** *dai::Node*

ImageManip node. Capability to crop, resize, warp, ... incoming image frames.

Public Types

using *Properties* = *dai::ImageManipProperties*

Public Functions

ImageManip (**const** std::shared_ptr<*PipelineImpl*> &*par*, int64_t *nodeId*)

void **setCropRect** (float *xmin*, float *ymin*, float *xmax*, float *ymax*)

void **setCenterCrop** (float *ratio*, float *whRatio* = 1.0f)

void **setResize** (int *w*, int *h*)

void **setResizeThumbnail** (int *w*, int *h*, int *bgRed* = 0, int *bgGreen* = 0, int *bgBlue* = 0)

void **setFrameType** (*dai::RawImgFrame::Type* *name*)

void **setHorizontalFlip** (bool *flip*)

void **setKeepAspectRatio** (bool *keep*)

void **setWaitForConfigInput** (bool *wait*)

Specify whether or not wait until configuration message arrives to inputConfig Input.

Parameters

- *wait*: True to wait for configuration message, false otherwise

void **setNumFramesPool** (int *numFramesPool*)

Specify number of frames in pool.

Parameters

- *numFramesPool*: How many frames should the pool have

void **setMaxOutputFrameSize** (int *maxFrameSize*)

Specify maximum size of output image.

Parameters

- *maxFrameSize*: Maximum frame size in bytes

Public Members

ImageManipConfig **initialConfig**

Initial config to use when manipulating frames

Input **inputConfig** = { *this, "inputConfig", Input::Type::SReceiver, true, 8, { { *DatatypeEnum::ImageManipConfig*, true } } }

Input *ImageManipConfig* message with ability to modify parameters in runtime Default queue is blocking with size 8

Input **inputImage** = { *this, "inputImage", Input::Type::SReceiver, true, 8, { { *DatatypeEnum::ImgFrame*, true } } }

Input image to be modified Default queue is blocking with size 8

Output **out** = { *this, "out", Output::Type::MSender, { { *DatatypeEnum::ImgFrame*, true } } }

Outputs *ImgFrame* message that carries modified image.

Private Functions

std::string **getName** () **const override**

Retrieves nodes name.

std::vector<Input> **getInputs** () **override**

Retrieves all nodes inputs.

std::vector<Output> **getOutputs** () **override**

Retrieves all nodes outputs.

nlohmann::json **getProperties** () **override**

std::shared_ptr<*Node*> **clone** () **override**

Private Members

Properties **properties**

std::shared_ptr<*RawImageManipConfig*> **rawConfig**

We're always happy to help with code or other questions you might have.

3.6.3 MobileNetDetectionNetwork

MobileNet detection network node is very similar to *NeuralNetwork* (in fact it extends it). The only difference is that this node is specifically for the MobileNet NN and it decodes the result of the NN on device. This means that out of this node is not a byte array but a *ImgDetections* that can easily be used in your code.

How to place it

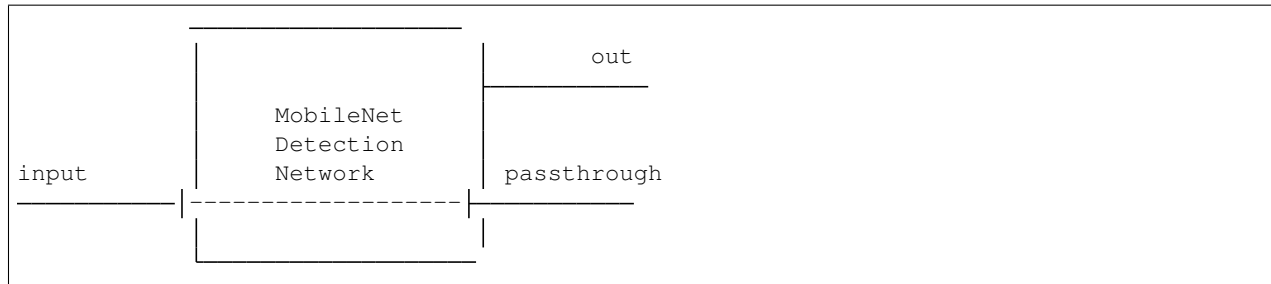
Python

```
pipeline = dai.Pipeline()
mobilenetDet = pipeline.createMobileNetDetectionNetwork()
```

C++

```
dai::Pipeline pipeline;
auto mobilenetDet = pipeline.create<dai::node::MobileNetDetectionNetwork>();
```

Inputs and Outputs



Message types

- input - *ImgFrame*
- out - *ImgDetections*
- passthrough - *ImgFrame*

Usage

Python

```
pipeline = dai.Pipeline()
mobilenetDet = pipeline.createMobileNetDetectionNetwork()

mobilenetDet.setConfidenceThreshold(0.5)
mobilenetDet.setBlobPath(nnBlobPath)
mobilenetDet.setNumInferenceThreads(2)
mobilenetDet.input.setBlocking(False)
```

C++

```
dai::Pipeline pipeline;
auto mobilenetDet = pipeline.create<dai::node::MobileNetDetectionNetwork>();

mobilenetDet->setConfidenceThreshold(0.5f);
mobilenetDet->setBlobPath(nnBlobPath);
mobilenetDet->setNumInferenceThreads(2);
mobilenetDet->input.setBlocking(false);
```

Examples of functionality

- 08 - RGB & MobilenetSSD
- 09 - Mono & MobilenetSSD
- 10 - Mono & MobilenetSSD & Depth

Reference

Python

```
class depthai.MobileNetDetectionNetwork
    MobileNetDetectionNetwork node. Parses MobileNet results

    class Connection
        Connection between an Input and Output

    class Id
        Node identificator. Unique for every node on a single Pipeline

    Properties
        alias of depthai.DetectionNetworkProperties

    getAssets (self: depthai.Node) → List[depthai.Asset]
        Retrieves all nodes assets

    getInputs (self: depthai.Node) → List[dai::Node::Input]
        Retrieves all nodes inputs

    getName (self: depthai.Node) → str
        Retrieves nodes name

    getNumInferenceThreads (self: depthai.NeuralNetwork) → int
        How many inference threads will be used to run the network

        Returns Number of threads, 0, 1 or 2. Zero means AUTO

    getOutputs (self: depthai.Node) → List[dai::Node::Output]
        Retrieves all nodes outputs

    property id
        Id of node

    property input
        Input message with data to be infered upon Default queue is blocking with size 5

    property out
        Outputs ImgDetections message that carries parsed detection results.

    property passthrough
        Passthrough message on which the inference was performed.

        Suitable for when input queue is set to non-blocking behavior.

    setBlobPath (self: depthai.NeuralNetwork, path: str) → None
        Load network blob into assets and use once pipeline is started.

        Throws if file doesn't exist or isn't a valid network blob.

        Parameter path: Path to network blob

    setConfidenceThreshold (self: depthai.DetectionNetwork, thresh: float) → None
        Specifies confidence threshold at which to filter the rest of the detections.

        Parameter thresh: Detection confidence must be greater than specified threshold to be added to the list

    setNumInferenceThreads (self: depthai.NeuralNetwork, numThreads: int) → None
        How many threads should the node use to run the network.

        Parameter numThreads: Number of threads to dedicate to this node
```

setNumNCEPerInferenceThread (*self*: [depthai.NeuralNetwork](#), *numNCEPerThread*: *int*) → *None*

How many Neural Compute Engines should a single thread use for inference

Parameter numNCEPerThread: Number of NCE per thread

setNumPoolFrames (*self*: [depthai.NeuralNetwork](#), *numFrames*: *int*) → *None*

Specifies how many frames will be available in the pool

Parameter numFrames: How many frames will pool have

C++

```
class dai::node::MobileNetDetectionNetwork : public dai::node::DetectionNetwork
    MobileNetDetectionNetwork node. Parses MobileNet results.
```

Public Functions

MobileNetDetectionNetwork (**const** std::shared_ptr<[PipelineImpl](#)> &par, int64_t nodeId)

We're always happy to help with code or other questions you might have.

3.6.4 MobileNetSpatialDetectionNetwork

Spatial detection for the MobileNet NN. It is similar to a combination of the [MobileNetDetectionNetwork](#) and [Spatial-LocationCalculator](#).

How to place it

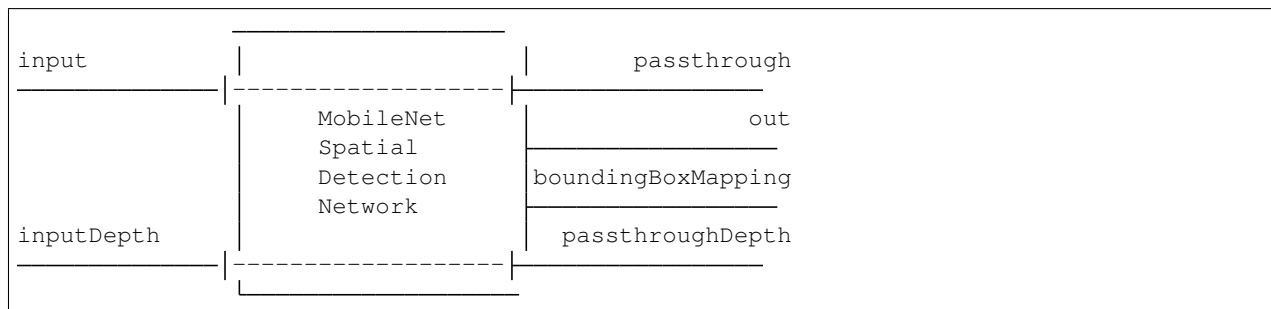
Python

```
pipeline = dai.Pipeline()
mobilenetSpatial = pipeline.createMobileNetSpatialDetectionNetwork()
```

C++

```
dai::Pipeline pipeline;
auto mobilenetSpatial = pipeline.create<dai::node::MobileNetSpatialDetectionNetwork>
    ();
```

Inputs and Outputs



Message types

- input - [ImgFrame](#)

- inputDepth - *ImgFrame*
- passthrough - *ImgFrame*
- out - *SpatialImgDetections*
- boundingBoxMapping - *SpatialLocationCalculatorConfig*
- passthroughDepth - *ImgFrame*

Usage

Python

```
pipeline = dai.Pipeline()
mobilenetSpatial = pipeline.createMobileNetSpatialDetectionNetwork()

mobilenetSpatial.setBlobPath(nnBlobPath)
# Will ignore all detections whose confidence is below 50%
mobilenetSpatial.setConfidenceThreshold(0.5)
mobilenetSpatial.input.setBlocking(False)
# How big the ROI will be (smaller value can provide a more stable reading)
mobilenetSpatial.setBoundingBoxScaleFactor(0.5)
# Min/Max threshold. Values out of range will be set to 0 (invalid)
mobilenetSpatial.setDepthLowerThreshold(100)
mobilenetSpatial.setDepthUpperThreshold(5000)

# Link depth from the StereoDepth node
stereo.depth.link(mobilenetSpatial.inputDepth)
```

C++

```
dai::Pipeline pipeline;
auto mobilenetSpatial = pipeline.create<dai::node::MobileNetSpatialDetectionNetwork>
    <()>;

mobilenetSpatial->setBlobPath(nnBlobPath);
// Will ignore all detections whose confidence is below 50%
mobilenetSpatial->setConfidenceThreshold(0.5f);
mobilenetSpatial->input.setBlocking(false);
// How big the ROI will be (smaller value can provide a more stable reading)
mobilenetSpatial->setBoundingBoxScaleFactor(0.5f);
// Min/Max threshold. Values out of range will be set to 0 (invalid)
mobilenetSpatial->setDepthLowerThreshold(100);
mobilenetSpatial->setDepthUpperThreshold(5000);

// Link depth from the StereoDepth node
stereo->depth.link(mobilenetSpatial->inputDepth);
```

Examples of functionality

- 26.1 - RGB & MobilenetSSD with spatial data
- 26.2 - Mono & MobilenetSSD with spatial data

Reference

Python

```
class depthai.MobileNetSpatialDetectionNetwork
    MobileNetSpatialDetectionNetwork. Mobilenet-SSD based network with spatial location data.

    class Connection
        Connection between an Input and Output

    class Id
        Node identificator. Unique for every node on a single Pipeline

    Properties
        alias of depthai.SpatialDetectionNetworkProperties

    property boundingBoxMapping
        Outputs mapping of detected bounding boxes relative to depth map

        Suitable for when displaying remapped bounding boxes on depth frame

    getAssets (self: depthai.Node) → List[depthai.Asset]
        Retrieves all nodes assets

    getInputs (self: depthai.Node) → List[dai::Node::Input]
        Retrieves all nodes inputs

    getName (self: depthai.Node) → str
        Retrieves nodes name

    getNumInferenceThreads (self: depthai.NeuralNetwork) → int
        How many inference threads will be used to run the network

        Returns Number of threads, 0, 1 or 2. Zero means AUTO

    getOutputs (self: depthai.Node) → List[dai::Node::Output]
        Retrieves all nodes outputs

    property id
        Id of node

    property input
        Input message with data to be infered upon Default queue is blocking with size 5

    property inputDepth
        Input message with depth data used to retrieve spatial information about detected object Default queue is
        non-blocking with size 4

    property out
        Outputs ImgDetections message that carries parsed detection results.

    property passthrough
        Passthrough message on which the inference was performed.

        Suitable for when input queue is set to non-blocking behavior.
```


property passthroughDepth

Passthrough message for depth frame on which the spatial location calculation was performed.

Suitable for when input queue is set to non-blocking behavior.

setBlobPath (*self*: [depthai.NeuralNetwork](#), *path*: *str*) → [None](#)

Load network blob into assets and use once pipeline is started.

Throws if file doesn't exist or isn't a valid network blob.

Parameter path: Path to network blob

setBoundingBoxScaleFactor (*self*: [depthai.SpatialDetectionNetwork](#), *scaleFactor*: *float*) → [None](#)

Specifies scale factor for detected bounding boxes.

Parameter scaleFactor: Scale factor must be in the interval (0,1].

setConfidenceThreshold (*self*: [depthai.DetectionNetwork](#), *thresh*: *float*) → [None](#)

Specifies confidence threshold at which to filter the rest of the detections.

Parameter thresh: Detection confidence must be greater than specified threshold to be added to the list

setDepthLowerThreshold (*self*: [depthai.SpatialDetectionNetwork](#), *lowerThreshold*: *int*) → [None](#)

Specifies lower threshold in millimeters for depth values which will used to calculate spatial data

Parameter lowerThreshold: LowerThreshold must be in the interval [0,upperThreshold] and less than upperThreshold.

setDepthUpperThreshold (*self*: [depthai.SpatialDetectionNetwork](#), *upperThreshold*: *int*) → [None](#)

Specifies upper threshold in millimeters for depth values which will used to calculate spatial data

Parameter upperThreshold: UpperThreshold must be in the interval (lowerThreshold,65535].

setNumInferenceThreads (*self*: [depthai.NeuralNetwork](#), *numThreads*: *int*) → [None](#)

How many threads should the node use to run the network.

Parameter numThreads: Number of threads to dedicate to this node

setNumNCEPerInferenceThread (*self*: [depthai.NeuralNetwork](#), *numNCEPerThread*: *int*) → [None](#)

How many Neural Compute Engines should a single thread use for inference

Parameter numNCEPerThread: Number of NCE per thread

setNumPoolFrames (*self*: [depthai.NeuralNetwork](#), *numFrames*: *int*) → [None](#)

Specifies how many frames will be available in the pool

Parameter numFrames: How many frames will pool have

C++

class *dai::node::MobileNetSpatialDetectionNetwork* : **public** *dai::node::SpatialDetectionNetwork*
MobileNetSpatialDetectionNetwork. Mobilenet-SSD based network with spatial location data.

Public Functions

MobileNetSpatialDetectionNetwork(const std::shared_ptr<PipelineImpl> &par, int64_t nodeId)

We're always happy to help with code or other questions you might have.

3.6.5 MonoCamera

MonoCamera node is a source of *image frames*. You can control in at runtime with the `inputControl`. Some DepthAI modules don't have mono camera(s). Two mono cameras are used to calculate stereo depth (with *StereoDepth* node).

How to place it

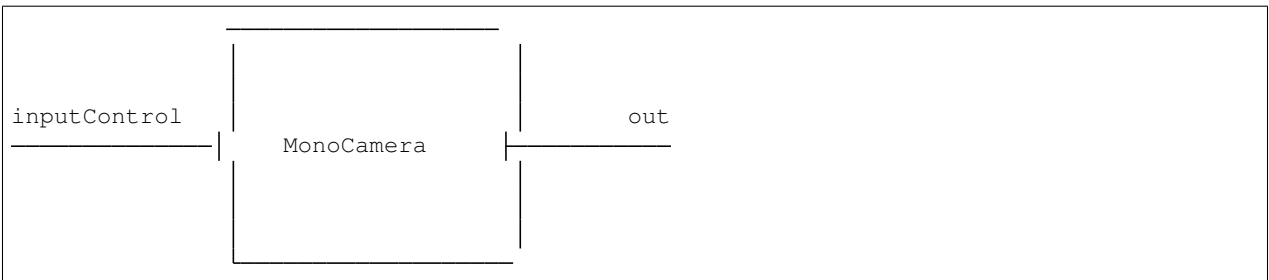
Python

```
pipeline = dai.Pipeline()
mono = pipeline.createMonoCamera()
```

C++

```
dai::Pipeline pipeline;
auto mono = pipeline.create<dai::node::MonoCamera>();
```

Inputs and Outputs



Message types

- `inputControl` - *CameraControl*
- `out` - *ImgFrame*

Usage

Python

```
pipeline = dai.Pipeline()
mono = pipeline.createMonoCamera()
mono.setBoardSocket(dai.CameraBoardSocket.RIGHT)
mono.setResolution(dai.MonoCameraProperties.SensorResolution.THE_720_P)
```

C++

```
dai::Pipeline pipeline;
auto mono = pipeline.create<dai::node::MonoCamera>();
mono->setBoardSocket(dai::CameraBoardSocket::RIGHT);
mono->setResolution(dai::MonoCameraProperties::SensorResolution::THE_720_P);
```

Examples of functionality

- [02 - Mono Preview](#)
- [09 - Mono & MobilenetSSD](#)
- [19 - Mono Camera Control](#)

Reference

Python

class depthai.MonoCamera

MonoCamera node. For use with grayscale sensors.

class Connection

Connection between an Input and Output

class Id

Node identificator. Unique for every node on a single Pipeline

Properties

alias of *depthai.MonoCameraProperties*

getAssets (*self*: depthai.Node) → List[depthai.Asset]

Retrieves all nodes assets

getBoardSocket (*self*: depthai.MonoCamera) → dai::CameraBoardSocket

Retrieves which board socket to use

Returns Board socket to use

getCamId (*self*: depthai.MonoCamera) → int

getFps (*self*: depthai.MonoCamera) → float

Get rate at which camera should produce frames

Returns Rate in frames per second

getImageOrientation (*self*: depthai.MonoCamera) → dai::CameraImageOrientation

Get camera image orientation

getInputs (*self*: depthai.Node) → List[dai::Node::Input]

Retrieves all nodes inputs

getName (*self*: depthai.Node) → str

Retrieves nodes name

getOutputs (*self*: depthai.Node) → List[dai::Node::Output]

Retrieves all nodes outputs

getResolution (*self*: depthai.MonoCamera) → dai::MonoCameraProperties::SensorResolution

Get sensor resolution

getResolutionHeight (*self*: depthai.MonoCamera) → int

Get sensor resolution height

getResolutionSize (*self*: [depthai.MonoCamera](#)) → Tuple[int, int]

Get sensor resolution as size

getResolutionWidth (*self*: [depthai.MonoCamera](#)) → int

Get sensor resolution width

property id

Id of node

property initialControl

Initial control options to apply to sensor

property inputControl

Input for CameraControl message, which can modify camera parameters in runtime Default queue is blocking with size 8

property out

Outputs ImgFrame message that carries RAW8 encoded (grayscale) frame data.

Suitable for use StereoDepth node

setBoardSocket (*self*: [depthai.MonoCamera](#), *boardSocket*: [dai::CameraBoardSocket](#)) → None

Specify which board socket to use

Parameter boardSocket: Board socket to use

setCamId (*self*: [depthai.MonoCamera](#), *arg0*: int) → None

setFps (*self*: [depthai.MonoCamera](#), *fps*: float) → None

Set rate at which camera should produce frames

Parameter fps: Rate in frames per second

setImageOrientation (*self*: [depthai.MonoCamera](#), *imageOrientation*: [dai::CameraImageOrientation](#)) → None

Set camera image orientation

setResolution (*self*: [depthai.MonoCamera](#), *resolution*: [dai::MonoCameraProperties::SensorResolution](#)) → None

Set sensor resolution

C++

class [dai::node::MonoCamera](#) : **public** [dai::Node](#)
[MonoCamera](#) node. For use with grayscale sensors.

Public Types

using Properties = [dai::MonoCameraProperties](#)

Public Functions

MonoCamera (**const** std::shared_ptr<[PipelineImpl](#)> &par, int64_t nodeId)

void **setBoardSocket** ([CameraBoardSocket](#) boardSocket)

Specify which board socket to use

Parameters

- boardSocket: Board socket to use

[CameraBoardSocket](#) **getBoardSocket** () **const**

Retrieves which board socket to use

Return Board socket to use

void **setCamId** (int64_t *id*)

int64_t **getCamId** () **const**

void **setImageOrientation** (*CameraImageOrientation* *imageOrientation*)
Set camera image orientation.

CameraImageOrientation **getImageOrientation** () **const**
Get camera image orientation.

void **setResolution** (*Properties::SensorResolution* *resolution*)
Set sensor resolution.

Properties::SensorResolution **getResolution** () **const**
Get sensor resolution.

void **setFps** (float *fps*)
Set rate at which camera should produce frames

Parameters

- *fps*: Rate in frames per second

float **getFps** () **const**
Get rate at which camera should produce frames

Return Rate in frames per second

std::tuple<int, int> **getResolutionSize** () **const**
Get sensor resolution as size.

int **getResolutionWidth** () **const**
Get sensor resolution width.

int **getResolutionHeight** () **const**
Get sensor resolution height.

Public Members

CameraControl **initialControl**
Initial control options to apply to sensor

Input **inputControl** = { *this, "inputControl", Input::Type::SReceiver, true, 8, { { *DatatypeEnum::CameraControl*, false } } }
Input for *CameraControl* message, which can modify camera parameters in runtime Default queue is blocking with size 8

Output **out** = { *this, "out", Output::Type::MSender, { { *DatatypeEnum::ImgFrame*, false } } }
Outputs *ImgFrame* message that carries RAW8 encoded (grayscale) frame data.
Suitable for use *StereoDepth* node

Private Functions

```
std::string getName () const override
    Retrieves nodes name.

std::vector<Output> getOutputs () override
    Retrieves all nodes outputs.

std::vector<Input> getInputs () override
    Retrieves all nodes inputs.

nlohmann::json getProperties () override

std::shared_ptr<Node> clone () override
```

Private Members

```
Properties properties

std::shared_ptr<RawCameraControl> rawControl
```

We're always happy to help with code or other questions you might have.

3.6.6 NeuralNetwork

Runs a neural inference on input data. Neural network has to be a `.blob` type. Instructions on how to compile your neural network (NN) to `.blob` can be found at [Local OpenVINO Model Conversion](#).

How to place it

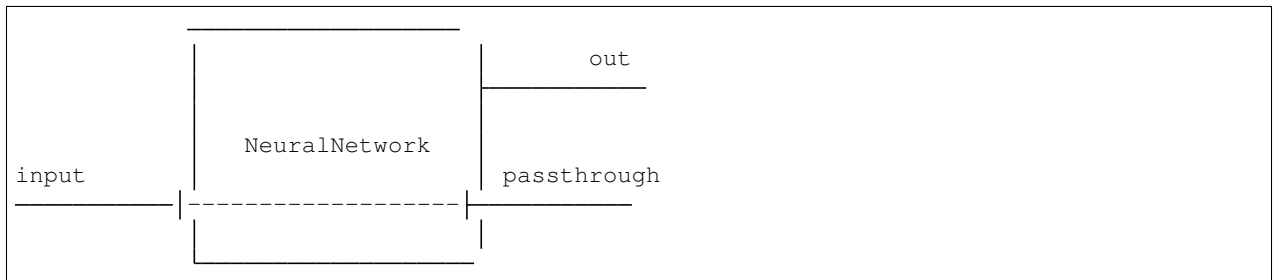
Python

```
pipeline = dai.Pipeline()
nn = pipeline.createNeuralNetwork()
```

C++

```
dai::Pipeline pipeline;
auto nn = pipeline.create<dai::node::NeuralNetwork>();
```

Inputs and Outputs



Message types

- input - *ImgFrame*

- out - *NNData*
- passthrough - *ImgFrame*

Passthrough mechanism

The passthrough mechanism is very useful when a node specifies its input to be non-blocking, where messages can be overwritten. There we don't know on which message the node performed its operation (eg NN, was inference done on frame 25 or skipped 25 and performed inference on 26). At the same time means that if: xlink and host input queues are blocking, and we receive both say passthrough and output we can do a blocking get on both of those queues and be sure to always get matching frames. They might not arrive at the same time, but both of them will arrive, and be in queue in correct spot to be taken out together.

Usage

Python

```
pipeline = dai.Pipeline()
nn = pipeline.createNeuralNetwork()
nn.setBlobPath(bbBlobPath)
cam.out.link(nn.input)

# Send NN out to the host via XLink
nnXout = pipeline.createXLinkOut()
nnXout.setStreamName("nn")
nn.out.link(nnXout.input)

with dai.Device(pipeline) as device:
    qNn = device.getOutputQueue("nn")

    nnData = qNn.get() # Blocking

    # NN can output from multiple layers. Print all layer names:
    print(nnData.getAllLayerNames())

    # Get layer named "Layer1_FP16" as FP16
    layer1Data = nnData.getLayerFp16("Layer1_FP16")

    # You can now decode the output of your NN
```

C++

```
dai::Pipeline pipeline;
auto nn = pipeline.create<dai::node::NeuralNetwork>();
nn->setBlobPath(bbBlobPath);
cam->out.link(nn->input);

// Send NN out to the host via XLink
auto nnXout = pipeline.create<dai::node::XLinkOut>();
nnXout->setStreamName("nn");
nn->out.link(nnXout->input);

dai::Device device(pipeline);
// Start the pipeline
device.startPipeline();
```

(continues on next page)

(continued from previous page)

```

auto qNn = device.getOutputQueue("nn");

auto nnData = qNn->get<dai::NNData>(); // Blocking

// NN can output from multiple layers. Print all layer names:
cout << nnData->getAllLayerNames();

// Get layer named "Layer1_FP16" as FP16
auto layer1Data = nnData->getLayerFp16("Layer1_FP16");

// You can now decode the output of your NN

```

Examples of functionality

- DepplabV3 experiment
- Age/gender experiment
- License plate recognition experiment

Reference

Python

class depthai.NeuralNetwork

NeuralNetwork node. Runs a neural inference on input data.

class Connection

Connection between an Input and Output

class Id

Node identificator. Unique for every node on a single Pipeline

Properties

alias of *depthai.NeuralNetworkProperties*

getAssets (*self*: depthai.Node) → List[depthai.Asset]

Retrieves all nodes assets

getInputs (*self*: depthai.Node) → List[dai::Node::Input]

Retrieves all nodes inputs

getName (*self*: depthai.Node) → str

Retrieves nodes name

getNumInferenceThreads (*self*: depthai.NeuralNetwork) → int

How many inference threads will be used to run the network

Returns Number of threads, 0, 1 or 2. Zero means AUTO

getOutputs (*self*: depthai.Node) → List[dai::Node::Output]

Retrieves all nodes outputs

property id

Id of node

property input

Input message with data to be inferred upon Default queue is blocking with size 5

property out

Outputs NNData message that carries inference results

property passthrough

Passthrough message on which the inference was performed.

Suitable for when input queue is set to non-blocking behavior.

setBlobPath (*self*: [depthai.NeuralNetwork](#), *path*: *str*) → *None*

Load network blob into assets and use once pipeline is started.

Throws if file doesn't exist or isn't a valid network blob.

Parameter path: Path to network blob

setNumInferenceThreads (*self*: [depthai.NeuralNetwork](#), *numThreads*: *int*) → *None*

How many threads should the node use to run the network.

Parameter numThreads: Number of threads to dedicate to this node

setNumNCEPerInferenceThread (*self*: [depthai.NeuralNetwork](#), *numNCEPerThread*: *int*) → *None*

How many Neural Compute Engines should a single thread use for inference

Parameter numNCEPerThread: Number of NCE per thread

setNumPoolFrames (*self*: [depthai.NeuralNetwork](#), *numFrames*: *int*) → *None*

Specifies how many frames will be available in the pool

Parameter numFrames: How many frames will pool have

C++

```
class dai::node::NeuralNetwork: public dai::Node
    NeuralNetwork node. Runs a neural inference on input data.
```

Subclassed by [dai::node::DetectionNetwork](#)

Public Types

```
using Properties = dai::NeuralNetworkProperties
```

Public Functions

```
NeuralNetwork (const std::shared_ptr<PipelineImpl> &par, int64_t nodeId)
```

```
void setBlobPath (const std::string &path)
```

Load network blob into assets and use once pipeline is started.

Throws if file doesn't exist or isn't a valid network blob.

Parameters

- *path*: Path to network blob

```
void setNumPoolFrames (int numFrames)
```

Specifies how many frames will be available in the pool

Parameters

- *numFrames*: How many frames will pool have

```
void setNumInferenceThreads (int numThreads)
```

How many threads should the node use to run the network.

Parameters

- `numThreads`: Number of threads to dedicate to this node

void **setNumNCEPerInferenceThread** (int *numNCEPerThread*)
How many Neural Compute Engines should a single thread use for inference

Parameters

- `numNCEPerThread`: Number of NCE per thread

int **getNumInferenceThreads** ()
How many inference threads will be used to run the network

Return Number of threads, 0, 1 or 2. Zero means AUTO

Public Members

Input **input** = { *this, "in", Input::Type::SReceiver, true, 5, { { *DatatypeEnum::Buffer*, true } } }
Input message with data to be inferred upon Default queue is blocking with size 5

Output **out** = { *this, "out", Output::Type::MSender, { { *DatatypeEnum::NNData*, false } } }
Outputs *NNData* message that carries inference results

Output **passthrough** = { *this, "passthrough", Output::Type::MSender, { { *DatatypeEnum::Buffer*, true } } }
Passthrough message on which the inference was performed.

Suitable for when input queue is set to non-blocking behavior.

Private Functions

std::string **getName** () **const override**
Retrieves nodes name.

std::vector<Output> **getOutputs** () **override**
Retrieves all nodes outputs.

std::vector<Input> **getInputs** () **override**
Retrieves all nodes inputs.

nlohmann::json **getProperties** () **override**

std::shared_ptr<*Node*> **clone** () **override**

tl::optional<*OpenVINO::Version*> **getRequiredOpenVINOVersion** () **override**

Private Members

Properties **properties**

We're always happy to help with code or other questions you might have.

3.6.7 SpatialLocationCalculator

SpatialLocationCalculator will calculate the depth based on the depth map from the `inputDepth` and ROI (region-of-interest) provided from the `inputConfig`. It will average the depth values in the ROI and remove the ones out of range.

How to place it

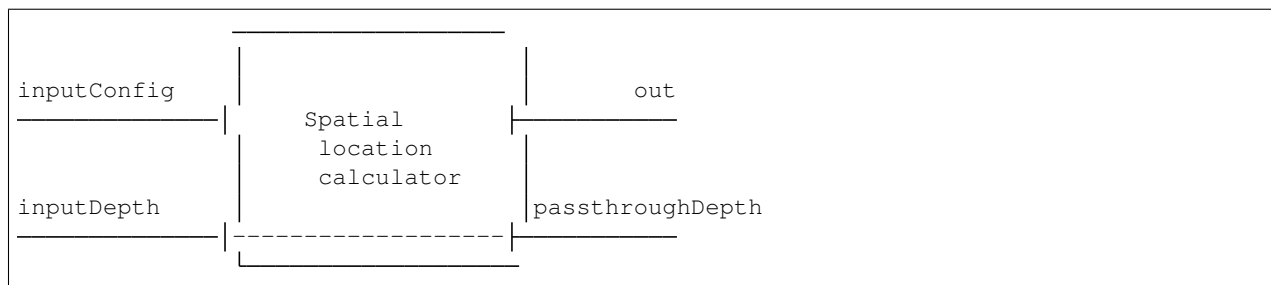
Python

```
pipeline = dai.Pipeline()
spatialCalc = pipeline.SpatialLocationCalculator()
```

C++

```
dai::Pipeline pipeline;
auto spatialCalc = pipeline.create<dai::node::SpatialLocationCalculator>();
```

Inputs and Outputs



Message types

- `inputConfig` - *SpatialLocationCalculatorConfig*
- `inputDepth` - *ImgFrame*
- `out` - *SpatialLocationCalculatorData*
- `passthroughDepth` - *ImgFrame*

Usage

Python

```
pipeline = dai.Pipeline()
spatialCalc = pipeline.SpatialLocationCalculator()
spatialCalc.setWaitForConfigInput(False)

# Set initial config
config = dai.SpatialLocationCalculatorConfigData()
config.depthThresholds.lowerThreshold = 100
config.depthThresholds.upperThreshold = 10000

topLeft = dai.Point2f(0.4, 0.4)
bottomRight = dai.Point2f(0.6, 0.6)
```

(continues on next page)

(continued from previous page)

```

config.roi = dai.Rect(topLeft, bottomRight)

spatial_calc.initialConfig.addROI(config)

# You can later send configs from the host (XLinkIn) / scripting node to the
↳ InputConfig

```

C++

```

dai::Pipeline pipeline;
auto spatialCalc = pipeline.create<dai::node::SpatialLocationCalculator>();
spatialCalc->setWaitForConfigInput(false);

// Set initial config
dai::SpatialLocationCalculatorConfigData config;
config.depthThresholds.lowerThreshold = 100;
config.depthThresholds.upperThreshold = 10000;

dai::Point2f topLeft(0.4f, 0.4f);
dai::Point2f bottomRight(0.6f, 0.6f);
config.roi = dai::Rect(topLeft, bottomRight);

spatialCalc->initialConfig.addROI(config);

// You can later send configs from the host (XLinkIn) / scripting node to the
↳ InputConfig

```

Examples of functionality

- 27 - Spatial location calculator

Reference

Python

```

class depthai.SpatialLocationCalculator
    SpatialLocationCalculator node. Calculates spatial location data on a set of ROIs on depth map.

    class Connection
        Connection between an Input and Output

    class Id
        Node identifier. Unique for every node on a single Pipeline

    Properties
        alias of depthai.SpatialLocationCalculatorProperties

    getAssets (self: depthai.Node) → List[depthai.Asset]
        Retrieves all nodes assets

    getInputs (self: depthai.Node) → List[dai::Node::Input]
        Retrieves all nodes inputs

    getName (self: depthai.Node) → str
        Retrieves nodes name

```

getOutputs (*self*: [depthai.Node](#)) → List[dai::Node::Output]

Retrieves all nodes outputs

property id

Id of node

property initialConfig

Initial config to use when calculating spatial location data.

property inputConfig

Input SpatialLocationCalculatorConfig message with ability to modify parameters in runtime. Default queue is non-blocking with size 4.

property inputDepth

Input message with depth data used to retrieve spatial information about detected object. Default queue is non-blocking with size 4.

property out

Outputs SpatialLocationCalculatorData message that carries spatial location results.

property passthroughDepth

Passthrough message on which the calculation was performed. Suitable for when input queue is set to non-blocking behavior.

setWaitForConfigInput (*self*: [depthai.SpatialLocationCalculator](#), *wait*: *bool*) → None

Specify whether or not wait until configuration message arrives to inputConfig Input.

Parameter wait: True to wait for configuration message, false otherwise.

C++

class [dai::node::SpatialLocationCalculator](#) : **public** [dai::Node](#)

SpatialLocationCalculator node. Calculates spatial location data on a set of ROIs on depth map.

Public Types

using Properties = *dai::SpatialLocationCalculatorProperties*

Public Functions

SpatialLocationCalculator (**const** std::shared_ptr<*PipelineImpl*> &*par*, int64_t *nodeId*)

void **setWaitForConfigInput** (bool *wait*)

Specify whether or not wait until configuration message arrives to inputConfig Input.

Parameters

- *wait*: True to wait for configuration message, false otherwise.

Public Members

SpatialLocationCalculatorConfig **initialConfig**

Initial config to use when calculating spatial location data.

Input **inputConfig** = { *this, "inputConfig", Input::Type::SReceiver, false, 4, { { *DatatypeEnum::SpatialLocationCalculatorConfig*, false } } }

Input *SpatialLocationCalculatorConfig* message with ability to modify parameters in runtime. Default queue is non-blocking with size 4.

Input **inputDepth** = { *this, "inputDepth", Input::Type::SReceiver, false, 4, { { *DatatypeEnum::ImgFrame*, false } } }

Input message with depth data used to retrieve spatial information about detected object. Default queue is non-blocking with size 4.

Output **out** = { *this, "out", Output::Type::MSender, { { *DatatypeEnum::SpatialLocationCalculatorData*, false } } }

Outputs *SpatialLocationCalculatorData* message that carries spatial location results.

Output **passthroughDepth** = { *this, "passthroughDepth", Output::Type::MSender, { { *DatatypeEnum::ImgFrame*, false } } }

Passthrough message on which the calculation was performed. Suitable for when input queue is set to non-blocking behavior.

Private Functions

std::string **getName () const override**

Retrieves nodes name.

std::vector<Input> **getInputs () override**

Retrieves all nodes inputs.

std::vector<Output> **getOutputs () override**

Retrieves all nodes outputs.

nlohmann::json **getProperties () override**

std::shared_ptr<*Node*> **clone () override**

Private Members

std::shared_ptr<*RawSpatialLocationCalculatorConfig*> **rawConfig**

Properties **properties**

We're always happy to help with code or other questions you might have.

3.6.8 SPIOut

SPIOut node is used to send data through to a MCU via SPI. [LUX-ESP32](#) module has integrated an integrated ESP32 connected to the MyriadX via SPI. You can find demos [here](#).

How to place it

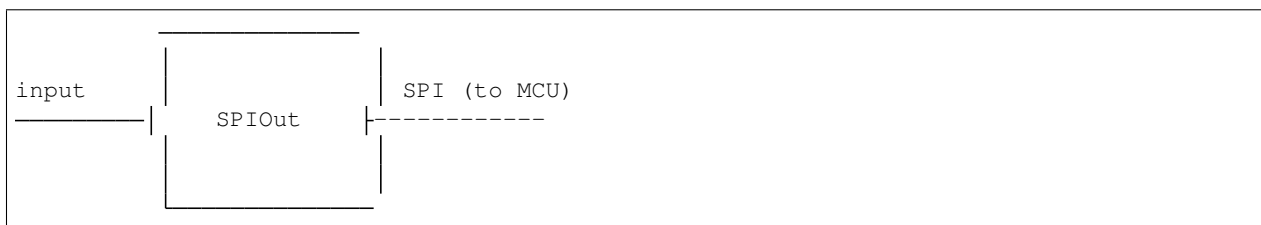
Python

```
pipeline = dai.Pipeline()
spi = pipeline.createSPIOut()
```

C++

```
dai::Pipeline pipeline;
auto spi = pipeline.create<dai::node::SPIOut>();
```

Inputs and Outputs



Message types

- input - Any

Usage

Python

```
pipeline = dai.Pipeline()
spi = pipeline.createSPIOut()

spi.setStreamName("spimetaout")
spi.setBusId(0)
```

C++

```
dai::Pipeline pipeline;
auto spi = pipeline.create<dai::node::SPIOut>();

spi->setStreamName("spimetaout");
spi->setBusId(0);
```

Examples of functionality

- SPI demos (host side)
- ESP32 code demos

Reference

Python

```
class depthai.SPIOut
    SPIOut node. Sends messages over SPI.

    class Connection
        Connection between an Input and Output

    class Id
        Node identificator. Unique for every node on a single Pipeline

    getAssets (self: depthai.Node) → List[depthai.Asset]
        Retrieves all nodes assets

    getInputs (self: depthai.Node) → List[dai::Node::Input]
        Retrieves all nodes inputs

    getName (self: depthai.Node) → str
        Retrieves nodes name

    getOutputs (self: depthai.Node) → List[dai::Node::Output]
        Retrieves all nodes outputs

    property id
        Id of node

    property input
        Input for any type of messages to be transfered over SPI stream

        Default queue is blocking with size 8

    setBusId (self: depthai.SPIOut, id: int) → None
        Specifies SPI Bus number to use

        Parameter id: SPI Bus id

    setStreamName (self: depthai.SPIOut, name: str) → None
        Specifies stream name over which the node will send data

        Parameter name: Stream name
```

C++

```
class dai::node::SPIOut : public dai::Node
    SPIOut node. Sends messages over SPI.
```

Public Functions

```
SPIOut (const std::shared_ptr<PipelineImpl> &par, int64_t nodeId)
```

```
void setStreamName (std::string name)
    Specifies stream name over which the node will send data
```

Parameters

- name: Stream name

```
void setBusId (int id)
    Specifies SPI Bus number to use
```

Parameters

- `id`: SPI Bus id

Public Members

Input **input** = {`*this`, "in", Input::Type::SReceiver, true, 8, {{*DatatypeEnum::Buffer*, true}}}

Input for any type of messages to be transfered over SPI stream

Default queue is blocking with size 8

Private Functions

std::string **getName** () **const**

Retrieves nodes name.

std::vector<Input> **getInputs** ()

Retrieves all nodes inputs.

std::vector<Output> **getOutputs** ()

Retrieves all nodes outputs.

nlohmann::json **getProperties** ()

std::shared_ptr<*Node*> **clone** ()

Private Members

dai::SPIOutProperties **properties**

We're always happy to help with code or other questions you might have.

3.6.9 StereoDepth

Stereo depth node calculates the disparity/depth from two *mono cameras*.

How to place it

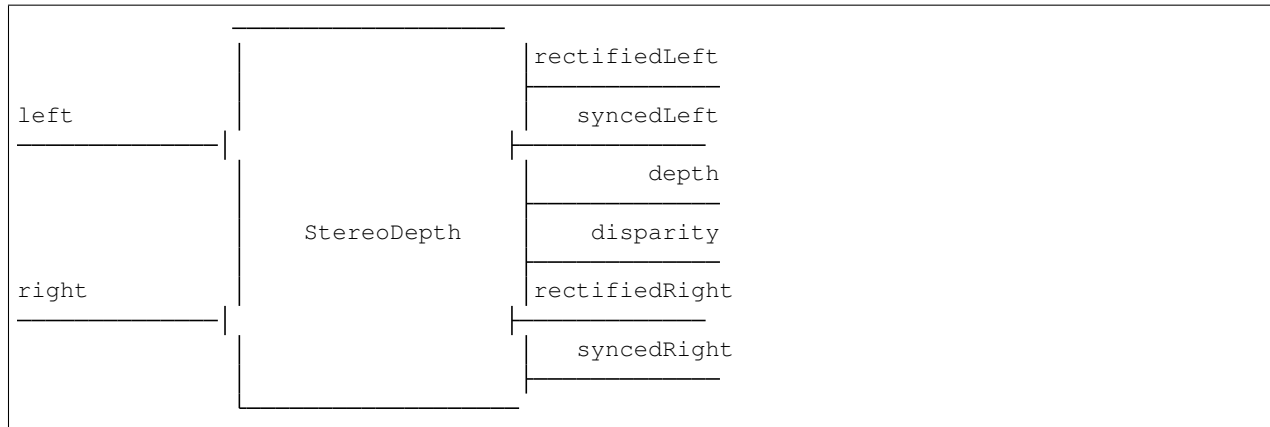
Python

```
pipeline = dai.Pipeline()
stereo = pipeline.createStereoDepth()
```

C++

```
dai::Pipeline pipeline;
auto stereo = pipeline.create<dai::node::StereoDepth>();
```

Inputs and Outputs



Message types

- left - *ImgFrame* from the left *MonoCamera*
- right - *ImgFrame* from the right *MonoCamera*
- rectifiedLeft - *ImgFrame*
- syncedLeft - *ImgFrame*
- depth - *ImgFrame*
- disparity - *ImgFrame*
- rectifiedRight - *ImgFrame*
- syncedRight - *ImgFrame*

Disparity

When calculating the disparity, each pixel in the disparity map gets assigned a confidence value 0..255 by the stereo matching algorithm, as: - 0 - maximum confidence that it holds a valid value - 255 - minimum confidence, so there are chances the value is incorrect (this confidence score is kind-of inverted, if say comparing with NN)

For the final disparity map, a filtering is applied based on the confidence threshold value: the pixels that have their confidence score larger than the threshold get invalidated, i.e. their disparity value is set to zero.

Current limitations

If one or more of the additional depth modes (lrcheck, extended, subpixel) are enabled, then:

- depth output is FP16.
- median filtering is disabled on device.
- with subpixel, either depth or disparity has valid data.

Otherwise, depth output is U16 (in milimeters) and median is functional.

Like on Gen1, either depth or disparity has valid data.

Usage

Python

```
pipeline = dai.Pipeline()
stereo = pipeline.createStereoDepth()

# Better handling for occlusions:
stereo.setLeftRightCheck(False)
# Closer-in minimum depth, disparity range is doubled:
stereo.setExtendedDisparity(False)
# Better accuracy for longer distance, fractional disparity 32-levels:
stereo.setSubpixel(False)

# Define and configure MonoCamera nodes beforehand
left.out.link(stereo.left)
right.out.link(stereo.right)
```

C++

```
dai::Pipeline pipeline;
auto stereo = pipeline.create<dai::node::StereoDepth>();

// Better handling for occlusions:
stereo->setLeftRightCheck(false);
// Closer-in minimum depth, disparity range is doubled:
stereo->setExtendedDisparity(false);
// Better accuracy for longer distance, fractional disparity 32-levels:
stereo->setSubpixel(false);

// Define and configure MonoCamera nodes beforehand
left->out.link(stereo->left);
right->out.link(stereo->right);
```

Examples of functionality

- [03 - Depth Preview](#)
- [10 - Mono & MobilenetSSD & Depth](#)
- [26.1 - RGB & MobilenetSSD with spatial data](#)

Reference

Python

class depthai.StereoDepth

StereoDepth node. Compute stereo disparity and depth from left-right image pair.

class Connection

Connection between an Input and Output

class Id

Node identifier. Unique for every node on a single Pipeline

Properties

alias of `depthai.StereoDepthProperties`

property depth

Outputs ImgFrame message that carries RAW16 encoded (0..65535) depth data in millimeters.

property disparity

Outputs ImgFrame message that carries RAW8 encoded (0..96 or 0..192 for Extended mode) disparity data.

getAssets (*self*: [depthai.Node](#)) → List[[depthai.Asset](#)]

Retrieves all nodes assets

getInputs (*self*: [depthai.Node](#)) → List[dai::Node::Input]

Retrieves all nodes inputs

getName (*self*: [depthai.Node](#)) → str

Retrieves nodes name

getOutputs (*self*: [depthai.Node](#)) → List[dai::Node::Output]

Retrieves all nodes outputs

property id

Id of node

property left

Input for left ImgFrame of left-right pair

Default queue is non-blocking with size 8

loadCalibrationData (*self*: [depthai.StereoDepth](#), *data*: List[int]) → None

Specify calibration data as a vector of bytes

Parameter path: Calibration data. If empty use EEPROM

loadCalibrationFile (*self*: [depthai.StereoDepth](#), *path*: str) → None

Specify local filesystem path to the calibration file

Parameter path: Path to calibration file. If empty use EEPROM

property rectifiedLeft

Outputs ImgFrame message that carries RAW8 encoded (grayscale) rectified frame data.

property rectifiedRight

Outputs ImgFrame message that carries RAW8 encoded (grayscale) rectified frame data.

property right

Input for right ImgFrame of left-right pair

Default queue is non-blocking with size 8

setConfidenceThreshold (*self*: [depthai.StereoDepth](#), *confThr*: int) → None

Confidence threshold for disparity calculation

Parameter confThr: Confidence threshold value 0..255

setEmptyCalibration (*self*: [depthai.StereoDepth](#)) → None

Specify that a passthrough/dummy calibration should be used, when input frames are already rectified (e.g. sourced from recordings on the host)

setExtendedDisparity (*self*: [depthai.StereoDepth](#), *enable*: bool) → None

Disparity range increased from 96 to 192, combined from full resolution and downscaled images.

Suitable for short range objects

setInputResolution (*self*: [depthai.StereoDepth](#), *width*: int, *height*: int) → None

Specify input resolution size

Optional if MonoCamera exists, otherwise necessary

setLeftRightCheck (*self*: [depthai.StereoDepth](#), *enable*: *bool*) → *None*

Computes and combines disparities in both L-R and R-L directions, and combine them.

For better occlusion handling

setMedianFilter (*self*: [depthai.StereoDepth](#), *median*: [dai::StereoDepthProperties::MedianFilter](#)) → *None*

Parameter median: Set kernel size for disparity/depth median filtering, or disable

setOutputDepth (*self*: [depthai.StereoDepth](#), *enable*: *bool*) → *None*

Enable outputting ‘depth’ stream (converted from disparity). In certain configurations, this will disable ‘disparity’ stream

setOutputRectified (*self*: [depthai.StereoDepth](#), *enable*: *bool*) → *None*

Enable outputting rectified frames. Optimizes computation on device side when disabled

setRectifyEdgeFillColor (*self*: [depthai.StereoDepth](#), *color*: *int*) → *None*

Fill color for missing data at frame edges

Parameter color: Grayscale 0..255, or -1 to replicate pixels

setRectifyMirrorFrame (*self*: [depthai.StereoDepth](#), *enable*: *bool*) → *None*

Mirror rectified frames

Parameter enable: True for normal disparity/depth, otherwise mirrored

setSubpixel (*self*: [depthai.StereoDepth](#), *enable*: *bool*) → *None*

Computes disparity with sub-pixel interpolation (5 fractional bits).

Suitable for long range

property syncedLeft

Passthrough ImgFrame message from ‘left’ Input.

property syncedRight

Passthrough ImgFrame message from ‘right’ Input.

C++

class [dai::node::StereoDepth](#) : **public** [dai::Node](#)

[StereoDepth](#) node. Compute stereo disparity and depth from left-right image pair.

Public Types

using Properties = [dai::StereoDepthProperties](#)

Public Functions

StereoDepth (**const** std::shared_ptr<[PipelineImpl](#)> &*par*, int64_t *nodeId*)

void **loadCalibrationFile** (**const** std::string &*path*)

Specify local filesystem path to the calibration file

Parameters

- *path*: Path to calibration file. If empty use EEPROM

void **loadCalibrationData** (**const** std::vector<std::uint8_t> &*data*)

Specify calibration data as a vector of bytes

Parameters

- `path`: Calibration data. If empty use EEPROM

void **setEmptyCalibration** ()

Specify that a passthrough/dummy calibration should be used, when input frames are already rectified (e.g. sourced from recordings on the host)

void **setInputResolution** (int *width*, int *height*)

Specify input resolution size

Optional if *MonoCamera* exists, otherwise necessary

void **setMedianFilter** (*Properties::MedianFilter median*)

Parameters

- `median`: Set kernel size for disparity/depth median filtering, or disable

void **setConfidenceThreshold** (int *confThr*)

Confidence threshold for disparity calculation

Parameters

- `confThr`: Confidence threshold value 0..255

void **setLeftRightCheck** (bool *enable*)

Computes and combines disparities in both L-R and R-L directions, and combine them.

For better occlusion handling

void **setSubpixel** (bool *enable*)

Computes disparity with sub-pixel interpolation (5 fractional bits).

Suitable for long range

void **setExtendedDisparity** (bool *enable*)

Disparity range increased from 96 to 192, combined from full resolution and downsampled images.

Suitable for short range objects

void **setRectifyEdgeFillColor** (int *color*)

Fill color for missing data at frame edges

Parameters

- `color`: Grayscale 0..255, or -1 to replicate pixels

void **setRectifyMirrorFrame** (bool *enable*)

Mirror rectified frames

Parameters

- `enable`: True for normal disparity/depth, otherwise mirrored

void **setOutputRectified** (bool *enable*)

Enable outputting rectified frames. Optimizes computation on device side when disabled

void **setOutputDepth** (bool *enable*)

Enable outputting ‘depth’ stream (converted from disparity). In certain configurations, this will disable ‘disparity’ stream

Public Members

Input **left** = { *this, "left", Input::Type::SReceiver, false, 8, {{ *DatatypeEnum::ImgFrame*, true }} }

Input for left *ImgFrame* of left-right pair

Default queue is non-blocking with size 8

Input **right** = { *this, "right", Input::Type::SReceiver, false, 8, {{ *DatatypeEnum::ImgFrame*, true }} }

Input for right *ImgFrame* of left-right pair

Default queue is non-blocking with size 8

Output **depth** = { *this, "depth", Output::Type::MSender, {{ *DatatypeEnum::ImgFrame*, false }} }

Outputs *ImgFrame* message that carries RAW16 encoded (0..65535) depth data in millimeters.

Output **disparity** = { *this, "disparity", Output::Type::MSender, {{ *DatatypeEnum::ImgFrame*, false }} }

Outputs *ImgFrame* message that carries RAW8 encoded (0..96 or 0..192 for Extended mode) disparity data.

Output **syncedLeft** = { *this, "syncedLeft", Output::Type::MSender, {{ *DatatypeEnum::ImgFrame*, false }} }

Passthrough *ImgFrame* message from 'left' Input.

Output **syncedRight** = { *this, "syncedRight", Output::Type::MSender, {{ *DatatypeEnum::ImgFrame*, false }} }

Passthrough *ImgFrame* message from 'right' Input.

Output **rectifiedLeft** = { *this, "rectifiedLeft", Output::Type::MSender, {{ *DatatypeEnum::ImgFrame*, false }} }

Outputs *ImgFrame* message that carries RAW8 encoded (grayscale) rectified frame data.

Output **rectifiedRight** = { *this, "rectifiedRight", Output::Type::MSender, {{ *DatatypeEnum::ImgFrame*, false }} }

Outputs *ImgFrame* message that carries RAW8 encoded (grayscale) rectified frame data.

Private Functions

std::string **getName** () **const override**

Retrieves nodes name.

std::vector<Output> **getOutputs** () **override**

Retrieves all nodes outputs.

std::vector<Input> **getInputs** () **override**

Retrieves all nodes inputs.

nlohmann::json **getProperties** () **override**

std::shared_ptr<*Node*> **clone** () **override**

Private Members

Properties **properties**

We're always happy to help with code or other questions you might have.

3.6.10 SystemLogger

SystemLogger node is used to get *system information* of the device.

How to place it

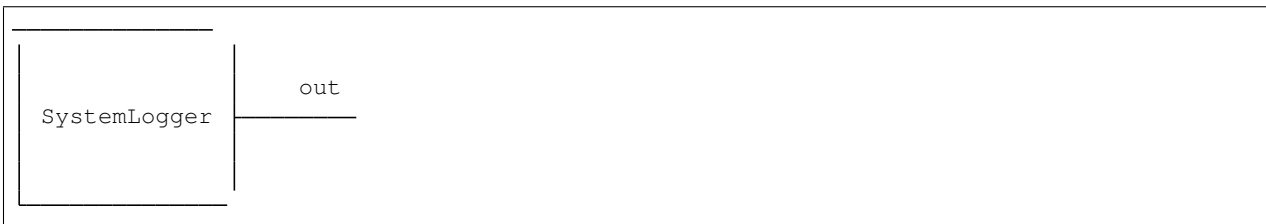
Python

```
pipeline = dai.Pipeline()
logger = pipeline.createSystemLogger()
```

C++

```
dai::Pipeline pipeline;
auto logger = pipeline.create<dai::node::SystemLogger>();
```

Inputs and Outputs



Message types

- out - *SystemInformation*

Usage

Python

```
pipeline = dai.Pipeline()
logger = pipeline.createSystemLogger()
logger.setRate(1) # 1 Hz

# Send system info to the host via XLink
xout = pipeline.createXLinkOut()
xout.setStreamName("sysinfo")
logger.out.link(xout.input)
```

C++

```
dai::Pipeline pipeline;
auto logger = pipeline.create<dai::node::SystemLogger>();\
logger->setRate(1.0f); // 1 Hz

// Send system info to the host via XLink
auto xout = pipeline.create<dai::node::XLinkOut>();
xout->setStreamName("sysinfo");
logger->out.link(xout->input);
```


Examples of functionality

- 27 - *Spatial location calculator*

Reference

Python

class `depthai.SystemLogger`

SystemLogger node. Send system information periodically.

class `Connection`

Connection between an Input and Output

class `Id`

Node identifier. Unique for every node on a single Pipeline

getAssets (*self*: `depthai.Node`) → List[`depthai.Asset`]

Retrieves all nodes assets

getInputs (*self*: `depthai.Node`) → List[`dai::Node::Input`]

Retrieves all nodes inputs

getName (*self*: `depthai.Node`) → str

Retrieves nodes name

getOutputs (*self*: `depthai.Node`) → List[`dai::Node::Output`]

Retrieves all nodes outputs

property `id`

Id of node

property `out`

Outputs SystemInformation message that carries various system information like memory and CPU usage, temperatures, ...

setRate (*self*: `depthai.SystemLogger`, *hz*: float) → None

Specify logging rate, at which messages will be sent to out output

Parameter hz: Sending rate in hertz (messages per second)

C++

class `dai::node::SystemLogger` : public `dai::Node`

SystemLogger node. Send system information periodically.

Public Functions

SystemLogger (**const** std::shared_ptr<*PipelineImpl*> &*par*, int64_t *nodeId*)

void **setRate** (float *hz*)

Specify logging rate, at which messages will be sent to out output

Parameters

- *hz*: Sending rate in hertz (messages per second)

Public Members

Output **out** = { *this, "out", Output::Type::MSender, {{ *DatatypeEnum::SystemInformation*, false }} }

Outputs *SystemInformation* message that carries various system information like memory and CPU usage, temperatures, ...

Private Functions

std::string **getName** () **const override**
Retrieves nodes name.

std::vector<Input> **getInputs** () **override**
Retrieves all nodes inputs.

std::vector<Output> **getOutputs** () **override**
Retrieves all nodes outputs.

nlohmann::json **getProperties** () **override**

std::shared_ptr<*Node*> **clone** () **override**

Private Members

dai::SystemLoggerProperties **properties**

We're always happy to help with code or other questions you might have.

3.6.11 VideoEncoder

VideoEncoder node is used to encode *image frames* into H264/H265/JPEG.

How to place it

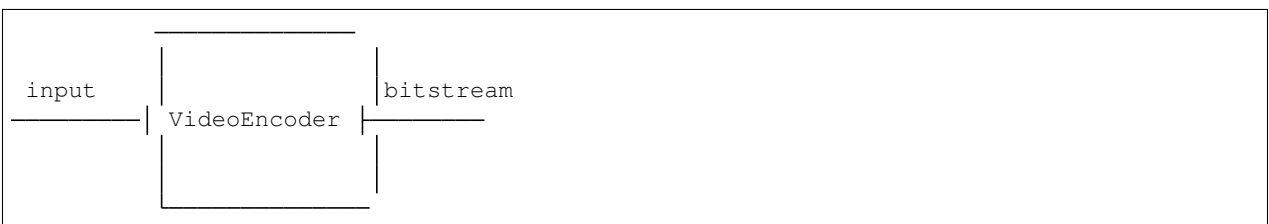
Python

```
pipeline = dai.Pipeline()
encoder = pipeline.createVideoEncoder()
```

C++

```
dai::Pipeline pipeline;
auto encoder = pipeline.create<dai::node::VideoEncoder>();
```

Inputs and Outputs



Message types

- input - *ImgFrame*
- bitstream - *ImgFrame*

Usage

Python

```
pipeline = dai.Pipeline()

# Create ColorCamera beforehand
# Set H265 encoding for the ColorCamera video output
videoEncoder = pipeline.createVideoEncoder()
videoEncoder.setDefaultProfilePreset(cam.getVideoSize(), cam.getFps(), dai.
↳VideoEncoderProperties.Profile.H265_MAIN)

# Create MJPEG encoding for still images
stillEncoder = pipeline.createVideoEncoder()
stillEncoder.setDefaultProfilePreset(cam.getStillSize(), 1, dai.
↳VideoEncoderProperties.Profile.MJPEG)

cam.still.link(stillEncoder.input)
cam.video.link(videoEncoder.input)
```

C++

```
dai::Pipeline pipeline;

// Create ColorCamera beforehand
// Set H265 encoding for the ColorCamera video output
auto videoEncoder = pipeline.create<dai::node::VideoEncoder>();
videoEncoder->setDefaultProfilePreset(cam->getVideoSize(), cam->getFps(),
↳dai::VideoEncoderProperties::Profile::H265_MAIN);

// Create MJPEG encoding for still images
stillEncoder = pipeline.createVideoEncoder();
stillEncoder->setDefaultProfilePreset(cam->getStillSize(), 1,
↳dai::VideoEncoderProperties::Profile::MJPEG);

cam->still.link(stillEncoder->input);
cam->video.link(videoEncoder->input);
```

Examples of functionality

- 04 - RGB Encoding
- 13 - Encoding Max Limit
- 18 - RGB Encoding with MobilenetSSD

Reference

Python

class `depthai.VideoEncoder`

VideoEncoder node. Encodes frames into MJPEG, H264 or H265.

class `Connection`

Connection between an Input and Output

class `Id`

Node identificator. Unique for every node on a single Pipeline

Properties

alias of `depthai.VideoEncoderProperties`

property `bitstream`

Outputs ImgFrame message that carries BITSTREAM encoded (MJPEG, H264 or H265) frame data.

getAssets (*self*: `depthai.Node`) → List[`depthai.Asset`]

Retrieves all nodes assets

getBitrate (*self*: `depthai.VideoEncoder`) → int

Get bitrate in bps

getBitrateKbps (*self*: `depthai.VideoEncoder`) → int

Get bitrate in kbps

getFrameRate (*self*: `depthai.VideoEncoder`) → int

Get frame rate

getHeight (*self*: `depthai.VideoEncoder`) → int

Get input height

getInputs (*self*: `depthai.Node`) → List[dai::Node::Input]

Retrieves all nodes inputs

getKeyframeFrequency (*self*: `depthai.VideoEncoder`) → int

Get keyframe frequency

getLossless (*self*: `depthai.VideoEncoder`) → bool

Get lossless mode. Applies only when using [M]JPEG profile.

getName (*self*: `depthai.Node`) → str

Retrieves nodes name

getNumBframes (*self*: `depthai.VideoEncoder`) → int

Get number of B frames

getNumFramesPool (*self*: `depthai.VideoEncoder`) → int

Get number of frames in pool

Returns Number of pool frames

getOutputs (*self*: `depthai.Node`) → List[dai::Node::Output]

Retrieves all nodes outputs

getProfile (*self*: `depthai.VideoEncoder`) → dai::VideoEncoderProperties::Profile

Get profile

getQuality (*self*: `depthai.VideoEncoder`) → int

Get quality

getRateControlMode (*self*: [depthai.VideoEncoder](#)) → [dai::VideoEncoderProperties::RateControlMode](#)

Get rate control mode

getSize (*self*: [depthai.VideoEncoder](#)) → [Tuple\[int, int\]](#)

Get input size

getWidth (*self*: [depthai.VideoEncoder](#)) → [int](#)

Get input width

property id

Id of node

property input

Input for NV12 ImgFrame to be encoded Default queue is blocking with size set by 'setNumFramesPool' (4).

setBitrate (*self*: [depthai.VideoEncoder](#), *bitrateKbps*: [int](#)) → [None](#)

Set output bitrate in bps. Final bitrate depends on rate control mode

setBitrateKbps (*self*: [depthai.VideoEncoder](#), *bitrateKbps*: [int](#)) → [None](#)

Set output bitrate in kbps. Final bitrate depends on rate control mode

setDefaultProfilePreset (**args*, ***kwargs*)

Overloaded function.

1. `setDefaultProfilePreset(self: depthai.VideoEncoder, width: int, height: int, fps: float, profile: dai::VideoEncoderProperties::Profile) -> None`

Sets a default preset based on specified input size, frame rate and profile

Parameter width: Input frame width

Parameter height: Input frame height

Parameter fps: Frame rate in frames per second

Parameter profile: Encoding profile

2. `setDefaultProfilePreset(self: depthai.VideoEncoder, size: Tuple[int, int], fps: float, profile: dai::VideoEncoderProperties::Profile) -> None`

Sets a default preset based on specified input size, frame rate and profile

Parameter size: Input frame size

Parameter fps: Frame rate in frames per second

Parameter profile: Encoding profile

setFrameRate (*self*: [depthai.VideoEncoder](#), *frameRate*: [int](#)) → [None](#)

Sets expected frame rate

Parameter frameRate: Frame rate in frames per second

setKeyframeFrequency (*self*: [depthai.VideoEncoder](#), *freq*: [int](#)) → [None](#)

Set keyframe frequency. Every Nth frame a keyframe is inserted.

Applicable only to H264 and H265 profiles

Examples:

- 30 FPS video, keyframe frequency: 30. Every 1s a keyframe will be inserted
- 60 FPS video, keyframe frequency: 180. Every 3s a keyframe will be inserted

setLossless (*self*: [depthai.VideoEncoder](#), *arg0*: *bool*) → *None*

Set lossless mode. Applies only to [M]JPEG profile

Parameter lossless: True to enable lossless jpeg encoding, false otherwise

setNumBFrames (*self*: [depthai.VideoEncoder](#), *numBFrames*: *int*) → *None*

Set number of B frames to be inserted

setNumFramesPool (*self*: [depthai.VideoEncoder](#), *frames*: *int*) → *None*

Set number of frames in pool

Parameter frames: Number of pool frames

setProfile (**args*, ***kwargs*)

Overloaded function.

1. **setProfile**(*self*: [depthai.VideoEncoder](#), *size*: *Tuple[int, int]*, *profile*: *dai::VideoEncoderProperties::Profile*) → *None*

Set encoding profile

2. **setProfile**(*self*: [depthai.VideoEncoder](#), *width*: *int*, *height*: *int*, *profile*: *dai::VideoEncoderProperties::Profile*) → *None*

Set encoding profile

setQuality (*self*: [depthai.VideoEncoder](#), *quality*: *int*) → *None*

Set quality

Parameter quality: Value between 0-100%. Approximates quality

setRateControlMode (*self*: [depthai.VideoEncoder](#), *mode*: *dai::VideoEncoderProperties::RateControlMode*) → *None*

Set rate control mode

C++

```
class dai::node::VideoEncoder : public dai::Node
```

VideoEncoder node. Encodes frames into MJPEG, H264 or H265.

Public Types

```
using Properties = dai::VideoEncoderProperties
```

Public Functions

```
VideoEncoder (const std::shared_ptr<PipelineImpl> &par, int64_t nodeId)
```

```
void setDefaultProfilePreset (int width, int height, float fps, Properties::Profile profile)
```

Sets a default preset based on specified input size, frame rate and profile

Parameters

- *width*: Input frame width
- *height*: Input frame height
- *fps*: Frame rate in frames per second
- *profile*: Encoding profile

```
void setDefaultProfilePreset (std::tuple<int, int> size, float fps, Properties::Profile profile)
```

Sets a default preset based on specified input size, frame rate and profile

Parameters

- `size`: Input frame size
- `fps`: Frame rate in frames per second
- `profile`: Encoding profile

void **setNumFramesPool** (int *frames*)
Set number of frames in pool

Parameters

- `frames`: Number of pool frames

int **getNumFramesPool** () **const**
Get number of frames in pool

Return Number of pool frames

void **setRateControlMode** (*Properties::RateControlMode mode*)
Set rate control mode.

void **setProfile** (std::tuple<int, int> *size*, *Properties::Profile profile*)
Set encoding profile.

void **setProfile** (int *width*, int *height*, *Properties::Profile profile*)
Set encoding profile.

void **setBitrate** (int *bitrate*)
Set output bitrate in bps. Final bitrate depends on rate control mode.

void **setBitrateKbps** (int *bitrateKbps*)
Set output bitrate in kbps. Final bitrate depends on rate control mode.

void **setKeyframeFrequency** (int *freq*)
Set keyframe frequency. Every Nth frame a keyframe is inserted.

Applicable only to H264 and H265 profiles

Examples:

- 30 FPS video, keyframe frequency: 30. Every 1s a keyframe will be inserted
- 60 FPS video, keyframe frequency: 180. Every 3s a keyframe will be inserted

void **setNumBFrames** (int *numBFrames*)
Set number of B frames to be inserted.

void **setQuality** (int *quality*)
Set quality

Parameters

- `quality`: Value between 0-100%. Approximates quality

void **setLossless** (bool *lossless*)
Set lossless mode. Applies only to [M]JPEG profile

Parameters

- `lossless`: True to enable lossless jpeg encoding, false otherwise

void **setFrameRate** (int *frameRate*)
Sets expected frame rate

Parameters

- `frameRate`: Frame rate in frames per second

Properties::RateControlMode **getRateControlMode () const**
Get rate control mode.

Properties::Profile **getProfile () const**
Get profile.

int getBitrate () const
Get bitrate in bps.

int getBitrateKbps () const
Get bitrate in kbps.

int getKeyframeFrequency () const
Get keyframe frequency.

int getNumBFrames () const
Get number of B frames.

int getQuality () const
Get quality.

std::tuple<int, int> getSize () const
Get input size.

int getWidth () const
Get input width.

int getHeight () const
Get input height.

int getFrameRate () const
Get frame rate.

bool getLossless () const
Get lossless mode. Applies only when using [M]JPEG profile.

Public Members

Input **input** = {**this*, "in", Input::Type::SReceiver, true, 4, {{*DatatypeEnum::ImgFrame*, true}}}
Input for NV12 *ImgFrame* to be encoded Default queue is blocking with size set by 'setNumFramesPool' (4).

Output **bitstream** = {**this*, "bitstream", Output::Type::MSender, {{*DatatypeEnum::ImgFrame*, false}}}
Outputs *ImgFrame* message that carries BITSTREAM encoded (MJPEG, H264 or H265) frame data.

Private Functions

std::string getName () const override
Retrieves nodes name.

std::vector<Input> getInputs () override
Retrieves all nodes inputs.

std::vector<Output> getOutputs () override
Retrieves all nodes outputs.

nlohmann::json getProperties () override


```
std::shared_ptr<Node> clone() override
```

Private Members

Properties **properties**

We're always happy to help with code or other questions you might have.

3.6.12 XLinkIn

XLinkIn node is used to send data from the host to the device via XLink.

How to place it

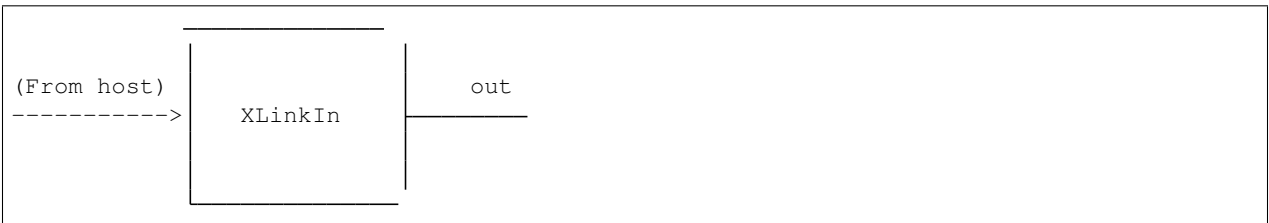
Python

```
pipeline = dai.Pipeline()
xlinkIn = pipeline.createXLinkIn()
```

C++

```
dai::Pipeline pipeline;
auto xlinkIn = pipeline.create<dai::node::XLinkIn>();
```

Inputs and Outputs



Message types

- out - Any

Usage

Python

```
pipeline = dai.Pipeline()
xIn = pipeline.createXLinkIn()
xIn.setStreamName("camControl")

# Create ColorCamera beforehand
xIn.out.link(cam.inputControl)

with dai.Device(pipeline) as device:
    device.startPipeline()
    qCamControl = device.getInputQueue("camControl")
```

(continues on next page)

(continued from previous page)

```
# Send a message to the ColorCamera to capture a still image
ctrl = dai.CameraControl()
ctrl.setCaptureStill(True)
qCamControl.send(ctrl)
```

C++

```
dai::Pipeline pipeline;
auto xIn = pipeline.create<dai::node::XLinkIn>();
xIn->setStreamName("camControl");

// Create ColorCamera beforehand
xIn->out.link(cam->inputControl);

// Connect to the device
dai::Device device(pipeline);
device.startPipeline();

auto qCamControl = device.getInputQueue("camControl");

// Send a message to the ColorCamera to capture a still image
dai::CameraControl ctrl;
ctrl.setCaptureStill(true);
qCamControl->send(ctrl)
```

Examples of functionality

- [14.1 - Color Camera Control](#)
- [17 - Video & MobilenetSSD](#)
- [20 - Color Rotate Warp](#)

Reference

Python

```
class depthai.XLinkIn
    XLinkIn node. Receives messages over XLink.

    class Connection
        Connection between an Input and Output

    class Id
        Node identificator. Unique for every node on a single Pipeline

    getAssets (self: depthai.Node) → List[depthai.Asset]
        Retrieves all nodes assets

    getInputs (self: depthai.Node) → List[dai::Node::Input]
        Retrieves all nodes inputs

    getMaxDataSize (self: depthai.XLinkIn) → int
        Get maximum messages size in bytes
```

getName (*self*: [depthai.Node](#)) → str
Retrieves nodes name

getNumFrames (*self*: [depthai.XLinkIn](#)) → int
Get number of frames in pool

getOutputs (*self*: [depthai.Node](#)) → List[dai::Node::Output]
Retrieves all nodes outputs

getStreamName (*self*: [depthai.XLinkIn](#)) → str
Get stream name

property id
Id of node

property out
Outputs message of same type as send from host.

setMaxDataSize (*self*: [depthai.XLinkIn](#), *maxDataSize*: int) → None
Set maximum message size it can receive

Parameter maxDataSize: Maximum size in bytes

setNumFrames (*self*: [depthai.XLinkIn](#), *numFrames*: int) → None
Set number of frames in pool for sending messages forward

Parameter numFrames: Maximum number of frames in pool

setStreamName (*self*: [depthai.XLinkIn](#), *streamName*: str) → None
Specifies XLink stream name to use.

The name should not start with double underscores ‘__’, as those are reserved for internal use.

Parameter name: Stream name

C++

class [dai::node::XLinkIn](#) : **public** [dai::Node](#)
XLinkIn node. Receives messages over XLink.

Public Functions

XLinkIn (**const** std::shared_ptr<[PipelineImpl](#)> &par, int64_t *nodeId*)

void **setStreamName** (**const** std::string &*name*)
Specifies XLink stream name to use.

The name should not start with double underscores ‘__’, as those are reserved for internal use.

Parameters

- *name*: Stream name

void **setMaxDataSize** (std::uint32_t *maxDataSize*)
Set maximum message size it can receive

Parameters

- *maxDataSize*: Maximum size in bytes

void **setNumFrames** (std::uint32_t *numFrames*)
Set number of frames in pool for sending messages forward

Parameters

- numFrames: Maximum number of frames in pool

std::string **getStreamName** () **const**
Get stream name.

std::uint32_t **getMaxDataSize** () **const**
Get maximum messages size in bytes.

std::uint32_t **getNumFrames** () **const**
Get number of frames in pool.

Public Members

Output **out** = { *this, "out", Output::Type::MSender, {{ *DatatypeEnum::Buffer*, true }} }
Outputs message of same type as send from host.

Private Functions

std::string **getName** () **const override**
Retrieves nodes name.

std::vector<Input> **getInputs** () **override**
Retrieves all nodes inputs.

std::vector<Output> **getOutputs** () **override**
Retrieves all nodes outputs.

nlohmann::json **getProperties** () **override**

std::shared_ptr<*Node*> **clone** () **override**

Private Members

dai::XLinkInProperties **properties**

We're always happy to help with code or other questions you might have.

3.6.13 XLinkOut

XLinkOut node is used to send data from the device to the host via XLink.

How to place it

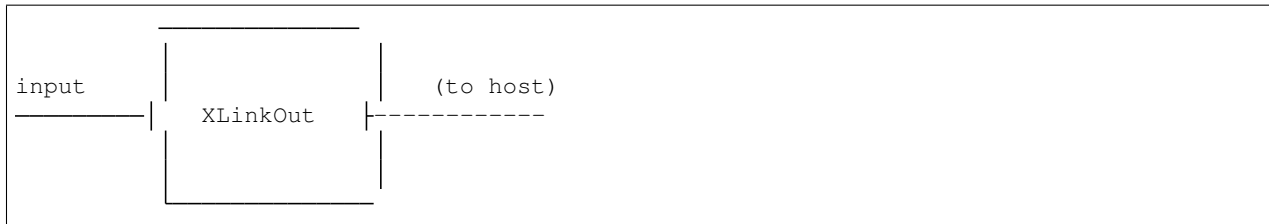
Python

```
pipeline = dai.Pipeline()
xlinkOut = pipeline.createXLinkOut()
```

C++

```
dai::Pipeline pipeline;
auto xlinkOut = pipeline.create<dai::node::XLinkOut>();
```

Inputs and Outputs



Message types

- input - Any

Usage

Python

```

pipeline = dai.Pipeline()
xOut = pipeline.createXLinkOut()
xOut.setStreamName("camOut")

# Here we will send camera preview (ImgFrame) to the host via XLink.
# Host can then display the frame to the user
cam.preview.link(xOut.input)

```

C++

```

dai::Pipeline pipeline;
auto xOut = pipeline.create<dai::node::XLinkOut>();
xOut->setStreamName("camOut");

// Here we will send camera preview (ImgFrame) to the host via XLink.
// Host can then display the frame to the user
cam->preview.link(xOut->input);

```

Examples of functionality

- 01 - RGB Preview
- 09 - Mono & MobilenetSSD
- 10 - Mono & MobilenetSSD & Depth

Reference

Python

class depthai.XLinkOut

XLinkOut node. Sends messages over XLink.

class Connection

Connection between an Input and Output

class Id

Node identifier. Unique for every node on a single Pipeline

getAssets (*self*: [depthai.Node](#)) → List[[depthai.Asset](#)]
Retrieves all nodes assets

getFpsLimit (*self*: [depthai.XLinkOut](#)) → float
Get rate limit in messages per second

getInputs (*self*: [depthai.Node](#)) → List[dai::Node::Input]
Retrieves all nodes inputs

getMetadataOnly (*self*: [depthai.XLinkOut](#)) → bool
Get whether to transfer only messages attributes and not buffer data

getName (*self*: [depthai.Node](#)) → str
Retrieves nodes name

getOutputs (*self*: [depthai.Node](#)) → List[dai::Node::Output]
Retrieves all nodes outputs

getStreamName (*self*: [depthai.XLinkOut](#)) → str
Get stream name

property id
Id of node

property input
Input for any type of messages to be transfered over XLink stream
Default queue is blocking with size 8

setFpsLimit (*self*: [depthai.XLinkOut](#), *fpsLimit*: float) → None
Specifies a message sending limit. It's approximated from specified rate.
Parameter fps: Approximate rate limit in messages per second

setMetadataOnly (*self*: [depthai.XLinkOut](#), *arg0*: bool) → None
Specify whether to transfer only messages attributes and not buffer data

setStreamName (*self*: [depthai.XLinkOut](#), *streamName*: str) → None
Specifies XLink stream name to use.
The name should not start with double underscores '___', as those are reserved for internal use.
Parameter name: Stream name

C++

```
class dai::node::XLinkOut : public dai::Node
    XLinkOut node. Sends messages over XLink.
```

Public Functions

XLinkOut (const std::shared_ptr<[PipelineImpl](#)> &par, int64_t nodeId)

void **setStreamName** (const std::string &name)
Specifies XLink stream name to use.
The name should not start with double underscores '___', as those are reserved for internal use.

Parameters

- name: Stream name

void **setFpsLimit** (float fps)
Specifies a message sending limit. It's approximated from specified rate.

Parameters

- `fps`: Approximate rate limit in messages per second

void **setMetadataOnly** (bool *metadataOnly*)
Specify whether to transfer only messages attributes and not buffer data

std::string **getStreamName** () **const**
Get stream name.

float **getFpsLimit** () **const**
Get rate limit in messages per second.

bool **getMetadataOnly** () **const**
Get whether to transfer only messages attributes and not buffer data.

Public Members

Input **input** = { *this, "in", Input::Type::SReceiver, true, 8, { { *DatatypeEnum::Buffer*, true } } }

Input for any type of messages to be transfered over XLink stream

Default queue is blocking with size 8

Private Functions

std::string **getName** () **const override**
Retrieves nodes name.

std::vector<Input> **getInputs** () **override**
Retrieves all nodes inputs.

std::vector<Output> **getOutputs** () **override**
Retrieves all nodes outputs.

nlohmann::json **getProperties** () **override**

std::shared_ptr<*Node*> **clone** () **override**

Private Members

dai::XLinkOutProperties **properties**

We're always happy to help with code or other questions you might have.

3.6.14 YoloDetectionNetwork

Yolo detection network node is very similar to *NeuralNetwork* (in fact it extends it). The only difference is that this node is specifically for the (tiny) **Yolo V3/V4** NN and it decodes the result of the NN on device. This means that `Out` of this node is not a *NNData* (a byte array) but a *ImgDetections* that can easily be used in your code.

How to place it

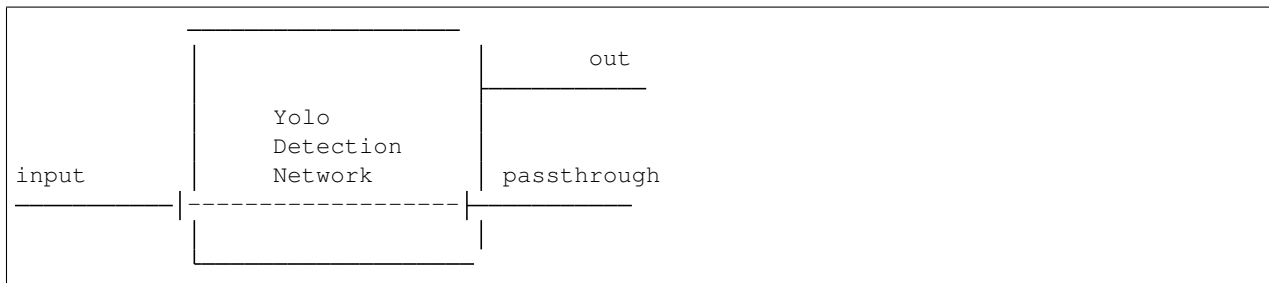
Python

```
pipeline = dai.Pipeline()
yoloDet = pipeline.createYoloDetectionNetwork()
```

C++

```
dai::Pipeline pipeline;
auto yoloDet = pipeline.create<dai::node::YoloDetectionNetwork>();
```

Inputs and Outputs



Message types

- input - *ImgFrame*
- out - *ImgDetections*
- passthrough - *ImgFrame*

Usage

Python

```
pipeline = dai.Pipeline()
yoloDet = pipeline.createYoloDetectionNetwork()
yoloDet.setBlobPath(nnBlobPath)

# Yolo specific parameters
yoloDet.setConfidenceThreshold(0.5)
yoloDet.setNumClasses(80)
yoloDet.setCoordinateSize(4)
yoloDet.setAnchors(np.array([10,14, 23,27, 37,58, 81,82, 135,169, 344,319]))
yoloDet.setAnchorMasks({"side26": np.array([1, 2, 3]), "side13": np.array([3, 4, 5])})
yoloDet.setIouThreshold(0.5)
```

C++

```
dai::Pipeline pipeline;
auto yoloDet = pipeline.create<dai::node::YoloDetectionNetwork>();
yoloDet->setBlobPath(nnBlobPath);

// yolo specific parameters
yoloDet->setConfidenceThreshold(0.5f);
```

(continues on next page)

(continued from previous page)

```

yoloDet->setNumClasses(80);
yoloDet->setCoordinateSize(4);
yoloDet->setAnchors({10, 14, 23, 27, 37, 58, 81, 82, 135, 169, 344, 319});
yoloDet->setAnchorMasks({{"side13", {3, 4, 5}}, {"side26", {1, 2, 3}}});
yoloDet->setIouThreshold(0.5f);

```

Examples of functionality

- [22.1 - RGB & TinyYoloV3 decoding on device](#)
- [22.2 - RGB & TinyYoloV4 decoding on device](#)

Reference

Python

```

class depthai.YoloDetectionNetwork
    YoloDetectionNetwork node. Parses Yolo results

    class Connection
        Connection between an Input and Output

    class Id
        Node identificator. Unique for every node on a single Pipeline

    Properties
        alias of depthai.DetectionNetworkProperties

    getAssets (self: depthai.Node) → List[depthai.Asset]
        Retrieves all nodes assets

    getInputs (self: depthai.Node) → List[dai::Node::Input]
        Retrieves all nodes inputs

    getName (self: depthai.Node) → str
        Retrieves nodes name

    getNumInferenceThreads (self: depthai.NeuralNetwork) → int
        How many inference threads will be used to run the network

        Returns Number of threads, 0, 1 or 2. Zero means AUTO

    getOutputs (self: depthai.Node) → List[dai::Node::Output]
        Retrieves all nodes outputs

    property id
        Id of node

    property input
        Input message with data to be infered upon Default queue is blocking with size 5

    property out
        Outputs ImgDetections message that carries parsed detection results.

    property passthrough
        Passthrough message on which the inference was performed.

        Suitable for when input queue is set to non-blocking behavior.

```

setAnchorMasks (*self*: depthai.YoloDetectionNetwork, *anchorMasks*: Dict[str, List[int]]) → None
Set anchor masks

setAnchors (*self*: depthai.YoloDetectionNetwork, *anchors*: List[float]) → None
Set anchors

setBlobPath (*self*: depthai.NeuralNetwork, *path*: str) → None
Load network blob into assets and use once pipeline is started.
Throws if file doesn't exist or isn't a valid network blob.

Parameter path: Path to network blob

setConfidenceThreshold (*self*: depthai.DetectionNetwork, *thresh*: float) → None
Specifies confidence threshold at which to filter the rest of the detections.

Parameter thresh: Detection confidence must be greater than specified threshold to be added to the list

setCoordinateSize (*self*: depthai.YoloDetectionNetwork, *coordinates*: int) → None
Set coordinate size

setIouThreshold (*self*: depthai.YoloDetectionNetwork, *thresh*: float) → None
Set Iou threshold

setNumClasses (*self*: depthai.YoloDetectionNetwork, *numClasses*: int) → None
Set num classes

setNumInferenceThreads (*self*: depthai.NeuralNetwork, *numThreads*: int) → None
How many threads should the node use to run the network.

Parameter numThreads: Number of threads to dedicate to this node

setNumNCEPerInferenceThread (*self*: depthai.NeuralNetwork, *numNCEPerThread*: int) → None
How many Neural Compute Engines should a single thread use for inference

Parameter numNCEPerThread: Number of NCE per thread

setNumPoolFrames (*self*: depthai.NeuralNetwork, *numFrames*: int) → None
Specifies how many frames will be available in the pool

Parameter numFrames: How many frames will pool have

C++

```
class dai::node::YoloDetectionNetwork : public dai::node::DetectionNetwork
    YoloDetectionNetwork node. Parses Yolo results.
```

Public Functions

YoloDetectionNetwork (const std::shared_ptr<PipelineImpl> &par, int64_t nodeId)

void **setNumClasses** (const int numClasses)
Set num classes.

void **setCoordinateSize** (const int coordinates)
Set coordinate size.

void **setAnchors** (std::vector<float> anchors)
Set anchors.

void **setAnchorMasks** (std::map<std::string, std::vector<int>> anchorMasks)
Set anchor masks.

```
void setIouThreshold (float thresh)
    Set Iou threshold.
```

We're always happy to help with code or other questions you might have.

3.6.15 YoloSpatialDetectionNetwork

Spatial detection for the Yolo NN. It is similar to a combination of the *YoloDetectionNetwork* and *SpatialLocationCalculator*.

How to place it

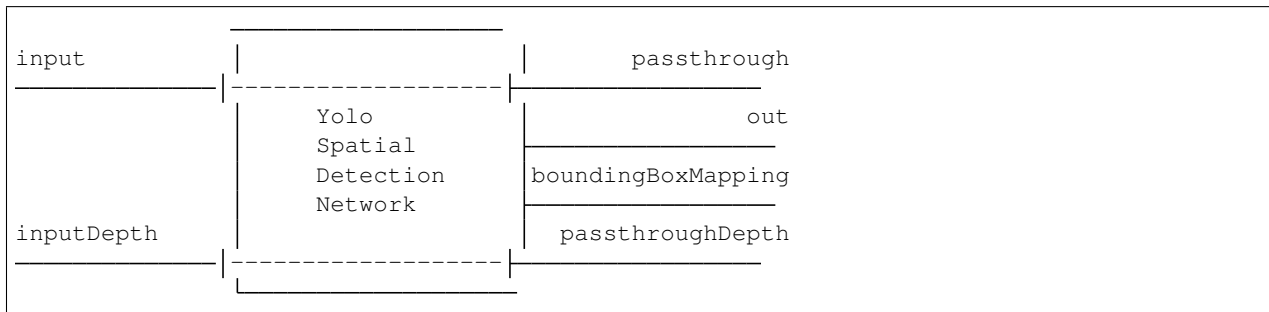
Python

```
pipeline = dai.Pipeline()
yoloSpatial = pipeline.createYoloSpatialDetectionNetwork()
```

C++

```
dai::Pipeline pipeline;
auto yoloSpatial = pipeline.create<dai::node::YoloSpatialDetectionNetwork>();
```

Inputs and Outputs



Message types

- input - *ImgFrame*
- inputDepth - *ImgFrame*
- passthrough - *ImgFrame*
- out - *SpatialImgDetections*
- boundingBoxMapping - *SpatialLocationCalculatorConfig*
- passthroughDepth - *ImgFrame*

Usage

Python

```

pipeline = dai.Pipeline()
yoloSpatial = pipeline.createYoloSpatialDetectionNetwork()
yoloSpatial.setBlobPath(nnBlobPath)

# Spatial detection specific parameters
yoloSpatial.setConfidenceThreshold(0.5)
yoloSpatial.input.setBlocking(False)
yoloSpatial.setBoundingBoxScaleFactor(0.5)
yoloSpatial.setDepthLowerThreshold(100) # Min 10 centimeters
yoloSpatial.setDepthUpperThreshold(5000) # Max 5 meters

# Yolo specific parameters
yoloSpatial.setNumClasses(80)
yoloSpatial.setCoordinateSize(4)
yoloSpatial.setAnchors(np.array([10,14, 23,27, 37,58, 81,82, 135,169, 344,319]))
yoloSpatial.setAnchorMasks({ "side26": np.array([1,2,3]), "side13": np.array([3,4,5])
↪ })
yoloSpatial.setIouThreshold(0.5)

```

C++

```

dai::Pipeline pipeline;
auto yoloSpatial = pipeline.create<dai::node::YoloSpatialDetectionNetwork>();
yoloSpatial->setBlobPath(nnBlobPath);

// Spatial detection specific parameters
yoloSpatial->setConfidenceThreshold(0.5f);
yoloSpatial->input.setBlocking(false);
yoloSpatial->setBoundingBoxScaleFactor(0.5);
yoloSpatial->setDepthLowerThreshold(100); // Min 10 centimeters
yoloSpatial->setDepthUpperThreshold(5000); // Max 5 meters

// yolo specific parameters
yoloSpatial->setNumClasses(80);
yoloSpatial->setCoordinateSize(4);
yoloSpatial->setAnchors({10, 14, 23, 27, 37, 58, 81, 82, 135, 169, 344, 319});
yoloSpatial->setAnchorMasks({{"side13", {3, 4, 5}}, {"side26", {1, 2, 3}}});
yoloSpatial->setIouThreshold(0.5f);

```

Examples of functionality

- [26.3 - RGB & TinyYolo with spatial data](#)

Reference

Python

```
class depthai.YoloSpatialDetectionNetwork
    YoloSpatialDetectionNetwork. (tiny)Yolov3/v4 based network with spatial location data.

    class Connection
        Connection between an Input and Output

    class Id
        Node identificator. Unique for every node on a single Pipeline

    Properties
        alias of depthai.SpatialDetectionNetworkProperties

    property boundingBoxMapping
        Outputs mapping of detected bounding boxes relative to depth map
        Suitable for when displaying remapped bounding boxes on depth frame

    getAssets (self: depthai.Node) → List[depthai.Asset]
        Retrieves all nodes assets

    getInputs (self: depthai.Node) → List[dai::Node::Input]
        Retrieves all nodes inputs

    getName (self: depthai.Node) → str
        Retrieves nodes name

    getNumInferenceThreads (self: depthai.NeuralNetwork) → int
        How many inference threads will be used to run the network

        Returns Number of threads, 0, 1 or 2. Zero means AUTO

    getOutputs (self: depthai.Node) → List[dai::Node::Output]
        Retrieves all nodes outputs

    property id
        Id of node

    property input
        Input message with data to be infered upon Default queue is blocking with size 5

    property inputDepth
        Input message with depth data used to retrieve spatial information about detected object Default queue is
        non-blocking with size 4

    property out
        Outputs ImgDetections message that carries parsed detection results.

    property passthrough
        Passthrough message on which the inference was performed.
        Suitable for when input queue is set to non-blocking behavior.

    property passthroughDepth
        Passthrough message for depth frame on which the spatial location calculation was performed.
```

Suitable for when input queue is set to non-blocking behavior.

setAnchorMasks (*self*: [depthai.YoloSpatialDetectionNetwork](#), *anchorMasks*: *Dict[str, List[int]]*) → *None*
Set anchor masks

setAnchors (*self*: [depthai.YoloSpatialDetectionNetwork](#), *anchors*: *List[float]*) → *None*
Set anchors

setBlobPath (*self*: [depthai.NeuralNetwork](#), *path*: *str*) → *None*
Load network blob into assets and use once pipeline is started.
Throws if file doesn't exist or isn't a valid network blob.

Parameter path: Path to network blob

setBoundingBoxScaleFactor (*self*: [depthai.SpatialDetectionNetwork](#), *scaleFactor*: *float*) → *None*
Specifies scale factor for detected bounding boxes.

Parameter scaleFactor: Scale factor must be in the interval (0,1].

setConfidenceThreshold (*self*: [depthai.DetectionNetwork](#), *thresh*: *float*) → *None*
Specifies confidence threshold at which to filter the rest of the detections.

Parameter thresh: Detection confidence must be greater than specified threshold to be added to the list

setCoordinateSize (*self*: [depthai.YoloSpatialDetectionNetwork](#), *coordinates*: *int*) → *None*
Set coordinate size

setDepthLowerThreshold (*self*: [depthai.SpatialDetectionNetwork](#), *lowerThreshold*: *int*) → *None*
Specifies lower threshold in millimeters for depth values which will used to calculate spatial data

Parameter lowerThreshold: LowerThreshold must be in the interval [0,upperThreshold] and less than upperThreshold.

setDepthUpperThreshold (*self*: [depthai.SpatialDetectionNetwork](#), *upperThreshold*: *int*) → *None*
Specifies upper threshold in millimeters for depth values which will used to calculate spatial data

Parameter upperThreshold: UpperThreshold must be in the interval [lowerThreshold,65535].

setIouThreshold (*self*: [depthai.YoloSpatialDetectionNetwork](#), *thresh*: *float*) → *None*
Set Iou threshold

setNumClasses (*self*: [depthai.YoloSpatialDetectionNetwork](#), *numClasses*: *int*) → *None*
Set num classes

setNumInferenceThreads (*self*: [depthai.NeuralNetwork](#), *numThreads*: *int*) → *None*
How many threads should the node use to run the network.

Parameter numThreads: Number of threads to dedicate to this node

setNumNCEPerInferenceThread (*self*: [depthai.NeuralNetwork](#), *numNCEPerThread*: *int*) → *None*
How many Neural Compute Engines should a single thread use for inference

Parameter numNCEPerThread: Number of NCE per thread

setNumPoolFrames (*self*: [depthai.NeuralNetwork](#), *numFrames*: *int*) → *None*
Specifies how many frames will be available in the pool

Parameter numFrames: How many frames will pool have

C++

```
class dai::node::YoloSpatialDetectionNetwork : public dai::node::SpatialDetectionNetwork
    YoloSpatialDetectionNetwork. (tiny)Yolov3/v4 based network with spatial location data.
```

Public Functions

YoloSpatialDetectionNetwork (**const** std::shared_ptr<*PipelineImpl*> &par, int64_t nodeId)

void **setNumClasses** (**const** int numClasses)
Set num classes.

void **setCoordinateSize** (**const** int coordinates)
Set coordinate size.

void **setAnchors** (std::vector<float> anchors)
Set anchors.

void **setAnchorMasks** (std::map<std::string, std::vector<int>> anchorMasks)
Set anchor masks.

void **setIouThreshold** (float thresh)
Set Iou threshold.

We're always happy to help with code or other questions you might have.

We're always happy to help with code or other questions you might have.

3.7 Messages

Messages are sent between linked *Nodes*. The only way nodes communicate with each other is by sending messages from one to another.

If we have Node1 whose output is linked with Node2's input, a **message** is created in the Node1, sent out of the Node1's output and to the Node2's input.

On the table of contents (left side of the page) all messages are listed under the Messages entry. You can click on them to find out more.

3.7.1 Buffer

Just a good old buffer. All other messages derive from the `Buffer` class.

Reference

Python

class depthai.**Buffer**

Base message - buffer of binary data

getData (self: *object*) → numpy.ndarray[numpy.uint8]

Returns Reference to internal buffer

getRaw (self: depthai.ADatatype) → *depthai.RawBuffer*

setData (*args, **kwargs)

Overloaded function.

1. setData(self: depthai.Buffer, arg0: List[int]) -> None

Parameter data: Copies data to internal buffer

2. setData(self: depthai.Buffer, arg0: numpy.ndarray[numpy.uint8]) -> None

Parameter data: Copies data to internal buffer

C++

class *dai::Buffer* : public *dai::ADatatype*

Base message - buffer of binary data.

Subclassed by *dai::CameraControl*, *dai::ImageManipConfig*, *dai::ImgDetections*,
dai::ImgFrame, *dai::NNData*, *dai::SpatialImgDetections*, *dai::SpatialLocationCalculatorConfig*,
dai::SpatialLocationCalculatorData, *dai::SystemInformation*, *dai::Tracklets*

Public Functions

Buffer ()

Creates *Buffer* message.

Buffer (std::shared_ptr<*dai::RawBuffer*> ptr)

~Buffer () = default

std::vector<std::uint8_t> &**getData** ()

Return Reference to internal buffer

void **setData** (std::vector<std::uint8_t> data)

Parameters

- data: Copies data to internal buffer

Private Functions

std::shared_ptr<*dai::RawBuffer*> **serialize** () const

We're always happy to help with code or other questions you might have.

3.7.2 CameraControl

This message is used for controlling the *color camera* as well as the *mono camera*. The message handles things like capturing still images, configuring auto focus, anti banding, white balance, scenes, effects etc.

Examples of functionality

- *14.1 - Color Camera Control*
- *19 - Mono Camera Control*
- *23 - Auto Exposure on ROI*

Reference

Python

class depthai.CameraControl

CameraControl message Specifies various camera control commands like:

- Still capture
- Auto focus
- Anti banding
- Auto white balance
- Scene
- Effect
- ...

class AntiBandingMode

Members:

OFF
MAINS_50_HZ
MAINS_60_HZ
AUTO

property name

class AutoFocusMode

Members:

OFF
AUTO
MACRO
CONTINUOUS_VIDEO
CONTINUOUS_PICTURE
EDOF

property name

class AutoWhiteBalanceMode

Members:

OFF
AUTO
INCANDESCENT
FLUORESCENT
WARM_FLUORESCENT
DAYLIGHT
CLOUDY_DAYLIGHT
TWILIGHT

SHADE

property name

class EffectMode

Members:

OFF

MONO

NEGATIVE

SOLARIZE

SEPIA

POSTERIZE

WHITEBOARD

BLACKBOARD

AQUA

property name

class SceneMode

Members:

UNSUPPORTED

FACE_PRIORITY

ACTION

PORTRAIT

LANDSCAPE

NIGHT

NIGHT_PORTRAIT

THEATRE

BEACH

SNOW

SUNSET

STEADYPHOTO

FIREWORKS

SPORTS

PARTY

CANDLELIGHT

BARCODE

property name

getCaptureStill (*self*: [depthai.CameraControl](#)) → bool

Check whether command to capture a still is set

Returns True if capture still command is set

getData (*self*: *object*) → numpy.ndarray[numpy.uint8]

Returns Reference to internal buffer

getRaw (*self*: depthai.ADatatype) → *depthai.RawBuffer*

setAntiBandingMode (*self*: depthai.CameraControl, *mode*: depthai.RawCameraControl.AntiBandingMode) → *None*

Set a command to specify auto banding mode

Parameter mode: Auto banding mode to use

setAutoExposureCompensation (*self*: depthai.CameraControl, *compensation*: *int*) → *None*

Set a command to specify auto exposure compenstaion

Parameter compensation: Compensation value between -128..127

setAutoExposureEnable (*self*: depthai.CameraControl) → *None*

Set a command to enable auto exposure

setAutoExposureLock (*self*: depthai.CameraControl, *lock*: *bool*) → *None*

Set a command to specify lock auto exposure

Parameter lock: Auto exposure lock mode enabled or disabled

setAutoExposureRegion (*self*: depthai.CameraControl, *startX*: *int*, *startY*: *int*, *width*: *int*, *height*: *int*) → *None*

Set a command to specify auto exposure region in pixels

Parameter startX: X coordinate of top left corner of region

Parameter startY: Y coordinate of top left corner of region

Parameter width: Region width

Parameter height: Region height

setAutoFocusMode (*self*: depthai.CameraControl, *mode*: depthai.RawCameraControl.AutoFocusMode) → *None*

Set a command to specify autofocus mode

setAutoFocusRegion (*self*: depthai.CameraControl, *startX*: *int*, *startY*: *int*, *width*: *int*, *height*: *int*) → *None*

Set a command to specify focus region in pixels

Parameter startX: X coordinate of top left corner of region

Parameter startY: Y coordinate of top left corner of region

Parameter width: Region width

Parameter height: Region height

setAutoFocusTrigger (*self*: depthai.CameraControl) → *None*

Set a command to trigger autofocus

setAutoWhiteBalanceLock (*self*: depthai.CameraControl, *lock*: *bool*) → *None*

Set a command to specify auto white balance lock

Parameter lock: Auto white balance lock mode enabled or disabled

setAutoWhiteBalanceMode (*self*: depthai.CameraControl, *mode*: depthai.RawCameraControl.AutoWhiteBalanceMode) → *None*

Set a command to specify auto white balance mode

Parameter mode: Auto white balance mode to use

setBrightness (*self*: depthai.CameraControl, *value*: int) → None

Set a command to specify auto white balance lock

Parameter lock: Auto white balance lock mode enabled or disabled

setCaptureStill (*self*: depthai.CameraControl, *capture*: bool) → None

Set a command to capture a still image

setChromaDenoise (*self*: depthai.CameraControl, *value*: int) → None

Set a command to specify chroma denoise value

Parameter value: Chroma denoise

setContrast (*self*: depthai.CameraControl, *value*: int) → None

Set a command to specify auto white balance lock

Parameter lock: Auto white balance lock mode enabled or disabled

setData (*args, **kwargs)

Overloaded function.

1. setData(self: depthai.Buffer, arg0: List[int]) -> None

Parameter data: Copies data to internal buffer

2. setData(self: depthai.Buffer, arg0: numpy.ndarray[numpy.uint8]) -> None

Parameter data: Copies data to internal buffer

setEffectMode (*self*: depthai.CameraControl, *mode*: depthai.RawCameraControl.EffectMode) → None

Set a command to specify effect mode

Parameter mode: Effect mode

setLumaDenoise (*self*: depthai.CameraControl, *value*: int) → None

Set a command to specify luma denoise value

Parameter value: Luma denoise

setManualExposure (*self*: depthai.CameraControl, *exposureTimeUs*: int, *sensitivityIso*: int) → None

Set a command to manually specify exposure

Parameter exposureTimeUs: Exposure time in microseconds

Parameter sensitivityIso: Sensitivity as ISO value

setManualFocus (*self*: depthai.CameraControl, *lensPosition*: int) → None

Set a command to specify manual focus position

Parameter lensPosition: specify lens position 0..255

setNoiseReductionStrength (*self*: depthai.CameraControl, *value*: int) → None

Set a command to specify noise reduction strength

Parameter value: Noise reduction strength

setSaturation (*self*: depthai.CameraControl, *value*: int) → None

Set a command to specify saturation value

Parameter value: Saturation

setSceneMode (*self*: depthai.CameraControl, *mode*: depthai.RawCameraControl.SceneMode) → None

Set a command to specify scene mode

Parameter mode: Scene mode

setSharpness (*self*: [depthai.CameraControl](#), *value*: *int*) → *None*
Set a command to specify sharpness value

Parameter value: Sharpness

setStartStreaming (*self*: [depthai.CameraControl](#)) → *None*
Set a command to start streaming

setStopStreaming (*self*: [depthai.CameraControl](#)) → *None*
Set a command to stop streaming

C++

class [dai::CameraControl](#) : **public** [dai::Buffer](#)
CameraControl message Specifies various camera control commands like:

- Still capture
- Auto focus
- Anti banding
- Auto white balance
- Scene
- Effect
- ...

Public Types

```
using AutoFocusMode = RawCameraControl::AutoFocusMode
using AntiBandingMode = RawCameraControl::AntiBandingMode
using AutoWhiteBalanceMode = RawCameraControl::AutoWhiteBalanceMode
using SceneMode = RawCameraControl::SceneMode
using EffectMode = RawCameraControl::EffectMode
```

Public Functions

```
CameraControl ()
    Construct CameraControl message.

CameraControl (std::shared_ptr<RawCameraControl> ptr)

~CameraControl () = default

void setCaptureStill (bool capture)
    Set a command to capture a still image

void setStartStreaming ()
    Set a command to start streaming

void setStopStreaming ()
    Set a command to stop streaming

void setAutoFocusMode (AutoFocusMode mode)
    Set a command to specify autofocus mode
```

void **setAutoFocusTrigger** ()
Set a command to trigger autofocus

void **setAutoFocusRegion** (uint16_t *startX*, uint16_t *startY*, uint16_t *width*, uint16_t *height*)
Set a command to specify focus region in pixels

Parameters

- *startX*: X coordinate of top left corner of region
- *startY*: Y coordinate of top left corner of region
- *width*: Region width
- *height*: Region height

void **setManualFocus** (uint8_t *lensPosition*)
Set a command to specify manual focus position

Parameters

- *lensPosition*: specify lens position 0..255

void **setAutoExposureEnable** ()
Set a command to enable auto exposure

void **setAutoExposureLock** (bool *lock*)
Set a command to specify lock auto exposure

Parameters

- *lock*: Auto exposure lock mode enabled or disabled

void **setAutoExposureRegion** (uint16_t *startX*, uint16_t *startY*, uint16_t *width*, uint16_t *height*)
Set a command to specify auto exposure region in pixels

Parameters

- *startX*: X coordinate of top left corner of region
- *startY*: Y coordinate of top left corner of region
- *width*: Region width
- *height*: Region height

void **setAutoExposureCompensation** (int8_t *compensation*)
Set a command to specify auto exposure compenstaion

Parameters

- *compensation*: Compensation value between -128..127

void **setAntiBandingMode** (*AntiBandingMode* *mode*)
Set a command to specify auto banding mode

Parameters

- *mode*: Auto banding mode to use

void **setManualExposure** (uint32_t *exposureTimeUs*, uint32_t *sensitivityIso*)
Set a command to manually specify exposure

Parameters

- *exposureTimeUs*: Exposure time in microseconds
- *sensitivityIso*: Sensitivity as ISO value

void **setAutoWhiteBalanceMode** (*AutoWhiteBalanceMode mode*)

Set a command to specify auto white balance mode

Parameters

- mode: Auto white balance mode to use

void **setAutoWhiteBalanceLock** (bool *lock*)

Set a command to specify auto white balance lock

Parameters

- lock: Auto white balance lock mode enabled or disabled

void **setBrightness** (uint16_t *value*)

Set a command to specify auto white balance lock

Parameters

- lock: Auto white balance lock mode enabled or disabled

void **setContrast** (uint16_t *value*)

Set a command to specify auto white balance lock

Parameters

- lock: Auto white balance lock mode enabled or disabled

void **setSaturation** (uint16_t *value*)

Set a command to specify saturation value

Parameters

- value: Saturation

void **setSharpness** (uint16_t *value*)

Set a command to specify sharpness value

Parameters

- value: Sharpness

void **setNoiseReductionStrength** (uint16_t *value*)

Set a command to specify noise reduction strength

Parameters

- value: Noise reduction strength

void **setLumaDenoise** (uint16_t *value*)

Set a command to specify luma denoise value

Parameters

- value: Luma denoise

void **setChromaDenoise** (uint16_t *value*)

Set a command to specify chroma denoise value

Parameters

- value: Chroma denoise

void **setSceneMode** (*SceneMode mode*)

Set a command to specify scene mode

Parameters

- mode: Scene mode

void **setEffectMode** (*EffectMode mode*)
Set a command to specify effect mode

Parameters

- mode: Effect mode

bool **getCaptureStill** () **const**
Check whether command to capture a still is set

Return True if capture still command is set

Private Functions

std::shared_ptr<*RawBuffer*> **serialize** () **const override**

Private Members

RawCameraControl &**cfg**

We're always happy to help with code or other questions you might have.

3.7.3 ImageManipConfig

This message can be used for cropping, warping, rotating, resizing, etc. an image in runtime. It is sent either from the host to *ColorCamera* or *ImageManip*.

Examples of functionality

- 14.1 - Color Camera Control
- 20 - Color Rotate Warp

Reference

Python

```
class depthai.ImageManipConfig  
    ImageManipConfig message. Specifies image manipulation options like:  
        • Crop  
        • Resize  
        • Warp  
        • ...  
  
    getCropXMax (self: depthai.ImageManipConfig) → float  
        Returns Bottom right X coordinate of crop region  
  
    getCropXMin (self: depthai.ImageManipConfig) → float  
        Returns Top left X coordinate of crop region  
  
    getCropYMax (self: depthai.ImageManipConfig) → float
```


Returns Bottom right Y coordinate of crop region

getCropYMin (*self*: [depthai.ImageManipConfig](#)) → float

Returns Top left Y coordinate of crop region

getData (*self*: *object*) → numpy.ndarray[numpy.uint8]

Returns Reference to internal buffer

getRaw (*self*: [depthai.ADatatype](#)) → [depthai.RawBuffer](#)

getResizeHeight (*self*: [depthai.ImageManipConfig](#)) → int

Returns Output image height

getResizeWidth (*self*: [depthai.ImageManipConfig](#)) → int

Returns Output image width

isResizeThumbnail (*self*: [depthai.ImageManipConfig](#)) → bool

Returns True if resize thumbnail mode is set, false otherwise

setCenterCrop (*self*: [depthai.ImageManipConfig](#), *ratio*: float, *whRatio*: float = 1.0) → None
Specifies a centered crop.

Parameter ratio: Ratio between input image and crop region (0..1)

Parameter whRatio: Crop region aspect ratio - 1 equals to square, 1.7 equals to 16:9, ...

ImageManipConfig.setCropRect (*self*: [depthai.ImageManipConfig](#), *xmin*: float, *ymin*: float, ...)
Specifies crop with rectangle with normalized values (0..1)

Parameter xmin: Top left X coordinate of rectangle

Parameter ymin: Top left Y coordinate of rectangle

Parameter xmax: Bottom right X coordinate of rectangle

Parameter ymax: Bottom right Y coordinate of rectangle

setCropRotatedRect (*self*: [depthai.ImageManipConfig](#), *rr*: [depthai.RotatedRect](#), *normalizedCoords*: bool = True) → None
Specifies crop with rotated rectangle. Optionally as non normalized coordinates

Parameter rr: Rotated rectangle which specifies crop

Parameter normalizedCoords: If true coordinates are in normalized range (0..1) otherwise absolute

setData (*args, **kwargs)
Overloaded function.

1. setData(*self*: [depthai.Buffer](#), *arg0*: List[int]) -> None

Parameter data: Copies data to internal buffer

2. setData(*self*: [depthai.Buffer](#), *arg0*: numpy.ndarray[numpy.uint8]) -> None

Parameter data: Copies data to internal buffer

setFrameType (*self*: [depthai.ImageManipConfig](#), *name*: [depthai.RawImgFrame.Type](#)) → None
Specify output frame type.

Parameter name: Frame type

setHorizontalFlip (*self*: [depthai.ImageManipConfig](#), *flip*: bool) → None
Specify horizontal flip

Parameter flip: True to enable flip, false otherwise

setKeepAspectRatio (*self*: [depthai.ImageManipConfig](#), *keep*: *bool*) → *None*
Specifies to whether to keep aspect ratio or not

setResize (*self*: [depthai.ImageManipConfig](#), *w*: *int*, *h*: *int*) → *None*
Specifies output image size. After crop stage the image will be stretched to fit.

Parameter w: Width in pixels

Parameter h: Height in pixels

setResizeThumbnail (*self*: [depthai.ImageManipConfig](#), *w*: *int*, *h*: *int*, *bgRed*: *int* = 0, *bgGreen*: *int* = 0, *bgBlue*: *int* = 0) → *None*
Specifies output image size. After crop stage the image will be resized by preserving aspect ration. Optionally background can be specified.

Parameter w: Width in pixels

Parameter h: Height in pixels

Parameter bgRed: Red component

Parameter bgGreen: Green component

Parameter bgBlue: Blue component

setReusePreviousImage (*self*: [depthai.ImageManipConfig](#), *reuse*: *bool*) → *None*
Instruct ImageManip to not remove current image from its queue and use the same for next message.

Parameter reuse: True to enable reuse, false otherwise

setRotationDegrees (*self*: [depthai.ImageManipConfig](#), *deg*: *float*) → *None*
Specifies clockwise rotation in degrees

Parameter deg: Rotation in degrees

setRotationRadians (*self*: [depthai.ImageManipConfig](#), *rad*: *float*) → *None*
Specifies clockwise rotation in radians

Parameter rad: Rotation in radians

setSkipCurrentImage (*self*: [depthai.ImageManipConfig](#), *skip*: *bool*) → *None*
Instructs ImageManip to skip current image and wait for next in queue.

Parameter skip: True to skip current image, false otherwise

setWarpBorderFillColor (*self*: [depthai.ImageManipConfig](#), *red*: *int*, *green*: *int*, *blue*: *int*) → *None*
Specifies fill color for border pixels. Example:

- `setWarpBorderFillColor(255,255,255)` -> white
- `setWarpBorderFillColor(0,0,255)` -> blue

Parameter red: Red component

Parameter green: Green component

Parameter blue: Blue component

setWarpBorderReplicatePixels (*self*: [depthai.ImageManipConfig](#)) → *None*
Specifies that warp replicates border pixels

setWarpTransformFourPoints (*self*: [depthai.ImageManipConfig](#), *pt*: *List[depthai.Point2f]*, *normalizedCoords*: *bool*) → *None*
Specifies warp by supplying 4 points in either absolute or normalized coordinates

Parameter pt: 4 points specifying warp

Parameter normalizedCoords: If true pt is interpreted as normalized, absolute otherwise

setWarpTransformMatrix3x3 (*self*: [depthai.ImageManipConfig](#), *mat*: *List[float]*) → *None*
Specifies warp with a 3x3 matrix

Parameter mat: 3x3 matrix

C++

class *dai::ImageManipConfig*: **public** *dai::Buffer*
ImageManipConfig message. Specifies image manipulation options like:

- Crop
- Resize
- Warp
- ...

Public Functions

ImageManipConfig()
Construct *ImageManipConfig* message.

ImageManipConfig (std::shared_ptr<*RawImageManipConfig*> *ptr*)

~ImageManipConfig() = default

void **setCropRect** (float *xmin*, float *ymin*, float *xmax*, float *ymax*)
Specifies crop with rectangle with normalized values (0..1)

Parameters

- *xmin*: Top left X coordinate of rectangle
- *ymin*: Top left Y coordinate of rectangle
- *xmax*: Bottom right X coordinate of rectangle
- *ymax*: Bottom right Y coordinate of rectangle

void **setCropRotatedRect** (*RotatedRect* *rr*, bool *normalizedCoords* = true)
Specifies crop with rotated rectangle. Optionally as non normalized coordinates

Parameters

- *rr*: Rotated rectangle which specifies crop
- *normalizedCoords*: If true coordinates are in normalized range (0..1) otherwise absolute

void **setCenterCrop** (float *ratio*, float *whRatio* = 1.0f)
Specifies a centered crop.

Parameters

- *ratio*: Ratio between input image and crop region (0..1)
- *whRatio*: Crop region aspect ratio - 1 equals to square, 1.7 equals to 16:9, ...

void **setWarpTransformFourPoints** (std::vector<*Point2f*> *pt*, bool *normalizedCoords*)
Specifies warp by supplying 4 points in either absolute or normalized coordinates

Parameters

- `pt`: 4 points specifying warp
- `normalizedCoords`: If true `pt` is interpreted as normalized, absolute otherwise

void **setWarpTransformMatrix3x3** (std::vector<float> *mat*)
Specifies warp with a 3x3 matrix

Parameters

- `mat`: 3x3 matrix

void **setWarpBorderReplicatePixels** ()
Specifies that warp replicates border pixels

void **setWarpBorderFillColor** (int *red*, int *green*, int *blue*)
Specifies fill color for border pixels. Example:

- `setWarpBorderFillColor(255,255,255)` -> white
- `setWarpBorderFillColor(0,0,255)` -> blue

Parameters

- `red`: Red component
- `green`: Green component
- `blue`: Blue component

void **setRotationDegrees** (float *deg*)
Specifies clockwise rotation in degrees

Parameters

- `deg`: Rotation in degrees

void **setRotationRadians** (float *rad*)
Specifies clockwise rotation in radians

Parameters

- `rad`: Rotation in radians

void **setResize** (int *w*, int *h*)
Specifies output image size. After crop stage the image will be stretched to fit.

Parameters

- `w`: Width in pixels
- `h`: Height in pixels

void **setResizeThumbnail** (int *w*, int *h*, int *bgRed* = 0, int *bgGreen* = 0, int *bgBlue* = 0)
Specifies output image size. After crop stage the image will be resized by preserving aspect ration. Optionally background can be specified.

Parameters

- `w`: Width in pixels
- `h`: Height in pixels
- `bgRed`: Red component
- `bgGreen`: Green component

- `bgBlue`: Blue component

void **setFrameType** (*ImgFrame::Type name*)
Specify output frame type.

Parameters

- `name`: Frame type

void **setHorizontalFlip** (bool *flip*)
Specify horizontal flip

Parameters

- `flip`: True to enable flip, false otherwise

void **setReusePreviousImage** (bool *reuse*)
Instruct ImageManip to not remove current image from its queue and use the same for next message.

Parameters

- `reuse`: True to enable reuse, false otherwise

void **setSkipCurrentImage** (bool *skip*)
Instructs ImageManip to skip current image and wait for next in queue.

Parameters

- `skip`: True to skip current image, false otherwise

void **setKeepAspectRatio** (bool *keep*)
Specifies to whether to keep aspect ratio or not

float **getCropXMin** () **const**

Return Top left X coordinate of crop region

float **getCropYMin** () **const**

Return Top left Y coordinate of crop region

float **getCropXMax** () **const**

Return Bottom right X coordinate of crop region

float **getCropYMax** () **const**

Return Bottom right Y coordinate of crop region

int **getResizeWidth** () **const**

Return Output image width

int **getResizeHeight** () **const**

Return Output image height

bool **isResizeThumbnail** () **const**

Return True if resize thumbnail mode is set, false otherwise

Private Functions

```
std::shared_ptr<RawBuffer> serialize () const override
```

Private Members

```
RawImageManipConfig &cfg
```

We're always happy to help with code or other questions you might have.

3.7.4 ImgDetections

Both *YoloDetectionNetwork* and *MobileNetDetectionNetwork* output this message. This message contains a list of detections, which contains label, confidence, and the bounding box information (xmin, ymin, xmax, ymax).

Examples of functionality

- *08 - RGB & MobilenetSSD*
- *09 - Mono & MobilenetSSD*
- *22.2 - RGB & TinyYoloV4 decoding on device*

Reference

Python

```
class depthai.ImgDetections
```

ImgDetections message. Carries normalized detection results

```
property detections
```

Detections

```
getData (self: object) → numpy.ndarray[numpy.uint8]
```

Returns Reference to internal buffer

```
getRaw (self: depthai.ADatatype) → depthai.RawBuffer
```

```
setData (*args, **kwargs)
```

Overloaded function.

1. setData(self: depthai.Buffer, arg0: List[int]) -> None

Parameter data: Copies data to internal buffer

2. setData(self: depthai.Buffer, arg0: numpy.ndarray[numpy.uint8]) -> None

Parameter data: Copies data to internal buffer

C++

```
class dai::ImgDetections : public dai::Buffer
```

ImgDetections message. Carries normalized detection results

Public Functions

ImgDetections ()
Construct *ImgDetections* message.

ImgDetections (std::shared_ptr<*RawImgDetections*> ptr)
~**ImgDetections** () = default

Public Members

std::vector<*ImgDetection*> &**detections**
Detections.

Private Functions

std::shared_ptr<*RawBuffer*> **serialize** () **const override**

Private Members

RawImgDetections &**dets**

We're always happy to help with code or other questions you might have.

3.7.5 ImgFrame

These are all the images (regardless of their encoding/format), as well as the depth/disparity “image”. *ColorCamera* and *MonoCamera* are the source of the image frame messages.

Examples of functionality

- *01 - RGB Preview*
- *02 - Mono Preview*
- *03 - Depth Preview*

Reference

Python

```
class depthai.ImgFrame
    ImgFrame message. Carries image data and metadata.

    class Type
        Members:
            YUV422i
            YUV444p
            YUV420p
            YUV422p
            YUV400p
```

RGBA8888
RGB161616
RGB888p
BGR888p
RGB888i
BGR888i
RGBF16F16F16p
BGRF16F16F16p
RGBF16F16F16i
BGRF16F16F16i
GRAY8
GRAYF16
LUT2
LUT4
LUT16
RAW16
RAW14
RAW12
RAW10
RAW8
PACK10
PACK12
YUV444i
NV12
NV21
BITSTREAM
HDR
NONE

property name

getCategory (*self*: [depthai.ImgFrame](#)) → int

Retrieves image category

getCvFrame (*self*: [object](#)) → [object](#)

Returns BGR or grayscale frame compatible with use in other opencv functions

getData (*self*: [object](#)) → [numpy.ndarray\[numpy.uint8\]](#)

Returns Reference to internal buffer

getFrame (*self*: [object](#), *copy*: [bool](#) = *False*) → [numpy.ndarray](#)

Returns numpy array with shape as specified by width, height and type

getHeight (*self*: `depthai.ImgFrame`) → `int`
Retrieves image height in pixels

getInstanceNum (*self*: `depthai.ImgFrame`) → `int`
Retrieves instance number

getRaw (*self*: `depthai.ADatatype`) → `depthai.RawBuffer`

getSequenceNum (*self*: `depthai.ImgFrame`) → `int`
Retrieves image sequence number

getTimestamp (*self*: `depthai.ImgFrame`) → `datetime.timedelta`
Retrieves image timestamp related to `steady_clock` / `time.monotonic`

getType (*self*: `depthai.ImgFrame`) → `depthai.RawImgFrame.Type`
Retrieves image type

getWidth (*self*: `depthai.ImgFrame`) → `int`
Retrieves image width in pixels

setCategory (*self*: `depthai.ImgFrame`, *category*: `int`) → `None`
Parameter category: Image category

setData (**args*, ***kwargs*)
Overloaded function.

1. `setData(self: depthai.Buffer, arg0: List[int]) -> None`

Parameter data: Copies data to internal buffer

2. `setData(self: depthai.Buffer, arg0: numpy.ndarray[numpy.uint8]) -> None`

Parameter data: Copies data to internal buffer

setFrame (*self*: `depthai.ImgFrame`, *array*: `numpy.ndarray`) → `None`
Copies array bytes to `ImgFrame` buffer

setHeight (*self*: `depthai.ImgFrame`, *height*: `int`) → `None`
Specifies frame height

Parameter width: frame height

setInstanceNum (*self*: `depthai.ImgFrame`, *instance*: `int`) → `None`
Instance number relates to the origin of the frame (which camera)

Parameter instance: Instance number

setSequenceNum (*self*: `depthai.ImgFrame`, *seq*: `int`) → `None`
Specifies sequence number

Parameter seq: Sequence number

setTimestamp (*self*: `depthai.ImgFrame`, *timestamp*: `datetime.timedelta`) → `None`
Specifies current timestamp, related to `steady_clock` / `time.monotonic`

setType (*self*: `depthai.ImgFrame`, *type*: `depthai.RawImgFrame.Type`) → `None`
Specifies frame type, RGB, BGR, ...

Parameter type: Type of image

setWidth (*self*: `depthai.ImgFrame`, *width*: `int`) → `None`
Specifies frame width

Parameter width: frame width

C++

```
class dai::ImgFrame : public dai::Buffer
    ImgFrame message. Carries image data and metadata.
```

Public Types

```
using Type = RawImgFrame::Type
using Specs = RawImgFrame::Specs
```

Public Functions

```
ImgFrame ()
    Construct ImgFrame message. Timestamp is set to now

ImgFrame (std::shared_ptr<RawImgFrame> ptr)

~ImgFrame () = default

std::chrono::time_point<std::chrono::steady_clock, std::chrono::steady_clock::duration> getTimestamp () const
    Retrieves image timestamp related to steady_clock / time.monotonic

unsigned int getInstanceNum () const
    Retrieves instance number

unsigned int getCategory () const
    Retrieves image category

unsigned int getSequenceNum () const
    Retrieves image sequence number

unsigned int getWidth () const
    Retrieves image width in pixels

unsigned int getHeight () const
    Retrieves image height in pixels

Type getType () const
    Retrieves image type

void setTimestamp (std::chrono::time_point<std::chrono::steady_clock,
    std::chrono::steady_clock::duration> timestamp)
    Specifies current timestamp, related to steady_clock / time.monotonic

void setInstanceNum (unsigned int instance)
    Instance number relates to the origin of the frame (which camera)
```

Parameters

- *instance*: Instance number

```
void setCategory (unsigned int category)
```

Parameters

- *category*: Image category

```
void setSequenceNum (unsigned int seq)
    Specifies sequence number
```

Parameters

- `seq`: Sequence number

void **setWidth** (unsigned int *width*)
Specifies frame width

Parameters

- `width`: frame width

void **setHeight** (unsigned int)
Specifies frame height

Parameters

- `width`: frame height

void **setType** (*Type type*)
Specifies frame type, RGB, BGR, ...

Parameters

- `type`: Type of image

void **setFrame** (cv::Mat *frame*)
Copies cv::Mat data to *ImgFrame* buffer
Note This API only available if OpenCV support enabled

Parameters

- `frame`: Input cv::Mat frame from which to copy the data

cv::Mat **getFrame** (bool *copy* = false)
Retrieves data as cv::Mat with specified width, height and type
Note This API only available if OpenCV support enabled

Return cv::Mat with corresponding to *ImgFrame* parameters

Parameters

- `copy`: If false only a reference to data is made, otherwise a copy

cv::Mat **getCvFrame** ()
Retrieves cv::Mat suitable for use in common opencv functions. *ImgFrame* is converted to color BGR interleaved or grayscale depending on type.
Note This API only available if OpenCV support enabled
A copy is always made
Return cv::Mat for use in opencv functions

Private Functions

```
std::shared_ptr<RawBuffer> serialize () const override
```

Private Members

```
RawImgFrame &img
```

We're always happy to help with code or other questions you might have.

3.7.6 NNData

This message carries tensors and its data. You can receive this message from the *NeuralNetwork* node or you could create this message on the host, populate the tensor with the data and send the message to the input of the *NeuralNetwork* node.

Reference

Python

```
class depthai.NNData
```

NNData message. Carries tensors and their metadata

```
getAllLayerNames (self: depthai.NNData) → List[str]
```

Returns Names of all layers added

```
getAllLayers (self: depthai.NNData) → List[depthai.TensorInfo]
```

Returns All layers and their information

```
getData (self: object) → numpy.ndarray[numpy.uint8]
```

Returns Reference to internal buffer

```
getFirstLayerFp16 (self: depthai.NNData) → List[float]
```

Convenience function to retrieve float values from first layers FP16 tensor

Returns Float data

```
getFirstLayerInt32 (self: depthai.NNData) → List[int]
```

Convenience function to retrieve INT32 values from first layers tensor

Returns INT32 data

```
getFirstLayerUInt8 (self: depthai.NNData) → List[int]
```

Convenience function to retrieve U8 data from first layer

Returns U8 binary data

```
getLayer (self: depthai.NNData, name: str, tensor: depthai.TensorInfo) → bool
```

Retrieve layers tensor information

Parameter name: Name of the layer

Parameter tensor: Outputs tensor information of that layer

Returns True if layer exists, false otherwise

getLayerDatatype (*self*: depthai.NNData, *name*: str, *datatype*: depthai.TensorInfo.DataType) → bool
 Retrieve datatype of a layers tensor

Parameter name: Name of the layer

Parameter datatype: Datatype of layers tensor

Returns True if layer exists, false otherwise

getLayerFp16 (*self*: depthai.NNData, *name*: str) → List[float]
 Convenience function to retrieve float values from layers FP16 tensor

Parameter name: Name of the layer

Returns Float data

getLayerInt32 (*self*: depthai.NNData, *name*: str) → List[int]
 Convenience function to retrieve INT32 values from layers tensor

Parameter name: Name of the layer

Returns INT32 data

getLayerUInt8 (*self*: depthai.NNData, *name*: str) → List[int]
 Convenience function to retrieve U8 data from layer

Parameter name: Name of the layer

Returns U8 binary data

getRaw (*self*: depthai.ADatatype) → depthai.RawBuffer

hasLayer (*self*: depthai.NNData, *name*: str) → bool
 Checks if given layer exists

Parameter name: Name of the layer

Returns True if layer exists, false otherwise

setData (*args, **kwargs)
 Overloaded function.

1. setData(*self*: depthai.Buffer, *arg0*: List[int]) -> None

Parameter data: Copies data to internal buffer

2. setData(*self*: depthai.Buffer, *arg0*: numpy.ndarray[numpy.uint8]) -> None

Parameter data: Copies data to internal buffer

setLayer (*args, **kwargs)
 Overloaded function.

1. setLayer(*self*: depthai.NNData, *name*: str, *data*: numpy.ndarray[numpy.uint8]) -> None

Set a layer with datatype U8.

Parameter name: Name of the layer

Parameter data: Data to store

2. setLayer(*self*: depthai.NNData, *name*: str, *data*: List[int]) -> None

Set a layer with datatype U8. Integers are casted to bytes.

Parameter name: Name of the layer

Parameter data: Data to store

3. `setLayer(self: depthai.NNData, name: str, data: List[float]) -> None`

Set a layer with datatype FP16. Float values are converted to FP16.

Parameter name: Name of the layer

Parameter data: Data to store

4. `setLayer(self: depthai.NNData, name: str, data: List[float]) -> None`

Set a layer with datatype FP16. Double values are converted to FP16.

Parameter name: Name of the layer

Parameter data: Data to store

C++

```
class dai::NNData: public dai::Buffer
    NNData message. Carries tensors and their metadata
```

Public Functions

NNData()

Construct *NNData* message.

NNData(std::shared_ptr<*RawNNData*> ptr)

~NNData() = default

void **setLayer**(const std::string &name, std::vector<std::uint8_t> data)

Set a layer with datatype U8.

Parameters

- name: Name of the layer
- data: Data to store

void **setLayer**(const std::string &name, const std::vector<int> &data)

Set a layer with datatype U8. Integers are casted to bytes.

Parameters

- name: Name of the layer
- data: Data to store

void **setLayer**(const std::string &name, std::vector<float> data)

Set a layer with datatype FP16. Float values are converted to FP16.

Parameters

- name: Name of the layer
- data: Data to store

void **setLayer**(const std::string &name, std::vector<double> data)

Set a layer with datatype FP16. Double values are converted to FP16.

Parameters

- name: Name of the layer
- data: Data to store

std::vector<std::string> **getAllLayerNames** () **const**

Return Names of all layers added

std::vector<*TensorInfo*> **getAllLayers** () **const**

Return All layers and their information

bool **getLayer** (**const** std::string &name, *TensorInfo* &tensor) **const**

Retrieve layers tensor information

Return True if layer exists, false otherwise

Parameters

- name: Name of the layer
- [out] tensor: Outputs tensor information of that layer

bool **hasLayer** (**const** std::string &name) **const**

Checks if given layer exists

Return True if layer exists, false otherwise

Parameters

- name: Name of the layer

bool **getLayerDatatype** (**const** std::string &name, *TensorInfo*::DataType &datatype) **const**

Retrieve datatype of a layers tensor

Return True if layer exists, false otherwise

Parameters

- name: Name of the layer
- [out] datatype: Datatype of layers tensor

std::vector<std::uint8_t> **getLayerUInt8** (**const** std::string &name) **const**

Convenience function to retrieve U8 data from layer

Return U8 binary data

Parameters

- name: Name of the layer

std::vector<float> **getLayerFp16** (**const** std::string &name) **const**

Convenience function to retrieve float values from layers FP16 tensor

Return Float data

Parameters

- name: Name of the layer

std::vector<std::int32_t> **getLayerInt32** (**const** std::string &name) **const**

Convenience function to retrieve INT32 values from layers tensor

Return INT32 data

Parameters

- `name`: Name of the layer

`std::vector<std::uint8_t> getFirstLayerUInt8 () const`

Convenience function to retrieve U8 data from first layer

Return U8 binary data

`std::vector<float> getFirstLayerFp16 () const`

Convenience function to retrieve float values from first layers FP16 tensor

Return Float data

`std::vector<std::int32_t> getFirstLayerInt32 () const`

Convenience function to retrieve INT32 values from first layers tensor

Return INT32 data

Private Functions

`std::shared_ptr<RawBuffer> serialize () const`

Private Members

RawNNData &`rawNn`

`std::unordered_map<std::string, std::vector<std::uint8_t>>` `u8Data`

`std::unordered_map<std::string, std::vector<std::uint16_t>>` `fp16Data`

Private Static Attributes

`constexpr int DATA_ALIGNMENT = 64`

We're always happy to help with code or other questions you might have.

3.7.7 SpatialImgDetections

Similar to *ImgDetections*, but this message includes **XYZ** coordinates of the detected objects as well.

Both *YoloSpatialDetectionNetwork* and *MobileNetSpatialDetectionNetwork* output this message.

Examples of functionality

- 26.1 - RGB & MobilenetSSD with spatial data
- 26.2 - Mono & MobilenetSSD with spatial data
- 26.3 - RGB & TinyYolo with spatial data

Reference

Python

```
class depthai.SpatialImgDetections
    SpatialImgDetections message. Carries detection results together with spatial location data

    getData (self: object) → numpy.ndarray[numpy.uint8]

        Returns Reference to internal buffer

    getRaw (self: depthai.ADatatype) → depthai.RawBuffer

    setData (*args, **kwargs)
        Overloaded function.

        1. setData(self: depthai.Buffer, arg0: List[int]) -> None

        Parameter data: Copies data to internal buffer

        2. setData(self: depthai.Buffer, arg0: numpy.ndarray[numpy.uint8]) -> None

        Parameter data: Copies data to internal buffer
```

C++

```
class dai::SpatialImgDetections : public dai::Buffer
    SpatialImgDetections message. Carries detection results together with spatial location data
```

Public Functions

```
SpatialImgDetections ()
    Construct SpatialImgDetections message.

SpatialImgDetections (std::shared_ptr<RawSpatialImgDetections> ptr)

~SpatialImgDetections () = default
```

Public Members

```
std::vector<SpatialImgDetection> &detections
    Detection results.
```

Private Functions

```
std::shared_ptr<RawBuffer> serialize () const override
```

Private Members

RawSpatialImgDetections & **dets**

We're always happy to help with code or other questions you might have.

3.7.8 SpatialLocationCalculatorConfig

This message is used to configure the *SpatialLocationCalculator* node.

Examples of functionality

- 27 - *Spatial location calculator*

Reference

Python

```
class depthai.SpatialLocationCalculatorConfig
    SpatialLocationCalculatorConfig message. Carries ROI (region of interest) and threshold for depth calculation

    addROI (self: depthai.SpatialLocationCalculatorConfig, ROI: depthai.SpatialLocationCalculatorConfigData)
        → None
        Add a new ROI to configuration data.

        Parameter roi: Configuration parameters for ROI (region of interest)

    getConfigData (self: depthai.SpatialLocationCalculatorConfig) → List[depthai.SpatialLocationCalculatorConfigData]
        Retrieve configuration data for SpatialLocationCalculator

        Returns Vector of configuration parameters for ROIs (region of interests)

    getData (self: object) → numpy.ndarray[numpy.uint8]

        Returns Reference to internal buffer

    getRaw (self: depthai.ADatatype) → depthai.RawBuffer

    setData (*args, **kwargs)
        Overloaded function.

        1. setData(self: depthai.Buffer, arg0: List[int]) -> None

        Parameter data: Copies data to internal buffer

        2. setData(self: depthai.Buffer, arg0: numpy.ndarray[numpy.uint8]) -> None

        Parameter data: Copies data to internal buffer

    setROIs (self: depthai.SpatialLocationCalculatorConfig, ROIs: List[depthai.SpatialLocationCalculatorConfigData])
        → None
        Set a vector of ROIs as configuration data.

        Parameter ROIs: Vector of configuration parameters for ROIs (region of interests)
```

C++

```
class dai::SpatialLocationCalculatorConfig : public dai::Buffer
    SpatialLocationCalculatorConfig message. Carries ROI (region of interest) and threshold for depth calculation
```

Public Functions

SpatialLocationCalculatorConfig()

Construct *SpatialLocationCalculatorConfig* message.

SpatialLocationCalculatorConfig (std::shared_ptr<*RawSpatialLocationCalculatorConfig*>
ptr)

~SpatialLocationCalculatorConfig() = default

void **setROIs** (std::vector<*SpatialLocationCalculatorConfigData*> ROIs)

Set a vector of ROIs as configuration data.

Parameters

- ROIs: Vector of configuration parameters for ROIs (region of interests)

void **addROI** (*SpatialLocationCalculatorConfigData* &ROI)

Add a new ROI to configuration data.

Parameters

- roi: Configuration parameters for ROI (region of interest)

std::vector<*SpatialLocationCalculatorConfigData*> **getConfigData()** const

Retrieve configuration data for SpatialLocationCalculator

Return Vector of configuration parameters for ROIs (region of interests)

Private Functions

std::shared_ptr<*RawBuffer*> **serialize()** const override

Private Members

RawSpatialLocationCalculatorConfig &cfg

We're always happy to help with code or other questions you might have.

3.7.9 SpatialLocationCalculatorData

This message is an output from the *SpatialLocationCalculator* node.

Examples of functionality

- 27 - *Spatial location calculator*

Reference

Python

```
class depthai.SpatialLocationCalculatorData
    SpatialLocationCalculatorData message. Carries spatial information (X,Y,Z) and their configuration parameters

    getData (self: object) → numpy.ndarray[numpy.uint8]
        Returns Reference to internal buffer

    getRaw (self: depthai.ADatatype) → depthai.RawBuffer

    getSpatialLocations (self: depthai.SpatialLocationCalculatorData) →
        List[depthai.SpatialLocations]
        Retrieve configuration data for SpatialLocationCalculatorData.

        Returns Vector of spatial location data, carrying spatial information (X,Y,Z)

    setData (*args, **kwargs)
        Overloaded function.

        1. setData(self: depthai.Buffer, arg0: List[int]) -> None

        Parameter data: Copies data to internal buffer

        2. setData(self: depthai.Buffer, arg0: numpy.ndarray[numpy.uint8]) -> None

        Parameter data: Copies data to internal buffer
```

C++

```
class dai::SpatialLocationCalculatorData : public dai::Buffer
    SpatialLocationCalculatorData message. Carries spatial information (X,Y,Z) and their configuration parameters
```

Public Functions

```
SpatialLocationCalculatorData ()
    Construct SpatialLocationCalculatorData message.

SpatialLocationCalculatorData (std::shared_ptr<RawSpatialLocations> ptr)

~SpatialLocationCalculatorData () = default

std::vector<SpatialLocations> &getSpatialLocations () const
    Retrieve configuration data for SpatialLocationCalculatorData.

Return Vector of spatial location data, carrying spatial information (X,Y,Z)
```

Public Members

`std::vector<SpatialLocations> &spatialLocations`

Private Functions

`std::shared_ptr<RawBuffer> serialize () const override`

Private Members

`RawSpatialLocations &rawdata`

We're always happy to help with code or other questions you might have.

3.7.10 SystemInformation

System information message is created by the *SystemLogger* node.

Examples of functionality

- 25 - *System information*

Reference

Python

class `depthai.SystemInformation`

SystemInformation message. Carries memory usage, cpu usage and chip temperatures.

getData (*self*: *object*) → `numpy.ndarray[numpy.uint8]`

Returns Reference to internal buffer

getRaw (*self*: `depthai.ADatatype`) → `depthai.RawBuffer`

setData (**args*, ***kwargs*)

Overloaded function.

1. `setData(self: depthai.Buffer, arg0: List[int]) -> None`

Parameter data: Copies data to internal buffer

2. `setData(self: depthai.Buffer, arg0: numpy.ndarray[numpy.uint8]) -> None`

Parameter data: Copies data to internal buffer

C++

class `dai::SystemInformation: public dai::Buffer`

SystemInformation message. Carries memory usage, cpu usage and chip temperatures.

Public Functions

SystemInformation()

Construct *SystemInformation* message.

SystemInformation(std::shared_ptr<*RawSystemInformation*> ptr)

~SystemInformation() = default

Public Members

MemoryInfo &ddrMemoryUsage

MemoryInfo &cmxMemoryUsage

MemoryInfo &leonCssMemoryUsage

MemoryInfo &leonMssMemoryUsage

CpuUsage &leonCssCpuUsage

CpuUsage &leonMssCpuUsage

ChipTemperature &chipTemperature

Private Functions

std::shared_ptr<*RawBuffer*> **serialize**() const override

Private Members

RawSystemInformation &**systemInformation**

We're always happy to help with code or other questions you might have.

We're always happy to help with code or other questions you might have.

3.8 Hello World

Learn how to use the DepthAI Python API to display a color video stream.

3.8.1 Demo

3.8.2 Dependencies

Let's get your development environment setup first. This tutorial uses:

- Python 3.6 (Ubuntu) or Python 3.7 (Raspbian).
- The DepthAI *Python API*
- The `cv2` and `numpy` Python modules.

3.8.3 Code Overview

The `depthai` Python module provides access to your board's 4K 60 Hz color camera. We'll display a video stream from this camera to your desktop. You can find the [complete source code for this tutorial on GitHub](#).

3.8.4 File Setup

Setup the following file structure on your computer:

```
cd ~
mkdir -p depthai-tutorials-practice/1-hello-world
touch depthai-tutorials-practice/1-hello-world/hello_world.py
cd depthai-tutorials-practice/1-hello-world
```

What's with the `-practice` suffix in parent directory name? Our tutorials are available on GitHub via the [depthai-tutorials](#) repository. We're appending `-practice` so you can distinguish between your work and our finished tutorials (should you choose to download those).

3.8.5 Install pip dependencies

To display the DepthAI color video stream we need to import a small number of packages. Download and install the requirements for this tutorial:

```
python3 -m pip install numpy opencv-python depthai --user
```

3.8.6 Test your environment

Let's verify we're able to load all of our dependencies. Open the `hello_world.py` file you *created earlier* in your code editor. Copy and paste the following into `hello_world.py`:

```
import numpy as np # numpy - manipulate the packet data returned by depthai
import cv2 # opencv - display the video stream
import depthai # access the camera and its data packets
```

Try running the script and ensure it executes without error:

```
python3 hello_world.py
```

If you see the following error:

```
ModuleNotFoundError: No module named 'depthai'
```

...follow [these steps in our troubleshooting section](#).

3.8.7 Define a pipeline

Any action from DepthAI, whether it's a neural inference or color camera output, require a **pipeline** to be defined, including nodes and connections corresponding to our needs.

In this case, we want to see the frames from **color camera**, as well as a simple **neural network** to be ran on top of them.

Let's start off with an empty Pipeline object

```
pipeline = depthai.Pipeline()
```

Now, first node we will add is a ColorCamera. We will use the preview output, resized to 300x300 to fit the **mobilenet-ssd input size** (which we will define later)

```
cam_rgb = pipeline.createColorCamera()
cam_rgb.setPreviewSize(300, 300)
cam_rgb.setInterleaved(False)
```

Up next, let's define a NeuralNetwork node with **mobilenet-ssd network**. The blob file for this example can be found [here](#)

```
detection_nn = pipeline.createNeuralNetwork()
detection_nn.setBlobPath("/path/to/mobilenet-ssd.blob")
```

And now, let's connect a color camera preview output to neural network input

```
cam_rgb.preview.link(detection_nn.input)
```

Finally, we want to receive both color camera frames and neural network inference results - as these are produced on the device, they need to be transported to our machine (host). The communication between device and host is handled by XLink, and in our case, since we want to receive data from device to host, we will use XLinkOut node

```
xout_rgb = pipeline.createXLinkOut()
xout_rgb.setStreamName("rgb")
cam_rgb.preview.link(xout_rgb.input)

xout_nn = pipeline.createXLinkOut()
xout_nn.setStreamName("nn")
detection_nn.out.link(xout_nn.input)
```

3.8.8 Initialize the DepthAI Device

Having the pipeline defined, we can now initialize a device and start it

```
device = depthai.Device(pipeline)
device.startPipeline()
```

Note: By default, the DepthAI is accessed as a USB3 device. This comes with [several limitations](#).

If you'd like to communicate via USB2, being free from these but having a limited bandwidth, initialize the DepthAI with the following code

```
device = depthai.Device(pipeline, True)
```


From this point on, the pipeline will be running on the device, producing results we requested. Let's grab them

3.8.9 Adding helpers

As XLinkOut nodes has been defined in the pipeline, we'll define now a host side output queues to access the produced results

```
q_rgb = device.getOutputQueue("rgb")
q_nn = device.getOutputQueue("nn")
```

These will fill up with results, so next thing to do is consume the results. We will need two placeholders - one for rgb frame and one for nn results

```
frame = None
bboxes = []
```

Also, due to neural network implementation details, bounding box coordinates in inference results are represented as floats from $<0..1>$ range - so relative to frame width/height (e.g. if image has 200px width and nn returned `x_min` coordinate equal to 0.2, this means the actual (normalised) `x_min` coordinate is 40px).

That's why we need to define a helper function, `frame_norm`, that will convert these $<0..1>$ values into actual pixel positions

```
def frame_norm(frame, bbox):
    return (np.array(bbox) * np.array([*frame.shape[:2], *frame.shape[:2]])[::-1]) .
    ↳ astype(int)
```

3.8.10 Consuming the results

Having everything prepared, we are ready to start out main program loop

```
while True:
    # ...
```

Now, inside this loop, first thing to do is fetching latest results from both nn node and color camera

```
in_rgb = q_rgb.tryGet()
in_nn = q_nn.tryGet()
```

The `tryGet` method returns either the latest result or `None` if the queue is empty.

Results, both from rgb camera or neural network, will be delivered as 1D arrays, so both of them will require transformations to be useful for display (we have already defined one of the transformations needed - the `frame_norm` function)

First up, if we receive a frame from rgb camera, we need to convert it from 1D array into HWC form (HWC stands for Height Width Channels, so 3D array, with first dimension being width, second height, and third the color channel)

```
if in_rgb is not None:
    shape = (3, in_rgb.getHeight(), in_rgb.getWidth())
    frame = in_rgb.getData().reshape(shape).transpose(1, 2, 0).astype(np.uint8)
    frame = np.ascontiguousarray(frame)
```

Second, the neural network results will also need transformations. These are also returned as a 1D array, but this time the array has a fixed size (constant, no matter how many results the neural network has actually produced). Actual results in array are followed with `-1` and then filled to meet the fixed size with `0`. One results has 7 fields, each being respectively `image_id`, `label`, `confidence`, `x_min`, `y_min`, `x_max`, `y_max`. We will want only the last four values (being the bounding box), but we'll also filter out the ones which confidence is below a certain threshold - it can be anywhere between `<0.1>`, and for this example we will use `0.8` threshold

```
if in_nn is not None:
    bboxes = np.array(in_nn.getFirstLayerFp16())
    bboxes = bboxes[:np.where(bboxes == -1)[0][0]]
    bboxes = bboxes.reshape((bboxes.size // 7, 7))
    bboxes = bboxes[bboxes[:, 2] > 0.8][:, 3:7]
```

To better understand this flow, let's take an example. Let's assume the `np.array(in_nn.getFirstLayerFp16())` returns the following array

```
[0, 15, 0.99023438, 0.45556641, 0.34399414 0.88037109, 0.9921875, 0, 15, 0.98828125,
↪0.03076172, 0.23388672, 0.60205078, 1.0078125, -1, 0, 0, 0, ...]
```

First operation, `bboxes[:np.where(bboxes == -1)[0][0]]`, removes the trailing zeros from the array, so now the bbox array will look like this

```
[0, 15, 0.99023438, 0.45556641, 0.34399414 0.88037109, 0.9921875, 0, 15, 0.98828125,
↪0.03076172, 0.23388672, 0.60205078, 1.0078125]
```

Second one - `bboxes.reshape((bboxes.size // 7, 7))`, reshapes the 1D array into 2D array - where each row is a separate result

```
[
  [0, 15, 0.99023438, 0.45556641, 0.34399414 0.88037109, 0.9921875],
  [0, 15, 0.98828125, 0.03076172, 0.23388672, 0.60205078, 1.0078125]
]
```

Last one - `bboxes = bboxes[bboxes[:, 2] > 0.8][:, 3:7]` - will filter the results based on the confidence column (3rd one, with index 2) to be above a defined threshold (`0.8`) - and from these results, it will only take the last 4 columns being the bounding boxes. Since both our results have a very high confidence (`0.99023438` and `0.98828125` respectively), they won't be filtered, and the final array will look like this

```
[
  [0.45556641, 0.34399414 0.88037109, 0.9921875],
  [0.03076172, 0.23388672, 0.60205078, 1.0078125]
]
```

3.8.11 Display the results

Up to this point, we have all our results consumed from the DepthAI device, and only thing left is to actually display them.

```
if frame is not None:
    for raw_bbox in bboxes:
        bbox = frame_norm(frame, raw_bbox)
        cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (255, 0, 0), 2)
    cv2.imshow("preview", frame)
```

You can see here the usage of `frame_norm` we defined earlier for bounding box coordinates normalization. By using `cv2.rectangle` we draw a rectangle on the `rgb` frame as an indicator where the face position is, and then we display the frame using `cv2.imshow`

Finally, we add a way to terminate our program (as it's running inside an infinite loop). We will use `cv2.waitKey` method, that waits for a key to be pressed by user - in our case, we want to break out of the loop when user presses `q` key

```
if cv2.waitKey(1) == ord('q'):  
    break
```

3.8.12 Running the example

Putting it all together, only thing left to do is to run the file we've prepared in this tutorial and see the results

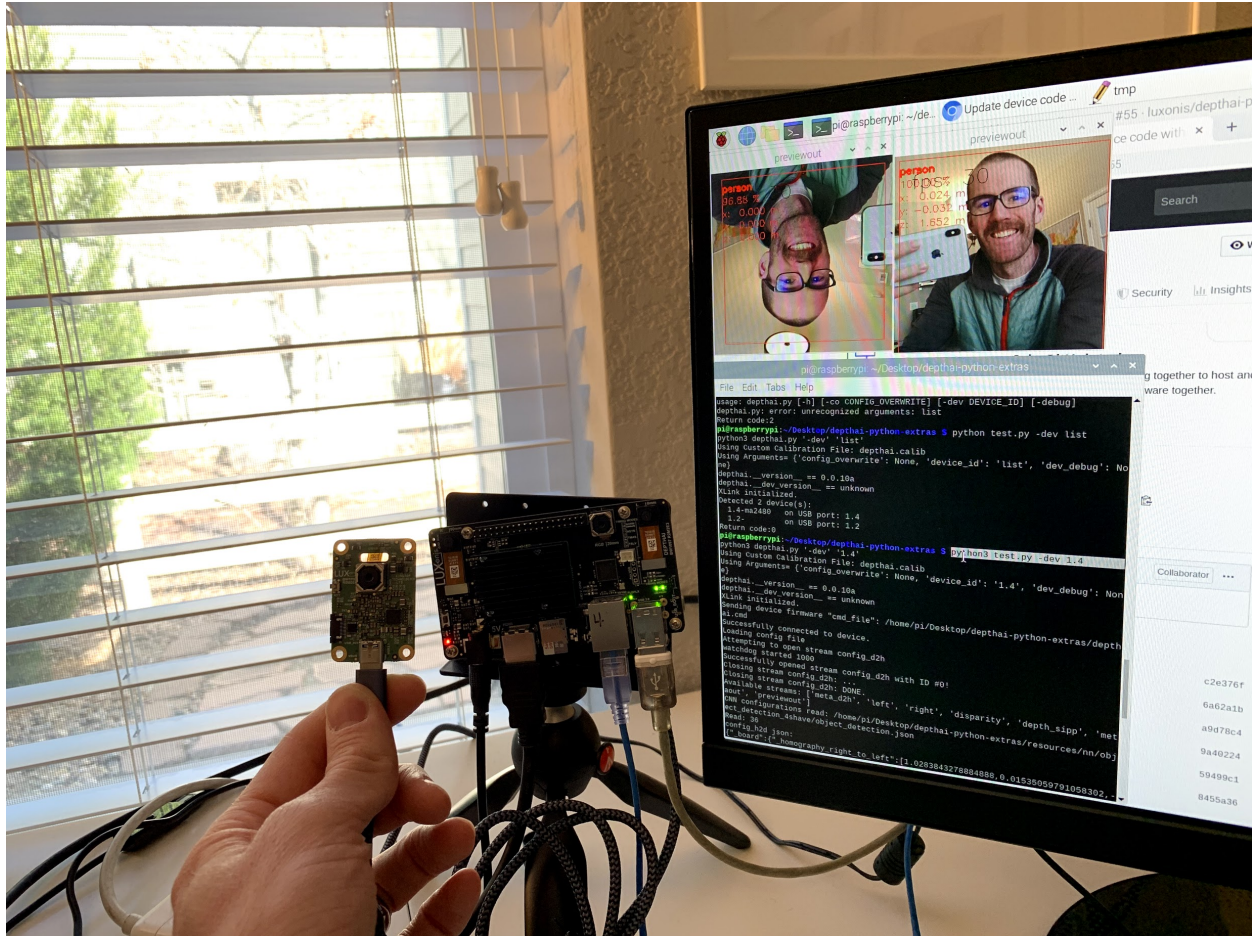
```
python3 hello_world.py
```

You're on your way! You can find the [complete code for this tutorial on GitHub](#).

We're always happy to help with code or other questions you might have.

3.9 Multiple DepthAI per Host

Learn how to discover DepthAI devices connected to your system, and use them individually.



Shown on the left is Luxonis **uAI (BW1093)** which is actually plugged into a **Raspberry Pi Compute Module Edition (BW1097)**.

So in this case, everything is running on the (single) Raspberry Pi 3B+ which is in the back of the BW1097.

3.9.1 Demo code

You can find demo code [here](#). The demo will find all devices connected to the host and display an RGB preview from each of them.

3.9.2 Dependencies

You have already set up the Python API on your system (if you have a Raspberry Pi Compute Module it came pre-setup). See [here](#) if you have not yet installed the DepthAI Python API on your system.

3.9.3 Discover DepthAI-USB Port Mapping

The DepthAI multi-device support is currently done by selecting the device `mx_id` (serial number) of a connected DepthAI device.

If you'd like to associate a given DepthAI device with specific code (e.g. neural model) to be run on it, it is recommended to plug in one device at a time, and then use the following code to determine which device is on which port:

```
import depthai
for device in depthai.Device.getAllAvailableDevices():
    print(f"{device.getMxId()} {device.state}")
```

Example results for 2x DepthAI on a system:

```
14442C10D13EABCE00 XLinkDeviceState.X_LINK_UNBOOTED
14442C1071659ACD00 XLinkDeviceState.X_LINK_UNBOOTED
```

3.9.4 Selecting a Specific DepthAI device to be used.

From the Detected devices(s) above, use the following code to select the device you would like to use with your pipeline. For example, if the first device is desirable from above use the following code:

```
found, device_info = depthai.Device.getDeviceByMxId("14442C10D13EABCE00")

if not found:
    raise RuntimeError("Device not found!")
```

You can then use the `device_info` to specify on which device you want to run your pipeline:

```
with depthai.Device(pipeline, device_info) as device:
```

And you can use this code as a basis for your own use cases, such that you can run differing neural models on different DepthAI/uAI models.

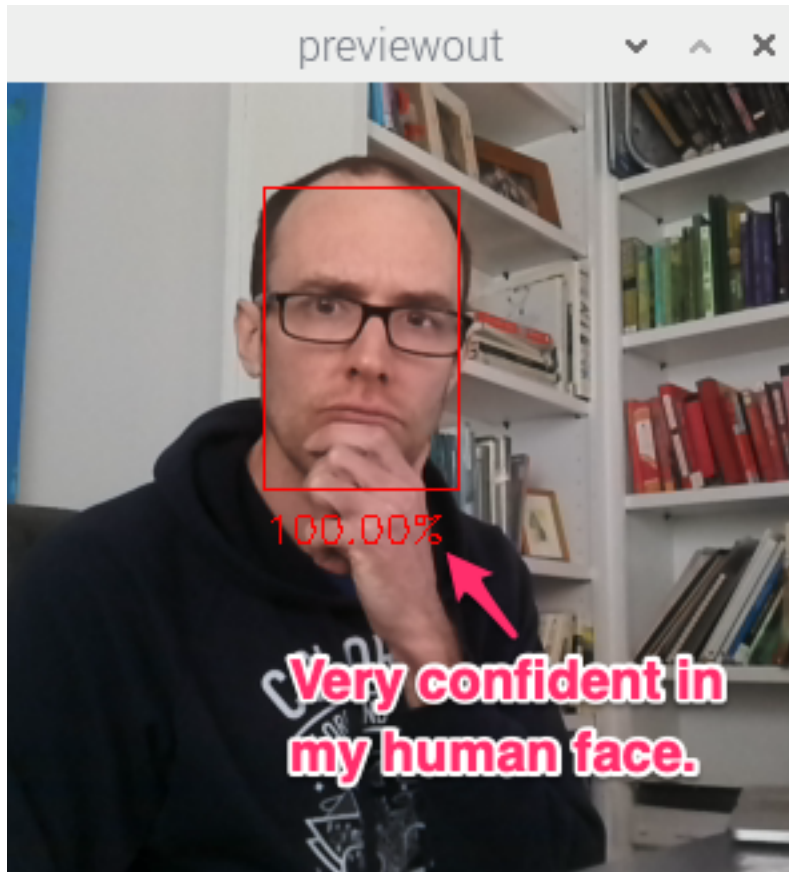
Now use as many DepthAI devices as you need!

And since DepthAI does all the heavy lifting, you can usually use quite a few of them with very little burden to the host.

We're always happy to help with code or other questions you might have.

3.10 Local OpenVINO Model Conversion

In this tutorial, you'll learn how to convert OpenVINO IR models into the format required to run on DepthAI, even on a low-powered Raspberry Pi. I'll introduce you to the OpenVINO toolset, the Open Model Zoo (where we'll download the [face-detection-retail-0004](#) model), and show you how to generate the files needed to run model inference on your DepthAI board.



Haven't heard of OpenVINO or the Open Model Zoo? I'll start with a quick introduction of why we need these tools.

3.10.1 What is OpenVINO?

Under-the-hood, DepthAI uses the Intel technology to perform high-speed model inference. However, you can't just dump your neural net into the chip and get high-performance for free. That's where [OpenVINO](#) comes in. OpenVINO is a free toolkit that converts a deep learning model into a format that runs on Intel Hardware. Once the model is converted, it's common to see Frames Per Second (FPS) improve by 25x or more. Are a couple of small steps worth a 25x FPS increase? Often, the answer is yes!

3.10.2 What is the Open Model Zoo?

The [Open Model Zoo](#) is a library of freely-available pre-trained models. The Zoo also contains scripts for downloading those models into a compile-ready format to run on DepthAI.

DepthAI is able to run many of the object detection models in the Zoo.

3.10.3 Install OpenVINO

Warning: If you have OpenVINO installed or want to follow [official installation](#), skip this step.

Please note that the following install instructions are for **Ubuntu 18.04** OS, if you intend to use other OS, follow the official OpenVINO installation

DepthAI requires OpenVINO version 2020.1. Let's get a package for our OS and meeting this version with the following command:

```
apt-get update
apt-get install -y software-properties-common
add-apt-repository -y ppa:deadsnakes/ppa
apt-get update
apt-get install -y wget pciutils python3.8 libpng-dev libcairo2-dev libpango1.0-dev \
↳ libglib2.0-dev libgtk2.0-dev libswscale-dev libavcodec-dev libavformat-dev
cd
mkdir openvino_install && cd openvino_install
wget http://registrationcenter-download.intel.com/akdlm/irc_nas/16345/l_openvino_
↳ toolkit_p_2020.1.023.tgz
tar --strip-components=1 -zxvf l_openvino_toolkit_p_2020.1.023.tgz
./install_openvino_dependencies.sh
./install.sh # when finished, you can go ahead and do "rm -r ~/openvino_install"
```

Now, first screen we'll see is EULA, just hit Enter, scroll through and type accept.

Next one is agreement to Intel Software Improvement Program, it's not relevant so you can choose whether consent (1) or not (2)

Next, you may see the Missing Prerequisites screen showing that Intel® Graphics Compute Runtime for OpenCL™ Driver is missing - you can go ahead and ignore this warning.

Finally, we'll see the install summary - please verify that it has a correct location pointed out - /opt/intel. If all looks good, go ahead and proceed (1). If the missing prerequisites screen appears again, feel free to skip it.

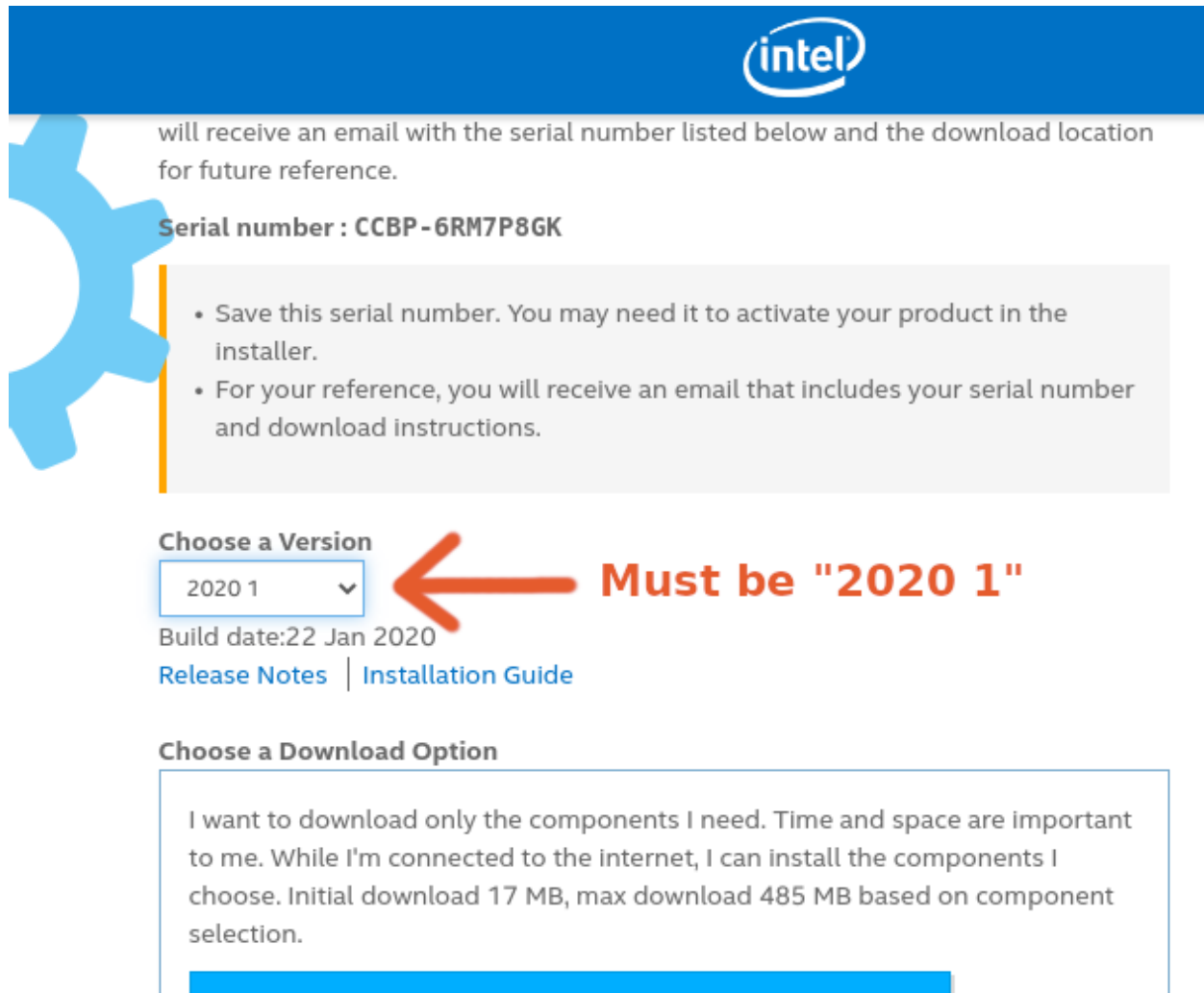
Let's verify that a correct version is installed on your host. Check your version by running the following from a terminal session:

```
cat /opt/intel/openvino/inference_engine/version.txt
```

You should see output similar to:

```
Thu Jan 23 19:14:14 MSK 2020
d349c3ba4a2508be72f413fa4dee92cc0e4bc0e1
releases_2020_1_InferenceEngine_37988
```

Verify that you see releases_2020_1 in your output. If you do, move on. If you are on a different version, goto the [OpenVINO site](#) and download the 2020.1 version for your OS:



will receive an email with the serial number listed below and the download location for future reference.

Serial number : CCBP-6RM7P8GK

- Save this serial number. You may need it to activate your product in the installer.
- For your reference, you will receive an email that includes your serial number and download instructions.

Choose a Version

2020 1 ▼ **← Must be "2020 1"**

Build date: 22 Jan 2020

[Release Notes](#) | [Installation Guide](#)

Choose a Download Option

I want to download only the components I need. Time and space are important to me. While I'm connected to the internet, I can install the components I choose. Initial download 17 MB, max download 485 MB based on component selection.

3.10.4 Check if the Model Downloader is installed

When installing OpenVINO, you can choose to perform a smaller install to save disk space. This custom install may not include the model downloader script. Lets check if the downloader was installed. In a terminal session, type the following:

```
find /opt/intel/ -iname downloader.py
```

Move on if you see the output below:

```
/opt/intel/openvino_2020.1.023/deployment_tools/open_model_zoo/tools/downloader/  
↪downloader.py
```

Didn't see any output? Don't fret if `downloader.py` isn't found. We'll install this below.

Install Open Model Zoo Downloader

If the downloader tools weren't found, we'll install the tools by cloning the [Open Model Zoo Repo](#) and installing the tool dependencies.

Start a terminal session and run the following commands in your terminal:

```
apt-get install -y git curl
curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
python3 get-pip.py
rm get-pip.py
cd ~
git clone https://github.com/opencv/open_model_zoo.git
cd open_model_zoo
git checkout tags/2020.1
cd tools/downloader
python3 -m pip install --user -r ./requirements.in
```

This clones the repo into a `~/open_model_zoo` directory, checks out the required `2020.1` version, and installs the downloader dependencies.

3.10.5 Create an `OPEN_MODEL_DOWNLOADER` environment variable

Typing the full path to `downloader.py` can use a lot of keystrokes. In an effort to extend your keyboard life, let's store the path to this script in an environment variable.

Run the following in your terminal:

```
export OPEN_MODEL_DOWNLOADER='INSERT PATH TO YOUR downloader.py SCRIPT'
```

Where `INSERT PATH TO YOUR downloader.py SCRIPT` can be found via:

```
find /opt/intel/ -iname downloader.py
find ~ -iname downloader.py
```

For example, if you installed `open_model_zoo` yourself:

```
export OPEN_MODEL_DOWNLOADER="$HOME/open_model_zoo/tools/downloader/downloader.py"
```

3.10.6 Download the face-detection-retail-0004 model

We've installed everything we need to download models from the Open Model Zoo! We'll now use the [Model Downloader](#) to download the `face-detection-retail-0004` model files. Run the following in your terminal:

```
$OPEN_MODEL_DOWNLOADER --name face-detection-retail-0004 --output_dir ~/open_model_
↪zoo_downloads/
```

This will download the model files to `~/open_model_zoo_downloads/`. Specifically, the model files we need are located at:

```
~/open_model_zoo_downloads/intel/face-detection-retail-0004/FP16
```

You'll see two files within the directory:

```
$ ls -lh
total 1.3M
-rw-r--r-- 1 root root 1.2M Jul 28 12:40 face-detection-retail-0004.bin
-rw-r--r-- 1 root root 100K Jul 28 12:40 face-detection-retail-0004.xml
```

The model is in the OpenVINO Intermediate Representation (IR) format:

- face-detection-retail-0004.xml - Describes the network topology
- face-detection-retail-0004.bin - Contains the weights and biases binary data.

This means we are ready to compile the model for the MyriadX!

3.10.7 Compile the model

The MyriadX chip used on our DepthAI board does not use the IR format files directly. Instead, we need to generate face-detection-retail-0004.blob using `myriad_compile` command.

Locate `myriad_compile`

Let's find where `myriad_compile` is located. In your terminal, run:

```
find /opt/intel/ -iname myriad_compile
```

You should see the output similar to this

```
find /opt/intel/ -iname myriad_compile
/opt/intel/openvino_2020.1.023/deployment_tools/inference_engine/lib/intel64/myriad_
↳ compile
```

Since it's such a long path, let's store the `myriad_compile` executable in an environment variable (just like `OPEN_MODEL_DOWNLOADER`):

```
export MYRIAD_COMPILE=$(find /opt/intel/ -iname myriad_compile)
```

Activate OpenVINO environment

In order to use `myriad_compile` tool, we need to activate our OpenVINO environment.

First, let's find `setupvars.sh` file

```
find /opt/intel/ -name "setupvars.sh"
/opt/intel/openvino_2020.1.023/opencv/setupvars.sh
/opt/intel/openvino_2020.1.023/bin/setupvars.sh
```

We're interested in `bin/setupvars.sh` file, so let's go ahead and source it to activate the environment:

```
source /opt/intel/openvino_2020.1.023/bin/setupvars.sh
[setupvars.sh] OpenVINO environment initialized
```

If you see `[setupvars.sh] OpenVINO environment initialized` then your environment should be initialized correctly

Run myriad_compile

```
$MYRIAD_COMPILE -m ~/open_model_zoo_downloads/intel/face-detection-retail-0004/FP16/
↳face-detection-retail-0004.xml -ip U8 -VPU_MYRIAD_PLATFORM VPU_MYRIAD_2480 -VPU_
↳NUMBER_OF_SHAVES 4 -VPU_NUMBER_OF_CMX_SLICES 4
```

You should see:

```
Inference Engine:
  API version ..... 2.1
  Build ..... 37988
  Description ..... API
Done
```

Where's the blob file? It's located in the same folder as `face-detection-retail-0004.xml`:

```
ls -lh ~/open_model_zoo_downloads/intel/face-detection-retail-0004/FP16/
total 2.6M
-rw-r--r-- 1 root root 1.2M Jul 28 12:40 face-detection-retail-0004.bin
-rw-r--r-- 1 root root 1.3M Jul 28 12:50 face-detection-retail-0004.blob
-rw-r--r-- 1 root root 100K Jul 28 12:40 face-detection-retail-0004.xml
```

3.10.8 Run and display the model output

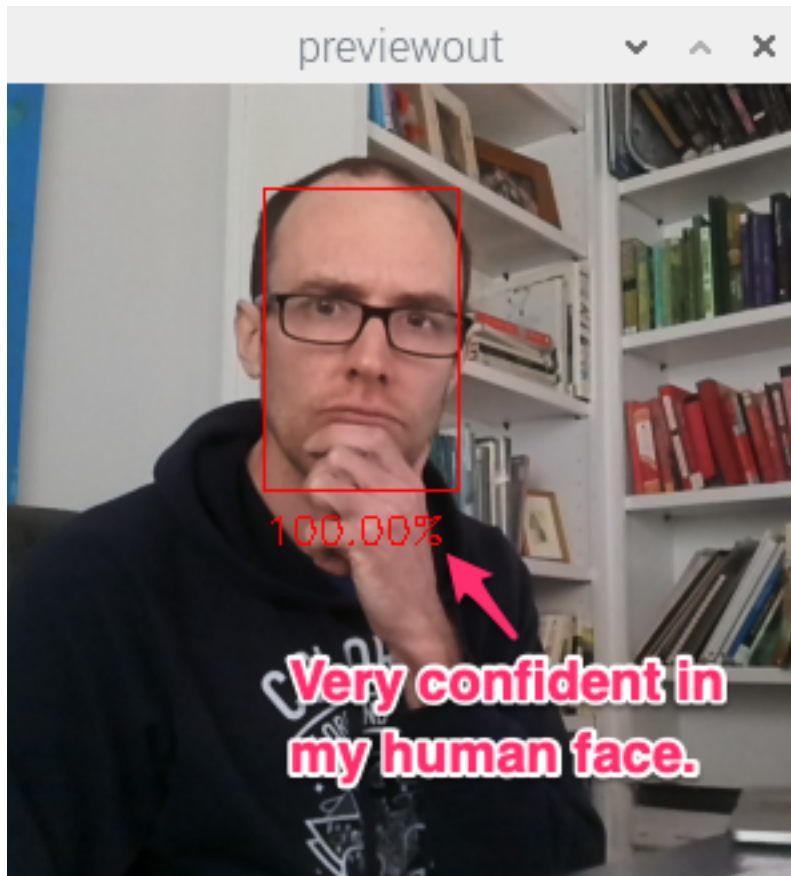
With neural network blob in place, we're ready to roll! To verify that the model is running correctly, let's modify a bit the program we've created in *Hello World* tutorial

In particular, let's change the `setBlobPath` invocation to load our model. **Remember to replace the paths to correct ones that you have!**

```
- detection_nn.setBlobPath("/path/to/mobilenet-ssd.blob")
- detection_nn.setBlobPath("/path/to/face-detection-retail-0004.blob")
```

And that's all!

You should see output annotated output similar to:



3.10.9 Reviewing the flow

The flow we walked through works for other pre-trained object detection models in the Open Model Zoo:

1. Download the model:

```
$OPEN_MODEL_DOWNLOADER --name [INSERT MODEL NAME] --output_dir ~/open_  
↪model_zoo_downloads/
```

2. Create the MyriadX blob file:

```
$MYRIAD_COMPILE -m [INSERT PATH TO MODEL XML FILE] -ip U8 -VPU_MYRIAD_  
↪PLATFORM VPU_MYRIAD_2480 -VPU_NUMBER_OF_SHAVES 4 -VPU_NUMBER_OF_CMX_  
↪SLICES 4
```

3. Use this model in your script

You're on your way! You can find the [complete code for this tutorial on GitHub](#).

We're always happy to help with code or other questions you might have.

3.11 Code samples

3.11.1 01 - RGB Preview

This example shows how to set up a pipeline that outputs a small preview of the RGB camera, connects over XLink to transfer these to the host real-time, and displays the RGB frames on the host with OpenCV.

Demo

Setup

Please run the following command to install the required dependencies

```
python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai
```

For additional information, please follow *installation guide*

Source code

Also available on [GitHub](#)

```

1  #!/usr/bin/env python3
2
3  import cv2
4  import depthai as dai
5
6  # Start defining a pipeline
7  pipeline = dai.Pipeline()
8
9  # Define a source - color camera
10 camRgb = pipeline.createColorCamera()
11 camRgb.setPreviewSize(300, 300)
12 camRgb.setBoardSocket(dai.CameraBoardSocket.RGB)
13 camRgb.setResolution(dai.ColorCameraProperties.SensorResolution.THE_1080_P)
14 camRgb.setInterleaved(False)
15 camRgb.setColorOrder(dai.ColorCameraProperties.ColorOrder.RGB)
16
17 # Create output
18 xoutRgb = pipeline.createXLinkOut()
19 xoutRgb.setStreamName("rgb")
20 camRgb.preview.link(xoutRgb.input)
21
22 # Pipeline is defined, now we can connect to the device
23 with dai.Device(pipeline) as device:
24     # Start pipeline
25     device.startPipeline()
26
27     # Output queue will be used to get the rgb frames from the output defined above
28     qRgb = device.getOutputQueue(name="rgb", maxSize=4, blocking=False)
29
30     while True:
31         inRgb = qRgb.get() # blocking call, will wait until a new data has arrived

```

(continues on next page)

(continued from previous page)

```

32
33     # Retrieve 'bgr' (opencv format) frame
34     cv2.imshow("bgr", inRgb.getCvFrame())
35
36     if cv2.waitKey(1) == ord('q'):
37         break

```

We're always happy to help with code or other questions you might have.

3.11.2 02 - Mono Preview

This example shows how to set up a pipeline that outputs the left and right grayscale camera images, connects over XLink to transfer these to the host real-time, and displays both using OpenCV.

Demo

Setup

Please run the following command to install the required dependencies

```

python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai

```

For additional information, please follow *installation guide*

Source code

Also available on [GitHub](#)

```

1  #!/usr/bin/env python3
2
3  import cv2
4  import depthai as dai
5
6  # Start defining a pipeline
7  pipeline = dai.Pipeline()
8
9  # Define a source - two mono (grayscale) cameras
10 camLeft = pipeline.createMonoCamera()
11 camLeft.setBoardSocket(dai.CameraBoardSocket.LEFT)
12 camLeft.setResolution(dai.MonoCameraProperties.SensorResolution.THE_720_P)
13
14 camRight = pipeline.createMonoCamera()
15 camRight.setBoardSocket(dai.CameraBoardSocket.RIGHT)
16 camRight.setResolution(dai.MonoCameraProperties.SensorResolution.THE_720_P)
17
18 # Create outputs
19 xoutLeft = pipeline.createXLinkOut()
20 xoutLeft.setStreamName('left')
21 camLeft.out.link(xoutLeft.input)
22
23 xoutRight = pipeline.createXLinkOut()

```

(continues on next page)

(continued from previous page)

```

24 xoutRight.setStreamName('right')
25 camRight.out.link(xoutRight.input)
26
27 # Pipeline is defined, now we can connect to the device
28 with dai.Device(pipeline) as device:
29     # Start pipeline
30     device.startPipeline()
31
32     # Output queues will be used to get the grayscale frames from the outputs defined
↪above
33     qLeft = device.getOutputQueue(name="left", maxSize=4, blocking=False)
34     qRight = device.getOutputQueue(name="right", maxSize=4, blocking=False)
35
36     frameLeft = None
37     frameRight = None
38
39     while True:
40         # Instead of get (blocking), we use tryGet (nonblocking) which will return
↪the available data or None otherwise
41         inLeft = qLeft.tryGet()
42         inRight = qRight.tryGet()
43
44         if inLeft is not None:
45             frameLeft = inLeft.getCvFrame()
46
47         if inRight is not None:
48             frameRight = inRight.getCvFrame()
49
50         # show the frames if available
51         if frameLeft is not None:
52             cv2.imshow("left", frameLeft)
53         if frameRight is not None:
54             cv2.imshow("right", frameRight)
55
56         if cv2.waitKey(1) == ord('q'):
57             break

```

We're always happy to help with code or other questions you might have.

3.11.3 03 - Depth Preview

This example shows how to set the SGBM (semi-global-matching) disparity-depth node, connects over XLink to transfer the results to the host real-time, and displays the depth map in OpenCV. Note that disparity is used in this case, as it colorizes in a more intuitive way. Below is also a preview of using different median filters side-by-side on a depth image.

Demo

Filtering depth using median filter

Setup

Please run the following command to install the required dependencies

```
python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai
```

For additional information, please follow *installation guide*

Source code

Also available on GitHub

```

1  #!/usr/bin/env python3
2
3  import cv2
4  import depthai as dai
5  import numpy as np
6
7
8  '''
9  If one or more of the additional depth modes (lrccheck, extended, subpixel)
10 are enabled, then:
11 - depth output is FP16. TODO enable U16.
12 - median filtering is disabled on device. TODO enable.
13 - with subpixel, either depth or disparity has valid data.
14 Otherwise, depth output is U16 (mm) and median is functional.
15 But like on Gen1, either depth or disparity has valid data. TODO enable both.
16 '''
17
18 # Closer-in minimum depth, disparity range is doubled (from 95 to 190):
19 extended_disparity = False
20 # Better accuracy for longer distance, fractional disparity 32-levels:
21 subpixel = False
22 # Better handling for occlusions:
23 lr_check = False
24
25 # Start defining a pipeline
26 pipeline = dai.Pipeline()
27
28 # Define a source - two mono (grayscale) cameras
29 left = pipeline.createMonoCamera()
30 left.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
31 left.setBoardSocket(dai.CameraBoardSocket.LEFT)
32
33 right = pipeline.createMonoCamera()
34 right.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
35 right.setBoardSocket(dai.CameraBoardSocket.RIGHT)
36
37 # Create a node that will produce the depth map (using disparity output as it's
  ↪ easier to visualize depth this way)

```

(continues on next page)

(continued from previous page)

```

38 depth = pipeline.createStereoDepth()
39 depth.setConfidenceThreshold(200)
40 depth.setOutputDepth(False)
41 # Options: MEDIAN_OFF, KERNEL_3x3, KERNEL_5x5, KERNEL_7x7 (default)
42 median = dai.StereoDepthProperties.MedianFilter.KERNEL_7x7 # For depth filtering
43 depth.setMedianFilter(median)
44
45 depth.setLeftRightCheck(lr_check)
46
47 # Normal disparity values range from 0..95, will be used for normalization
48 max_disparity = 95
49
50 if extended_disparity: max_disparity *= 2 # Double the range
51 depth.setExtendedDisparity(extended_disparity)
52
53 if subpixel: max_disparity *= 32 # 5 fractional bits, x32
54 depth.setSubpixel(subpixel)
55
56 # When we get disparity to the host, we will multiply all values with the multiplier
57 # for better visualization
58 multiplier = 255 / max_disparity
59
60 left.out.link(depth.left)
61 right.out.link(depth.right)
62
63 # Create output
64 xout = pipeline.createXLinkOut()
65 xout.setStreamName("disparity")
66 depth.disparity.link(xout.input)
67
68 # Pipeline is defined, now we can connect to the device
69 with dai.Device(pipeline) as device:
70     # Start pipeline
71     device.startPipeline()
72
73     # Output queue will be used to get the disparity frames from the outputs defined
74     ↪above
75     q = device.getOutputQueue(name="disparity", maxSize=4, blocking=False)
76     while True:
77         inDepth = q.get() # blocking call, will wait until a new data has arrived
78         frame = inDepth.getFrame()
79         frame = (frame*multiplier).astype(np.uint8)
80         # Available color maps: https://docs.opencv.org/3.4/d3/d50/group__imgproc__
81         ↪colormap.html
82         frame = cv2.applyColorMap(frame, cv2.COLORMAP_JET)
83
84         # frame is ready to be shown
85         cv2.imshow("disparity", frame)
86
87         if cv2.waitKey(1) == ord('q'):
88             break

```

We're always happy to help with code or other questions you might have.

3.11.4 04 - RGB Encoding

This example shows how to configure the depthai video encoder in h.265 format to encode the RGB camera input at 8MP/4K/2160p (3840x2160) at 30FPS (the maximum possible encoding resolution possible for the encoder, higher frame-rates are possible at lower resolutions, like 1440p at 60FPS), and transfers the encoded video over XLINK to the host, saving it to disk as a video file.

Pressing Ctrl+C will stop the recording and then convert it using ffmpeg into an mp4 to make it playable. Note that ffmpeg will need to be installed and runnable for the conversion to mp4 to succeed.

Be careful, this example saves encoded video to your host storage. So if you leave it running, you could fill up your storage on your host.

Demo

Setup

Please run the following command to install the required dependencies

```
python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai
```

For additional information, please follow [installation guide](#)

Source code

Also available on GitHub

```
1  #!/usr/bin/env python3
2
3  import depthai as dai
4
5  # Start defining a pipeline
6  pipeline = dai.Pipeline()
7
8  # Define a source - color camera
9  cam = pipeline.createColorCamera()
10 cam.setBoardSocket(dai.CameraBoardSocket.RGB)
11 cam.setResolution(dai.ColorCameraProperties.SensorResolution.THE_4_K)
12
13 # Create an encoder, consuming the frames and encoding them using H.265 encoding
14 videoEncoder = pipeline.createVideoEncoder()
15 videoEncoder.setDefaultProfilePreset(3840, 2160, 30, dai.VideoEncoderProperties.
16   ↪ Profile.H265_MAIN)
17 cam.video.link(videoEncoder.input)
18
19 # Create output
20 videoOut = pipeline.createXLinkOut()
21 videoOut.setStreamName('h265')
22 videoEncoder.bitstream.link(videoOut.input)
23
24 # Pipeline is defined, now we can connect to the device
25 with dai.Device(pipeline) as device:
26     # Start pipeline
27     device.startPipeline()
```

(continues on next page)

(continued from previous page)

```

27
28     # Output queue will be used to get the encoded data from the output defined above
29     q = device.getOutputQueue(name="h265", maxSize=30, blocking=True)
30
31     # The .h265 file is a raw stream file (not playable yet)
32     with open('video.h265', 'wb') as videoFile:
33         print("Press Ctrl+C to stop encoding...")
34         try:
35             while True:
36                 h264Packet = q.get() # Blocking call, will wait until a new data has
↳ arrived
37                 h264Packet.getData().tofile(videoFile) # Appends the packet data to
↳ the opened file
38             except KeyboardInterrupt:
39                 # Keyboard interrupt (Ctrl + C) detected
40                 pass
41
42         print("To view the encoded data, convert the stream file (.h265) into a video
↳ file (.mp4) using a command below:")
43         print("ffmpeg -framerate 30 -i video.h265 -c copy video.mp4")

```

We're always happy to help with code or other questions you might have.

3.11.5 05 - RGB & Mono Encoding

This example shows how to set up the encoder node to encode the RGB camera and both grayscale cameras (of DepthAI/OAK-D) at the same time. The RGB is set to 1920x1080 and the grayscale are set to 1280x720 each, all at 30FPS. Each encoded video stream is transferred over XLINK and saved to a respective file.

Pressing Ctrl+C will stop the recording and then convert it using ffmpeg into an mp4 to make it playable. Note that ffmpeg will need to be installed and runnable for the conversion to mp4 to succeed.

Be careful, this example saves encoded video to your host storage. So if you leave it running, you could fill up your storage on your host.

Demo

Setup

Please run the following command to install the required dependencies

```

python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai

```

For additional information, please follow [installation guide](#)

Source code

Also available on [GitHub](#)

```

1  #!/usr/bin/env python3
2
3  import depthai as dai
4
5  # Start defining a pipeline
6  pipeline = dai.Pipeline()
7
8  # Define a source - color and mono cameras
9  colorCam = pipeline.createColorCamera()
10 monoCam = pipeline.createMonoCamera()
11 monoCam.setBoardSocket(dai.CameraBoardSocket.LEFT)
12 monoCam2 = pipeline.createMonoCamera()
13 monoCam2.setBoardSocket(dai.CameraBoardSocket.RIGHT)
14
15 # Create encoders, one for each camera, consuming the frames and encoding them using
16 ↳ H.264 / H.265 encoding
17 ve1 = pipeline.createVideoEncoder()
18 ve1.setDefaultProfilePreset(1280, 720, 30, dai.VideoEncoderProperties.Profile.H264_
19 ↳ MAIN)
20 monoCam.out.link(ve1.input)
21
22 ve2 = pipeline.createVideoEncoder()
23 ve2.setDefaultProfilePreset(1920, 1080, 30, dai.VideoEncoderProperties.Profile.H265_
24 ↳ MAIN)
25 colorCam.video.link(ve2.input)
26
27 ve3 = pipeline.createVideoEncoder()
28 ve3.setDefaultProfilePreset(1280, 720, 30, dai.VideoEncoderProperties.Profile.H264_
29 ↳ MAIN)
30 monoCam2.out.link(ve3.input)
31
32 # Create outputs
33 ve1Out = pipeline.createXLinkOut()
34 ve1Out.setStreamName('ve1Out')
35 ve1.bitstream.link(ve1Out.input)
36
37 ve2Out = pipeline.createXLinkOut()
38 ve2Out.setStreamName('ve2Out')
39 ve2.bitstream.link(ve2Out.input)
40
41 ve3Out = pipeline.createXLinkOut()
42 ve3Out.setStreamName('ve3Out')
43 ve3.bitstream.link(ve3Out.input)
44
45 # Pipeline is defined, now we can connect to the device
46 with dai.Device(pipeline) as dev:
47     # Start pipeline
48     dev.startPipeline()
49
50     # Output queues will be used to get the encoded data from the outputs defined
51     ↳ above
52     outQ1 = dev.getOutputQueue(name='ve1Out', maxSize=30, blocking=True)
53     outQ2 = dev.getOutputQueue(name='ve2Out', maxSize=30, blocking=True)

```

(continues on next page)

(continued from previous page)

```

50 outQ3 = dev.getOutputQueue(name='ve3Out', maxSize=30, blocking=True)
51
52 # The .h264 / .h265 files are raw stream files (not playable yet)
53 with open('mono1.h264', 'wb') as fileMono1H264, open('color.h265', 'wb') as
54 →fileColorH265, open('mono2.h264', 'wb') as fileMono2H264:
55     print("Press Ctrl+C to stop encoding...")
56     while True:
57         try:
58             # Empty each queue
59             while outQ1.has():
60                 outQ1.get().getData().tofile(fileMono1H264)
61
62             while outQ2.has():
63                 outQ2.get().getData().tofile(fileColorH265)
64
65             while outQ3.has():
66                 outQ3.get().getData().tofile(fileMono2H264)
67         except KeyboardInterrupt:
68             # Keyboard interrupt (Ctrl + C) detected
69             break
70
71     print("To view the encoded data, convert the stream file (.h264/.h265) into a
72 →video file (.mp4), using commands below:")
73     cmd = "ffmpeg -framerate 30 -i {} -c copy {}"
74     print(cmd.format("mono1.h264", "mono1.mp4"))
75     print(cmd.format("mono2.h264", "mono2.mp4"))
76     print(cmd.format("color.h265", "color.mp4"))

```

We're always happy to help with code or other questions you might have.

3.11.6 06 - RGB Full Resolution Saver

This example does its best to save full-resolution 3840x2160 .jpeg files as fast as it can from the RGB sensor. It serves as an example of recording high resolution to disk for the purposes of high-resolution ground-truth data. We also recently added the options to save isp - YUV420p uncompressed frames, processed by ISP, and raw - BayerRG (R_Gr_Gb_B), as read from sensor, 10-bit packed. See [here](#) for the pull request on this capability.

Be careful, this example saves full resolution .jpeg pictures to your host storage. So if you leave it running, you could fill up your storage on your host.

Demo

Setup

Please run the following command to install the required dependencies

```

python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai

```

For additional information, please follow [installation guide](#)

Source code

Also available on [GitHub](#)

```

1  #!/usr/bin/env python3
2
3  import time
4  from pathlib import Path
5
6  import cv2
7  import depthai as dai
8
9  # Start defining a pipeline
10 pipeline = dai.Pipeline()
11
12 # Define a source - color camera
13 camRgb = pipeline.createColorCamera()
14 camRgb.setResolution(dai.ColorCameraProperties.SensorResolution.THE_4_K)
15
16 # Create RGB output
17 xoutRgb = pipeline.createXLinkOut()
18 xoutRgb.setStreamName("rgb")
19 camRgb.video.link(xoutRgb.input)
20
21 # Create encoder to produce JPEG images
22 videoEnc = pipeline.createVideoEncoder()
23 videoEnc.setDefaultProfilePreset(camRgb.getVideoSize(), camRgb.getFps(), dai.
24   ↳ VideoEncoderProperties.Profile.MJPEG)
25 camRgb.video.link(videoEnc.input)
26
27 # Create JPEG output
28 xoutJpeg = pipeline.createXLinkOut()
29 xoutJpeg.setStreamName("jpeg")
30 videoEnc.bitstream.link(xoutJpeg.input)
31
32 # Pipeline is defined, now we can connect to the device
33 with dai.Device(pipeline) as device:
34     # Start pipeline
35     device.startPipeline()
36
37     # Output queue will be used to get the rgb frames from the output defined above
38     qRgb = device.getOutputQueue(name="rgb", maxSize=30, blocking=False)
39     qJpeg = device.getOutputQueue(name="jpeg", maxSize=30, blocking=True)
40
41     # Make sure the destination path is present before starting to store the examples
42     Path('06_data').mkdir(parents=True, exist_ok=True)
43
44     while True:
45         inRgb = qRgb.tryGet() # Non-blocking call, will return a new data that has_
46         ↳ arrived or None otherwise
47
48         if inRgb is not None:
49             cv2.imshow("rgb", inRgb.getCvFrame())
50
51         for encFrame in qJpeg.tryGetAll():
52             with open(f"06_data/{int(time.time() * 1000)}.jpeg", "wb") as f:
53                 f.write(bytearray(encFrame.getData()))

```

(continues on next page)

(continued from previous page)

```

53
54     if cv2.waitKey(1) == ord('q'):
55         break

```

We're always happy to help with code or other questions you might have.

3.11.7 07 - Mono Full Resolution Saver

This example shows how to save 1280x720p .png of the right grayscale camera to disk. Left is defined as from the boards perspective.

Demo

Setup

Please run the following command to install the required dependencies

```

python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai

```

For additional information, please follow [installation guide](#)

Source code

Also available on [GitHub](#)

```

1  #!/usr/bin/env python3
2
3  import time
4  from pathlib import Path
5
6  import cv2
7  import depthai as dai
8
9  # Start defining a pipeline
10 pipeline = dai.Pipeline()
11
12 # Define a source - mono (grayscale) camera
13 camRight = pipeline.createMonoCamera()
14 camRight.setBoardSocket(dai.CameraBoardSocket.RIGHT)
15 camRight.setResolution(dai.MonoCameraProperties.SensorResolution.THE_720_P)
16
17 # Create output
18 xoutRight = pipeline.createXLinkOut()
19 xoutRight.setStreamName("right")
20 camRight.out.link(xoutRight.input)
21
22 # Pipeline is defined, now we can connect to the device
23 with dai.Device(pipeline) as device:
24     # Start pipeline
25     device.startPipeline()
26

```

(continues on next page)

(continued from previous page)

```

27     # Output queue will be used to get the grayscale frames from the output defined_
↪above
28     qRight = device.getOutputQueue(name="right", maxSize=4, blocking=False)
29
30     # Make sure the destination path is present before starting to store the examples
31     Path('07_data').mkdir(parents=True, exist_ok=True)
32
33     while True:
34         inRight = qRight.get() # Blocking call, will wait until a new data has_
↪arrived
35         # Data is originally represented as a flat 1D array, it needs to be converted_
↪into HxW form
36         frameRight = inRight.getCvFrame()
37         # Frame is transformed and ready to be shown
38         cv2.imshow("right", frameRight)
39         # After showing the frame, it's being stored inside a target directory as a_
↪PNG image
40         cv2.imwrite(f"07_data/{int(time.time() * 10000)}.png", frameRight)
41
42         if cv2.waitKey(1) == ord('q'):
43             break

```

We're always happy to help with code or other questions you might have.

3.11.8 08 - RGB & MobilenetSSD

This example shows how to run MobileNetV2SSD on the RGB input frame, and how to display both the RGB preview and the metadata results from the MobileNetV2SSD on the preview.

Demo

Setup

Please run the following command to install the required dependencies

```

python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai

```

For additional information, please follow [installation guide](#)

This example also requires MobilenetSSD blob (mobilenet-ssd_openvino_2021.2_6shave.blob file) to work - you can download it from [here](#)

Source code

Also available on [GitHub](#)

```

1  #!/usr/bin/env python3
2
3  from pathlib import Path
4  import cv2
5  import depthai as dai
6  import numpy as np
7  import time
8  import argparse
9
10 nnPathDefault = str((Path(__file__).parent / Path('models/mobilenet-ssd_openvino_2021.
    ↳ 2_6shave.blob')).resolve().absolute())
11 parser = argparse.ArgumentParser()
12 parser.add_argument('nnPath', nargs='?', help="Path to mobilenet detection network_
    ↳ blob", default=nnPathDefault)
13 parser.add_argument('-s', '--sync', action="store_true", help="Sync RGB output with_
    ↳ NN output", default=False)
14 args = parser.parse_args()
15
16 if not Path(nnPathDefault).exists():
17     import sys
18     raise FileNotFoundError(f'Required file/s not found, please run "{sys.executable}_'
    ↳ install_requirements.py")
19
20 # Start defining a pipeline
21 pipeline = dai.Pipeline()
22
23 # Define a source - color camera
24 camRgb = pipeline.createColorCamera()
25 camRgb.setPreviewSize(300, 300)
26 camRgb.setInterleaved(False)
27 camRgb.setFps(40)
28
29 # Define a neural network that will make predictions based on the source frames
30 nn = pipeline.createMobileNetDetectionNetwork()
31 nn.setConfidenceThreshold(0.5)
32 nn.setBlobPath(args.nnPath)
33 nn.setNumInferenceThreads(2)
34 nn.input.setBlocking(False)
35 camRgb.preview.link(nn.input)
36
37 # Create outputs
38 xoutRgb = pipeline.createXLinkOut()
39 xoutRgb.setStreamName("rgb")
40 if args.sync:
41     nn.passthrough.link(xoutRgb.input)
42 else:
43     camRgb.preview.link(xoutRgb.input)
44
45 nnOut = pipeline.createXLinkOut()
46 nnOut.setStreamName("nn")
47 nn.out.link(nnOut.input)
48
49 # MobilenetSSD label texts
50 labelMap = ["background", "aeroplane", "bicycle", "bird", "boat", "bottle", "bus",
    ↳ "car", "cat", "chair", "cow",

```

(continues on next page)

(continued from previous page)

```

51         "diningtable", "dog", "horse", "motorbike", "person", "pottedplant",
    ↪ "sheep", "sofa", "train", "tvmonitor"]
52
53
54 # Pipeline is defined, now we can connect to the device
55 with dai.Device(pipeline) as device:
56     # Start pipeline
57     device.startPipeline()
58
59     # Output queues will be used to get the rgb frames and nn data from the outputs_
    ↪ defined above
60     qRgb = device.getOutputQueue(name="rgb", maxSize=4, blocking=False)
61     qDet = device.getOutputQueue(name="nn", maxSize=4, blocking=False)
62
63     startTime = time.monotonic()
64     counter = 0
65     detections = []
66     frame = None
67
68     # nn data (bounding box locations) are in <0..1> range - they need to be_
    ↪ normalized with frame width/height
69     def frameNorm(frame, bbox):
70         normVals = np.full(len(bbox), frame.shape[0])
71         normVals[::2] = frame.shape[1]
72         return (np.clip(np.array(bbox), 0, 1) * normVals).astype(int)
73
74     def displayFrame(name, frame):
75         for detection in detections:
76             bbox = frameNorm(frame, (detection.xmin, detection.ymin, detection.xmax,
    ↪ detection.ymax))
77             cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (255, 0, 0),
    ↪ 2)
78             cv2.putText(frame, labelMap[detection.label], (bbox[0] + 10, bbox[1] +
    ↪ 20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
79             cv2.putText(frame, f"{int(detection.confidence * 100)}%", (bbox[0] + 10,
    ↪ bbox[1] + 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
80             cv2.imshow(name, frame)
81
82
83     while True:
84         if args.sync:
85             # Use blocking get() call to catch frame and inference result synced
86             inRgb = qRgb.get()
87             inDet = qDet.get()
88         else:
89             # Instead of get (blocking), we use tryGet (nonblocking) which will_
    ↪ return the available data or None otherwise
90             inRgb = qRgb.tryGet()
91             inDet = qDet.tryGet()
92
93             if inRgb is not None:
94                 frame = inRgb.getCvFrame()
95                 cv2.putText(frame, "NN fps: {:.2f}".format(counter / (time.monotonic() -
    ↪ startTime)),
96                             (2, frame.shape[0] - 4), cv2.FONT_HERSHEY_TRIPLEX, 0.4,
    ↪ color=(255, 255, 255))
97

```

(continues on next page)

(continued from previous page)

```

98     if inDet is not None:
99         detections = inDet.detections
100         counter += 1
101
102     # If the frame is available, draw bounding boxes on it and show the frame
103     if frame is not None:
104         displayFrame("rgb", frame)
105
106     if cv2.waitKey(1) == ord('q'):
107         break

```

We're always happy to help with code or other questions you might have.

3.11.9 09 - Mono & MobilenetSSD

This example shows how to run MobileNet2SSD on the right grayscale camera and how to display the neural network results on a preview of the right camera stream.

Demo

Setup

Please run the following command to install the required dependencies

```

python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai

```

For additional information, please follow [installation guide](#)

This example also requires MobilenetSSD blob (mobilenet-ssd_openvino_2021.2_6shave.blob file) to work - you can download it from [here](#)

Source code

Also available on GitHub

```

1  #!/usr/bin/env python3
2
3  from pathlib import Path
4  import sys
5  import cv2
6  import depthai as dai
7  import numpy as np
8
9  # Get argument first
10 nnPath = str((Path(__file__).parent / Path('models/mobilenet-ssd_openvino_2021.2_
    ↳ 6shave.blob')).resolve().absolute())
11 if len(sys.argv) > 1:
12     nnPath = sys.argv[1]
13
14 if not Path(nnPath).exists():
15     import sys

```

(continues on next page)

(continued from previous page)

```

16     raise FileNotFoundError(f'Required file/s not found, please run "{sys.executable} \
↳install_requirements.py"')
17
18 # Start defining a pipeline
19 pipeline = dai.Pipeline()
20
21 # Define a source - mono (grayscale) camera
22 camRight = pipeline.createMonoCamera()
23 camRight.setBoardSocket(dai.CameraBoardSocket.RIGHT)
24 camRight.setResolution(dai.MonoCameraProperties.SensorResolution.THE_720_P)
25
26 # Define a neural network that will make predictions based on the source frames
27 nn = pipeline.createMobileNetDetectionNetwork()
28 nn.setConfidenceThreshold(0.5)
29 nn.setBlobPath(nnPath)
30 nn.setNumInferenceThreads(2)
31 nn.input.setBlocking(False)
32
33 # Create a node to convert the grayscale frame into the nn-acceptable form
34 manip = pipeline.createImageManip()
35 manip.initialConfig.setResize(300, 300)
36 # The NN model expects BGR input. By default ImageManip output type would be same as
↳input (gray in this case)
37 manip.initialConfig.setFrameType(dai.ImgFrame.Type.BGR888p)
38 camRight.out.link(manip.inputImage)
39 manip.out.link(nn.input)
40
41 # Create outputs
42 manipOut = pipeline.createXLinkOut()
43 manipOut.setStreamName("right")
44 manip.out.link(manipOut.input)
45
46 nnOut = pipeline.createXLinkOut()
47 nnOut.setStreamName("nn")
48 nn.out.link(nnOut.input)
49
50 # MobilenetSSD label texts
51 labelMap = ["background", "aeroplane", "bicycle", "bird", "boat", "bottle", "bus",
↳"car", "cat", "chair", "cow",
52             "diningtable", "dog", "horse", "motorbike", "person", "pottedplant",
↳"sheep", "sofa", "train", "tvmonitor"]
53
54 # Pipeline is defined, now we can connect to the device
55 with dai.Device(pipeline) as device:
56     # Start pipeline
57     device.startPipeline()
58
59     # Output queues will be used to get the grayscale frames and nn data from the
↳outputs defined above
60     qRight = device.getOutputQueue("right", maxSize=4, blocking=False)
61     qDet = device.getOutputQueue("nn", maxSize=4, blocking=False)
62
63     frame = None
64     detections = []
65
66     # nn data, being the bounding box locations, are in <0..1> range - they need to
↳be normalized with frame width/height

```

(continues on next page)

(continued from previous page)

```

67     def frameNorm(frame, bbox):
68         normVals = np.full(len(bbox), frame.shape[0])
69         normVals[::2] = frame.shape[1]
70         return (np.clip(np.array(bbox), 0, 1) * normVals).astype(int)
71
72     def displayFrame(name, frame):
73         for detection in detections:
74             bbox = frameNorm(frame, (detection.xmin, detection.ymin, detection.xmax,
75 ↪ detection.ymax))
76             cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (255, 0, 0),
77 ↪ 2)
78             cv2.putText(frame, labelMap[detection.label], (bbox[0] + 10, bbox[1] +
79 ↪ 20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
80             cv2.putText(frame, f"{int(detection.confidence * 100)}%", (bbox[0] + 10,
81 ↪ bbox[1] + 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
82             cv2.imshow(name, frame)
83
84     while True:
85         # Instead of get (blocking), we use tryGet (nonblocking) which will return
86 ↪ the available data or None otherwise
87         inRight = qRight.tryGet()
88         inDet = qDet.tryGet()
89
90         if inRight is not None:
91             frame = inRight.getCvFrame()
92
93         if inDet is not None:
94             detections = inDet.detections
95
96         if frame is not None:
97             displayFrame("right", frame)
98
99         if cv2.waitKey(1) == ord('q'):
100             break

```

We're always happy to help with code or other questions you might have.

3.11.10 10 - Mono & MobilenetSSD & Depth

This example shows how to run MobileNetV2SSD on the left grayscale camera in parallel with running the disparity depth results, displaying both the depth map and the right grayscale stream, with the bounding box from the neural network overlaid.

Demo

Setup

Please run the following command to install the required dependencies

```

python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai

```

For additional information, please follow *installation guide*

This example also requires MobilenetSSD blob (mobilenet-ssd_openvino_2021.2_6shave.blob file) to work - you can download it from [here](#)

Source code

Also available on GitHub

```

1  #!/usr/bin/env python3
2
3  from pathlib import Path
4  import sys
5  import cv2
6  import depthai as dai
7  import numpy as np
8
9
10 flipRectified = True
11
12 # Get argument first
13 nnPath = str((Path(__file__).parent / Path('models/mobilenet-ssd_openvino_2021.2_
    ↳6shave.blob')).resolve().absolute())
14 if len(sys.argv) > 1:
15     nnPath = sys.argv[1]
16
17 if not Path(nnPath).exists():
18     import sys
19     raise FileNotFoundError(f'Required file/s not found, please run "{sys.executable}
    ↳install_requirements.py"')
20
21 # Start defining a pipeline
22 pipeline = dai.Pipeline()
23
24 # Define a source - mono (grayscale) cameras
25 left = pipeline.createMonoCamera()
26 left.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
27 left.setBoardSocket(dai.CameraBoardSocket.LEFT)
28
29 right = pipeline.createMonoCamera()
30 right.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
31 right.setBoardSocket(dai.CameraBoardSocket.RIGHT)
32
33 # Create a node that will produce the depth map (using disparity output as it's
    ↳easier to visualize depth this way)
34 stereo = pipeline.createStereoDepth()
35 stereo.setOutputRectified(True) # The rectified streams are horizontally mirrored by
    ↳default
36 stereo.setConfidenceThreshold(255)
37 stereo.setRectifyEdgeFillColor(0) # Black, to better see the cutout from
    ↳rectification (black stripe on the edges)
38
39 left.out.link(stereo.left)
40 right.out.link(stereo.right)
41
42 # Create a node to convert the grayscale frame into the nn-acceptable form
43 manip = pipeline.createImageManip()
```

(continues on next page)

(continued from previous page)

```

44 manip.initialConfig.setResize(300, 300)
45 # The NN model expects BGR input. By default ImageManip output type would be same as_
↳input (gray in this case)
46 manip.initialConfig.setFrameType(dai.ImgFrame.Type.BGR888p)
47 stereo.rectifiedRight.link(manip.inputImage)
48
49 # Define a neural network that will make predictions based on the source frames
50 nn = pipeline.createMobileNetDetectionNetwork()
51 nn.setConfidenceThreshold(0.5)
52 nn.setBlobPath(nnPath)
53 nn.setNumInferenceThreads(2)
54 nn.input.setBlocking(False)
55 manip.out.link(nn.input)
56
57 # Create outputs
58 disparityOut = pipeline.createXLinkOut()
59 disparityOut.setStreamName("disparity")
60 stereo.disparity.link(disparityOut.input)
61
62 xoutRight = pipeline.createXLinkOut()
63 xoutRight.setStreamName("rectifiedRight")
64 manip.out.link(xoutRight.input)
65
66 nnOut = pipeline.createXLinkOut()
67 nnOut.setStreamName("nn")
68 nn.out.link(nnOut.input)
69
70 # MobilenetSSD label nnLabels
71 labelMap = ["background", "aeroplane", "bicycle", "bird", "boat", "bottle", "bus",
↳"car", "cat", "chair", "cow",
72             "diningtable", "dog", "horse", "motorbike", "person", "pottedplant",
↳"sheep", "sofa", "train", "tvmonitor"]
73
74 # Pipeline is defined, now we can connect to the device
75 with dai.Device(pipeline) as device:
76     # Start pipeline
77     device.startPipeline()
78
79     # Output queues will be used to get the grayscale / depth frames and nn data from_
↳the outputs defined above
80     qRight = device.getOutputQueue("rectifiedRight", maxSize=4, blocking=False)
81     qDisparity = device.getOutputQueue("disparity", maxSize=4, blocking=False)
82     qDet = device.getOutputQueue("nn", maxSize=4, blocking=False)
83
84     rightFrame = None
85     depthFrame = None
86     detections = []
87     offsetX = (right.getResolutionWidth() - right.getResolutionHeight()) // 2
88     croppedFrame = np.zeros((right.getResolutionHeight(), right.
↳getResolutionHeight()))
89
90     # nn data, being the bounding box locations, are in <0..1> range - they need to_
↳be normalized with frame width/height
91     def frameNorm(frame, bbox):
92         normVals = np.full(len(bbox), frame.shape[0])
93         normVals[::2] = frame.shape[1]
94         return (np.clip(np.array(bbox), 0, 1) * normVals).astype(int)

```

(continues on next page)

(continued from previous page)

```

95
96     # Add bounding boxes and text to the frame and show it to the user
97     def show(name, frame):
98         for detection in detections:
99             bbox = frameNorm(frame, (detection.xmin, detection.ymin, detection.xmax,
100 detection.ymax))
101             cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (255, 0, 0),
102 2)
103             cv2.putText(frame, labelMap[detection.label], (bbox[0] + 10, bbox[1] +
104 20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
105             cv2.putText(frame, f"{int(detection.confidence * 100)}%", (bbox[0] + 10,
106 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
107         # Show the frame
108         cv2.imshow(name, frame)
109
110     disparity_multiplier = 255 / 95 # Disparity range is 0..95
111     while True:
112         # Instead of get (blocking), we use tryGet (nonblocking) which will return
113         # the available data or None otherwise
114         inRight = qRight.tryGet()
115         inDet = qDet.tryGet()
116         inDisparity = qDisparity.tryGet()
117
118         if inRight is not None:
119             rightFrame = inRight.getCvFrame()
120             if flipRectified:
121                 rightFrame = cv2.flip(rightFrame, 1)
122
123         if inDet is not None:
124             detections = inDet.detections
125             if flipRectified:
126                 for detection in detections:
127                     swap = detection.xmin
128                     detection.xmin = 1 - detection.xmax
129                     detection.xmax = 1 - swap
130
131         if inDisparity is not None:
132             # Frame is transformed, normalized, and color map will be applied to
133             # highlight the depth info
134             disparityFrame = inDisparity.getFrame()
135             disparityFrame = (disparityFrame*disparity_multiplier).astype(np.uint8)
136             # Available color maps: https://docs.opencv.org/3.4/d3/d50/group__imgproc_
137             # _colormap.html
138             disparityFrame = cv2.applyColorMap(disparityFrame, cv2.COLORMAP_JET)
139             show("disparity", disparityFrame)
140
141         if rightFrame is not None:
142             show("rectified right", rightFrame)
143
144         detections = []
145
146         if cv2.waitKey(1) == ord('q'):
147             break

```

We're always happy to help with code or other questions you might have.

3.11.11 11 - RGB & Encoding & Mono & MobilenetSSD

This example shows how to configure the depthai video encoder in h.265 format to encode the RGB camera input at Full-HD resolution at 30FPS, and transfers the encoded video over XLINK to the host, saving it to disk as a video file. At the same time, a MobileNetV2SSD network is ran on the frames from right grayscale camera

Pressing Ctrl+C will stop the recording and then convert it using ffmpeg into an mp4 to make it playable. Note that ffmpeg will need to be installed and runnable for the conversion to mp4 to succeed.

Be careful, this example saves encoded video to your host storage. So if you leave it running, you could fill up your storage on your host.

Demo

Setup

Please run the following command to install the required dependencies

```
python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai
```

For additional information, please follow [installation guide](#)

This example also requires MobilenetSSD blob (mobilenet-ssd_openvino_2021.2_6shave.blob file) to work - you can download it from [here](#)

Source code

Also available on GitHub

```
1  #!/usr/bin/env python3
2
3  from pathlib import Path
4  import sys
5  import cv2
6  import depthai as dai
7  import numpy as np
8
9  # Get argument first
10 nnPath = str((Path(__file__).parent / Path('models/mobilenet-ssd_openvino_2021.2_
    ↳6shave.blob')).resolve().absolute())
11 if len(sys.argv) > 1:
12     nnPath = sys.argv[1]
13
14 if not Path(nnPath).exists():
15     import sys
16     raise FileNotFoundError(f'Required file/s not found, please run "{sys.executable}
    ↳install_requirements.py"')
17
18 pipeline = dai.Pipeline()
19
20 cam = pipeline.createColorCamera()
21 cam.setBoardSocket(dai.CameraBoardSocket.RGB)
22 cam.setResolution(dai.ColorCameraProperties.SensorResolution.THE_1080_P)
23
```

(continues on next page)

(continued from previous page)

```

24 videoEncoder = pipeline.createVideoEncoder()
25 videoEncoder.setDefaultProfilePreset(1920, 1080, 30, dai.VideoEncoderProperties.
   ↳ Profile.H265_MAIN)
26 cam.video.link(videoEncoder.input)
27
28 videoOut = pipeline.createXLinkOut()
29 videoOut.setStreamName('h265')
30 videoEncoder.bitstream.link(videoOut.input)
31
32 camRight = pipeline.createMonoCamera()
33 camRight.setBoardSocket(dai.CameraBoardSocket.RIGHT)
34 camRight.setResolution(dai.MonoCameraProperties.SensorResolution.THE_720_P)
35
36 nn = pipeline.createMobileNetDetectionNetwork()
37 nn.setConfidenceThreshold(0.5)
38 nn.setBlobPath(nnPath)
39 nn.setNumInferenceThreads(2)
40 nn.input.setBlocking(False)
41
42 manip = pipeline.createImageManip()
43 manip.initialConfig.setResize(300, 300)
44 # The NN model expects BGR input. By default ImageManip output type would be same as
   ↳ input (gray in this case)
45 manip.initialConfig.setFrameType(dai.ImgFrame.Type.BGR888p)
46 camRight.out.link(manip.inputImage)
47 manip.out.link(nn.input)
48
49 xoutRight = pipeline.createXLinkOut()
50 xoutRight.setStreamName("right")
51 camRight.out.link(xoutRight.input)
52
53 manipOut = pipeline.createXLinkOut()
54 manipOut.setStreamName("manip")
55 manip.out.link(manipOut.input)
56
57 nnOut = pipeline.createXLinkOut()
58 nnOut.setStreamName("nn")
59 nn.out.link(nnOut.input)
60
61 # MobilenetSSD label texts
62 labelMap = ["background", "aeroplane", "bicycle", "bird", "boat", "bottle", "bus",
   ↳ "car", "cat", "chair", "cow",
63           "diningtable", "dog", "horse", "motorbike", "person", "pottedplant",
   ↳ "sheep", "sofa", "train", "tvmonitor"]
64
65
66 # Pipeline is defined, now we can connect to the device
67 with dai.Device(pipeline) as device:
68     # Start pipeline
69     device.startPipeline()
70
71     queue_size = 8
72     qRight = device.getOutputQueue("right", queue_size)
73     qManip = device.getOutputQueue("manip", queue_size)
74     qDet = device.getOutputQueue("nn", queue_size)
75     qRgbEnc = device.getOutputQueue('h265', maxSize=30, blocking=True)
76

```

(continues on next page)

(continued from previous page)

```

77     frame = None
78     frameManip = None
79     detections = []
80     offsetX = (camRight.getResolutionWidth() - camRight.getResolutionHeight()) // 2
81     croppedFrame = np.zeros((camRight.getResolutionHeight(), camRight.
↪ getResolutionHeight()))
82
83     def frameNorm(frame, bbox):
84         normVals = np.full(len(bbox), frame.shape[0])
85         normVals[::2] = frame.shape[1]
86         return (np.clip(np.array(bbox), 0, 1) * normVals).astype(int)
87
88     videoFile = open('video.h265', 'wb')
89     cv2.namedWindow("right", cv2.WINDOW_NORMAL)
90     cv2.namedWindow("manip", cv2.WINDOW_NORMAL)
91
92     while True:
93         inRight = qRight.tryGet()
94         inManip = qManip.tryGet()
95         inDet = qDet.tryGet()
96
97         while qRgbEnc.has():
98             qRgbEnc.get().getData().tofile(videoFile)
99
100         if inRight is not None:
101             frame = inRight.getCvFrame()
102
103         if inManip is not None:
104             frameManip = inManip.getCvFrame()
105
106         if inDet is not None:
107             detections = inDet.detections
108
109         if frame is not None:
110             for detection in detections:
111                 bbox = frameNorm(croppedFrame, (detection.xmin, detection.ymin,
↪ detection.xmax, detection.ymax))
112                 bbox[::2] += offsetX
113                 cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (255, 0,
↪ 0), 2)
114                 cv2.putText(frame, labelMap[detection.label], (bbox[0] + 10, bbox[1]
↪ + 20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
115                 cv2.putText(frame, f"{int(detection.confidence * 100)}%", (bbox[0] +
↪ 10, bbox[1] + 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
116                 cv2.imshow("right", frame)
117
118         if frameManip is not None:
119             for detection in detections:
120                 bbox = frameNorm(frameManip, (detection.xmin, detection.ymin,
↪ detection.xmax, detection.ymax))
121                 cv2.rectangle(frameManip, (bbox[0], bbox[1]), (bbox[2], bbox[3]),
↪ (255, 0, 0), 2)
122                 cv2.putText(frameManip, labelMap[detection.label], (bbox[0] + 10,
↪ bbox[1] + 20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
123                 cv2.putText(frameManip, f"{int(detection.confidence * 100)}%",
↪ (bbox[0] + 10, bbox[1] + 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
124                 cv2.imshow("manip", frameManip)

```

(continues on next page)

(continued from previous page)

```

125         if cv2.waitKey(1) == ord('q'):
126             break
127
128     videoFile.close()
129
130     print("To view the encoded data, convert the stream file (.h265) into a video_
131 ↪file (.mp4) using a command below:")
132     print("ffmpeg -framerate 30 -i video.h265 -c copy video.mp4")

```

We're always happy to help with code or other questions you might have.

3.11.12 12 - RGB Encoding & Mono with MobilenetSSD & Depth

This example shows how to configure the depthai video encoder in h.265 format to encode the RGB camera input at Full-HD resolution at 30FPS, and transfers the encoded video over XLINK to the host, saving it to disk as a video file. At the same time, a MobileNetV2SSD network is ran on the frames from right grayscale camera, while the application also displays the depth map produced by both of the grayscale cameras. Note that disparity is used in this case, as it colorizes in a more intuitive way.

Pressing Ctrl+C will stop the recording and then convert it using ffmpeg into an mp4 to make it playable. Note that ffmpeg will need to be installed and runnable for the conversion to mp4 to succeed.

Be careful, this example saves encoded video to your host storage. So if you leave it running, you could fill up your storage on your host.

Demo

Setup

Please run the following command to install the required dependencies

```

python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai

```

For additional information, please follow [installation guide](#)

This example also requires MobilenetSSD blob (mobilenet-ssd_openvino_2021.2_6shave.blob file) to work - you can download it from [here](#)

Source code

Also available on GitHub

```

1  #!/usr/bin/env python3
2
3  from pathlib import Path
4  import sys
5  import cv2
6  import depthai as dai
7  import numpy as np
8

```

(continues on next page)

(continued from previous page)

```

9  flipRectified = True
10
11  # Get argument first
12  nnPath = str((Path(__file__).parent / Path('models/mobilenet-ssd_openvino_2021.2_
    ↳6shave.blob')).resolve().absolute())
13  if len(sys.argv) > 1:
14      nnPath = sys.argv[1]
15
16  if not Path(nnPath).exists():
17      import sys
18      raise FileNotFoundError(f'Required file/s not found, please run "{sys.executable} _
    ↳install_requirements.py"')
19
20  pipeline = dai.Pipeline()
21
22  cam = pipeline.createColorCamera()
23  cam.setBoardSocket(dai.CameraBoardSocket.RGB)
24  cam.setResolution(dai.ColorCameraProperties.SensorResolution.THE_1080_P)
25
26  videoEncoder = pipeline.createVideoEncoder()
27  videoEncoder.setDefaultProfilePreset(1920, 1080, 30, dai.VideoEncoderProperties.
    ↳Profile.H265_MAIN)
28  cam.video.link(videoEncoder.input)
29
30  videoOut = pipeline.createXLinkOut()
31  videoOut.setStreamName('h265')
32  videoEncoder.bitstream.link(videoOut.input)
33  camLeft = pipeline.createMonoCamera()
34  camLeft.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
35  camLeft.setBoardSocket(dai.CameraBoardSocket.LEFT)
36
37  camRight = pipeline.createMonoCamera()
38  camRight.setBoardSocket(dai.CameraBoardSocket.RIGHT)
39  camRight.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
40
41  depth = pipeline.createStereoDepth()
42  depth.setConfidenceThreshold(255)
43  # Note: the rectified streams are horizontally mirrored by default
44  depth.setOutputRectified(True)
45  depth.setRectifyMirrorFrame(False)
46  depth.setRectifyEdgeFillColor(0) # Black, to better see the cutout
47  camLeft.out.link(depth.left)
48  camRight.out.link(depth.right)
49  # Disparity range is 0..95, used for normalization
50  disparity_multiplier = 255 / 95
51
52  disparityOut = pipeline.createXLinkOut()
53  disparityOut.setStreamName("disparity")
54  depth.disparity.link(disparityOut.input)
55
56  nn = pipeline.createMobileNetDetectionNetwork()
57  nn.setConfidenceThreshold(0.5)
58  nn.setBlobPath(nnPath)
59  nn.setNumInferenceThreads(2)
60  nn.input.setBlocking(False)
61
62  manip = pipeline.createImageManip()

```

(continues on next page)

(continued from previous page)

```

63 manip.initialConfig.setResize(300, 300)
64 # The NN model expects BGR input. By default ImageManip output type would be same as
   ↳ input (gray in this case)
65 manip.initialConfig.setFrameType(dai.ImgFrame.Type.BGR888p)
66 depth.rectifiedRight.link(manip.inputImage)
67 manip.out.link(nn.input)
68
69 xoutRight = pipeline.createXLinkOut()
70 xoutRight.setStreamName("right")
71 camRight.out.link(xoutRight.input)
72
73 manipOut = pipeline.createXLinkOut()
74 manipOut.setStreamName("manip")
75 manip.out.link(manipOut.input)
76
77 nnOut = pipeline.createXLinkOut()
78 nnOut.setStreamName("nn")
79 nn.out.link(nnOut.input)
80
81 # MobilenetSSD label texts
82 labelMap = ["background", "aeroplane", "bicycle", "bird", "boat", "bottle", "bus",
   ↳ "car", "cat", "chair", "cow",
83             "diningtable", "dog", "horse", "motorbike", "person", "pottedplant",
   ↳ "sheep", "sofa", "train", "tvmonitor"]
84
85
86 # Pipeline is defined, now we can connect to the device
87 with dai.Device(pipeline) as device:
88     # Start pipeline
89     device.startPipeline()
90
91     queueSize = 8
92     qRight = device.getOutputQueue("right", queueSize)
93     qDisparity = device.getOutputQueue("disparity", queueSize)
94     qManip = device.getOutputQueue("manip", queueSize)
95     qDet = device.getOutputQueue("nn", queueSize)
96     qRgbEnc = device.getOutputQueue('h265', maxSize=30, blocking=True)
97
98     frame = None
99     frameManip = None
100    frameDisparity = None
101    detections = []
102    offsetX = (camRight.getResolutionWidth() - camRight.getResolutionHeight()) // 2
103    croppedFrame = np.zeros((camRight.getResolutionHeight(), camRight.
   ↳ getResolutionHeight()))
104
105    def frameNorm(frame, bbox):
106        normVals = np.full(len(bbox), frame.shape[0])
107        normVals[::2] = frame.shape[1]
108        return (np.clip(np.array(bbox), 0, 1) * normVals).astype(int)
109
110    videoFile = open('video.h265', 'wb')
111    cv2.namedWindow("right", cv2.WINDOW_NORMAL)
112    cv2.namedWindow("manip", cv2.WINDOW_NORMAL)
113    cv2.namedWindow("disparity", cv2.WINDOW_NORMAL)
114
115    while True:

```

(continues on next page)

(continued from previous page)

```

116     inRight = qRight.tryGet()
117     inManip = qManip.tryGet()
118     inDet = qDet.tryGet()
119     inDisparity = qDisparity.tryGet()
120
121     while qRgbEnc.has():
122         qRgbEnc.get().getData().tofile(videoFile)
123
124     if inRight is not None:
125         frame = cv2.flip(inRight.getCvFrame(), 1)
126         if flipRectified:
127             frame = cv2.flip(frame, 1)
128
129     if inManip is not None:
130         frameManip = inManip.getCvFrame()
131
132     if inDisparity is not None:
133         # Flip disparity frame, normalize it and apply color map for better_
↪ visualization
134         frameDisparity = inDisparity.getFrame()
135         if flipRectified:
136             frameDisparity = cv2.flip(frameDisparity, 1)
137         frameDisparity = (frameDisparity * disparity_multiplier).astype(np.uint8)
138         frameDisparity = cv2.applyColorMap(frameDisparity, cv2.COLORMAP_JET)
139
140     if inDet is not None:
141         detections = inDet.detections
142
143     if frame is not None:
144         for detection in detections:
145             bbox = frameNorm(croppedFrame, (detection.xmin, detection.ymin,
↪ detection.xmax, detection.ymax))
146             bbox[::2] += offsetX
147             cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (255, 0,
↪ 0), 2)
148             cv2.putText(frame, labelMap[detection.label], (bbox[0] + 10, bbox[1]
↪ + 20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
149             cv2.putText(frame, f"{int(detection.confidence * 100)}%", (bbox[0] +
↪ 10, bbox[1] + 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
150             cv2.imshow("right", frame)
151
152     if frameDisparity is not None:
153         for detection in detections:
154             bbox = frameNorm(croppedFrame, (detection.xmin, detection.ymin,
↪ detection.xmax, detection.ymax))
155             bbox[::2] += offsetX
156             cv2.rectangle(frameDisparity, (bbox[0], bbox[1]), (bbox[2], bbox[3]),
↪ (255, 0, 0), 2)
157             cv2.putText(frameDisparity, labelMap[detection.label], (bbox[0] + 10,
↪ bbox[1] + 20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
158             cv2.putText(frameDisparity, f"{int(detection.confidence * 100)}%",
↪ (bbox[0] + 10, bbox[1] + 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
159             cv2.imshow("disparity", frameDisparity)
160
161     if frameManip is not None:
162         for detection in detections:
163             bbox = frameNorm(frameManip, (detection.xmin, detection.ymin,
↪ detection.xmax, detection.ymax))

```

(continues on next page)

(continued from previous page)

```

164         cv2.rectangle(frameManip, (bbox[0], bbox[1]), (bbox[2], bbox[3]),
↳ (255, 0, 0), 2)
165         cv2.putText(frameManip, labelMap[detection.label], (bbox[0] + 10,
↳ bbox[1] + 20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
166         cv2.putText(frameManip, f"{int(detection.confidence * 100)}%",
↳ (bbox[0] + 10, bbox[1] + 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
167         cv2.imshow("manip", frameManip)
168
169         if cv2.waitKey(1) == ord('q'):
170             break
171
172     videoFile.close()
173
174     print("To view the encoded data, convert the stream file (.h265) into a video
↳ file (.mp4) using a command below:")
175     print("ffmpeg -framerate 30 -i video.h265 -c copy video.mp4")

```

We're always happy to help with code or other questions you might have.

3.11.13 13 - Encoding Max Limit

This example shows how to set up the encoder node to encode the RGB camera and both grayscale cameras (of DepthAI/OAK-D) at the same time, having all encoder parameters set to maximum quality and FPS. The RGB is set to 4K (3840x2160) and the grayscale are set to 1280x720 each, all at 25FPS. Each encoded video stream is transferred over XLINK and saved to a respective file.

Pressing Ctrl+C will stop the recording and then convert it using ffmpeg into an mp4 to make it playable. Note that ffmpeg will need to be installed and runnable for the conversion to mp4 to succeed.

Be careful, this example saves encoded video to your host storage. So if you leave it running, you could fill up your storage on your host.

Demo

Setup

Please run the following command to install the required dependencies

```

python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai

```

For additional information, please follow [installation guide](#)

Source code

Also available on [GitHub](#)

```

1  #!/usr/bin/env python3
2
3  import depthai as dai
4
5  pipeline = dai.Pipeline()
6
7  # Nodes
8  colorCam = pipeline.createColorCamera()
9  colorCam.setResolution(dai.ColorCameraProperties.SensorResolution.THE_4_K)
10 monoCam = pipeline.createMonoCamera()
11 monoCam2 = pipeline.createMonoCamera()
12 ve1 = pipeline.createVideoEncoder()
13 ve2 = pipeline.createVideoEncoder()
14 ve3 = pipeline.createVideoEncoder()
15
16 ve1Out = pipeline.createXLinkOut()
17 ve2Out = pipeline.createXLinkOut()
18 ve3Out = pipeline.createXLinkOut()
19
20 # Properties
21 monoCam.setBoardSocket(dai.CameraBoardSocket.LEFT)
22 monoCam2.setBoardSocket(dai.CameraBoardSocket.RIGHT)
23 ve1Out.setStreamName('ve1Out')
24 ve2Out.setStreamName('ve2Out')
25 ve3Out.setStreamName('ve3Out')
26
27 # Setting to 26fps will trigger error
28 ve1.setDefaultProfilePreset(1280, 720, 25, dai.VideoEncoderProperties.Profile.H264_
↳MAIN)
29 ve2.setDefaultProfilePreset(3840, 2160, 25, dai.VideoEncoderProperties.Profile.H265_
↳MAIN)
30 ve3.setDefaultProfilePreset(1280, 720, 25, dai.VideoEncoderProperties.Profile.H264_
↳MAIN)
31
32 # Link nodes
33 monoCam.out.link(ve1.input)
34 colorCam.video.link(ve2.input)
35 monoCam2.out.link(ve3.input)
36
37 ve1.bitstream.link(ve1Out.input)
38 ve2.bitstream.link(ve2Out.input)
39 ve3.bitstream.link(ve3Out.input)
40
41
42 # Pipeline is defined, now we can connect to the device
43 with dai.Device(pipeline) as dev:
44
45     # Prepare data queues
46     outQ1 = dev.getOutputQueue('ve1Out', maxSize=30, blocking=True)
47     outQ2 = dev.getOutputQueue('ve2Out', maxSize=30, blocking=True)
48     outQ3 = dev.getOutputQueue('ve3Out', maxSize=30, blocking=True)
49
50     # Start the pipeline
51     dev.startPipeline()

```

(continues on next page)

(continued from previous page)

```

52
53     # Processing loop
54     with open('mono1.h264', 'wb') as fileMono1H264, open('color.h265', 'wb') as
↪fileColorH265, open('mono2.h264', 'wb') as fileMono2H264:
55         print("Press Ctrl+C to stop encoding...")
56         while True:
57             try:
58                 # Empty each queue
59                 while outQ1.has():
60                     outQ1.get().getData().tofile(fileMono1H264)
61
62                 while outQ2.has():
63                     outQ2.get().getData().tofile(fileColorH265)
64
65                 while outQ3.has():
66                     outQ3.get().getData().tofile(fileMono2H264)
67             except KeyboardInterrupt:
68                 break
69
70         print("To view the encoded data, convert the stream file (.h264/.h265) into a
↪video file (.mp4), using commands below:")
71         cmd = "ffmpeg -framerate 25 -i {} -c copy {}"
72         print(cmd.format("mono1.h264", "mono1.mp4"))
73         print(cmd.format("mono2.h264", "mono2.mp4"))
74         print(cmd.format("color.h265", "color.mp4"))

```

We're always happy to help with code or other questions you might have.

3.11.14 14.1 - Color Camera Control

This example shows how to control the device-side crop and camera triggers. An output is a displayed RGB cropped frame, that can be manipulated using the following keys:

1. *a* will move the crop left
2. *d* will move the crop right
3. *w* will move the crop up
4. *s* will move the crop down
5. *c* will trigger a *still* event, causing the current frame to be captured and sent over *still* output from camera node
6. *t* will trigger autofocus
7. *f* will trigger autofocus continuously
8. *e* will trigger autoexposure
9. *i* and *o* will decrease/increase the exposure time
10. *k* and *l* will decrease/increase the sensitivity iso
11. *,* and *.* will decrease/increase the focus range

Demo

Setup

Please run the following command to install the required dependencies

```
python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai
```

For additional information, please follow *installation guide*

Source code

Also available on GitHub

```

1  #!/usr/bin/env python3
2
3  """
4  This example shows usage of Camera Control message as well as ColorCamera configInput
5  ↳to change crop x and y
6  Uses 'WASD' controls to move the crop window, 'C' to capture a still image, 'T' to
7  ↳trigger autofocus, 'IOKL,.'
8  for manual exposure/focus:
9      Control:      key[dec/inc]  min..max
10     exposure time:  I   O        1..33000 [us]
11     sensitivity iso: K   L        100..1600
12     focus:         ,   .         0..255 [far..near]
13 To go back to auto controls:
14     'E' - autoexposure
15     'F' - autofocus (continuous)
16 """
17
18 import depthai as dai
19 import cv2
20
21 # Step size ('W','A','S','D' controls)
22 STEP_SIZE = 8
23 # Manual exposure/focus set step
24 EXP_STEP = 500 # us
25 ISO_STEP = 50
26 LENS_STEP = 3
27
28 pipeline = dai.Pipeline()
29
30 # Nodes
31 colorCam = pipeline.createColorCamera()
32 controlIn = pipeline.createXLinkIn()
33 configIn = pipeline.createXLinkIn()
34 videoEncoder = pipeline.createVideoEncoder()
35 stillEncoder = pipeline.createVideoEncoder()
36 videoMjpegOut = pipeline.createXLinkOut()
37 stillMjpegOut = pipeline.createXLinkOut()
38 previewOut = pipeline.createXLinkOut()

```

(continues on next page)

(continued from previous page)

```

39 # Properties
40 colorCam.setVideoSize(640, 360)
41 colorCam.setPreviewSize(300, 300)
42 controlIn.setStreamName('control')
43 configIn.setStreamName('config')
44 videoEncoder.setDefaultProfilePreset(colorCam.getVideoSize(), colorCam.getFps(), dai.
↳VideoEncoderProperties.Profile.MJPEG)
45 stillEncoder.setDefaultProfilePreset(colorCam.getStillSize(), 1, dai.
↳VideoEncoderProperties.Profile.MJPEG)
46 videoMjpegOut.setStreamName('video')
47 stillMjpegOut.setStreamName('still')
48 previewOut.setStreamName('preview')
49
50
51 # Link nodes
52 colorCam.video.link(videoEncoder.input)
53 colorCam.still.link(stillEncoder.input)
54 colorCam.preview.link(previewOut.input)
55 controlIn.out.link(colorCam.inputControl)
56 configIn.out.link(colorCam.inputConfig)
57 videoEncoder.bitstream.link(videoMjpegOut.input)
58 stillEncoder.bitstream.link(stillMjpegOut.input)
59
60
61 def clamp(num, v0, v1):
62     return max(v0, min(num, v1))
63
64
65 # Pipeline is defined, now we can connect to the device
66 with dai.Device(pipeline) as dev:
67
68     # Get data queues
69     controlQueue = dev.getInputQueue('control')
70     configQueue = dev.getInputQueue('config')
71     previewQueue = dev.getOutputQueue('preview')
72     videoQueue = dev.getOutputQueue('video')
73     stillQueue = dev.getOutputQueue('still')
74
75     # Start pipeline
76     dev.startPipeline()
77
78     # Max cropX & cropY
79     maxCropX = (colorCam.getResolutionWidth() - colorCam.getVideoWidth()) / colorCam.
↳getResolutionWidth()
80     maxCropY = (colorCam.getResolutionHeight() - colorCam.getVideoHeight()) /
↳colorCam.getResolutionHeight()
81
82     # Default crop
83     cropX = 0
84     cropY = 0
85     sendCamConfig = True
86
87     # Defaults and limits for manual focus/exposure controls
88     lensPos = 150
89     lensMin = 0
90     lensMax = 255
91

```

(continues on next page)

(continued from previous page)

```

92     expTime = 20000
93     expMin = 1
94     expMax = 33000
95
96     sensIso = 800
97     sensMin = 100
98     sensMax = 1600
99
100    while True:
101
102        previewFrames = previewQueue.tryGetAll()
103        for previewFrame in previewFrames:
104            cv2.imshow('preview', previewFrame.getData().reshape(previewFrame.
↳getWidth(), previewFrame.getHeight(), 3))
105
106        videoFrames = videoQueue.tryGetAll()
107        for videoFrame in videoFrames:
108            # Decode JPEG
109            frame = cv2.imdecode(videoFrame.getData(), cv2.IMREAD_UNCHANGED)
110            # Display
111            cv2.imshow('video', frame)
112
113            # Send new cfg to camera
114            if sendCamConfig:
115                cfg = dai.ImageManipConfig()
116                cfg.setCropRect(cropX, cropY, 0, 0)
117                configQueue.send(cfg)
118                print('Sending new crop - x: ', cropX, ' y: ', cropY)
119                sendCamConfig = False
120
121        stillFrames = stillQueue.tryGetAll()
122        for stillFrame in stillFrames:
123            # Decode JPEG
124            frame = cv2.imdecode(stillFrame.getData(), cv2.IMREAD_UNCHANGED)
125            # Display
126            cv2.imshow('still', frame)
127
128
129        # Update screen
130        key = cv2.waitKey(1)
131        if key == ord('q'):
132            break
133        elif key == ord('c'):
134            ctrl = dai.CameraControl()
135            ctrl.setCaptureStill(True)
136            controlQueue.send(ctrl)
137        elif key == ord('t'):
138            print("Autofocus trigger (and disable continuous)")
139            ctrl = dai.CameraControl()
140            ctrl.setAutoFocusMode(dai.CameraControl.AutoFocusMode.AUTO)
141            ctrl.setAutoFocusTrigger()
142            controlQueue.send(ctrl)
143        elif key == ord('f'):
144            print("Autofocus enable, continuous")
145            ctrl = dai.CameraControl()
146            ctrl.setAutoFocusMode(dai.CameraControl.AutoFocusMode.CONTINUOUS_VIDEO)
147            controlQueue.send(ctrl)

```

(continues on next page)

(continued from previous page)

```

148     elif key == ord('e'):
149         print("Autoexposure enable")
150         ctrl = dai.CameraControl()
151         ctrl.setAutoExposureEnable()
152         controlQueue.send(ctrl)
153     elif key in [ord(','), ord('.')]:
154         if key == ord(','): lensPos -= LENS_STEP
155         if key == ord('.'): lensPos += LENS_STEP
156         lensPos = clamp(lensPos, lensMin, lensMax)
157         print("Setting manual focus, lens position:", lensPos)
158         ctrl = dai.CameraControl()
159         ctrl.setManualFocus(lensPos)
160         controlQueue.send(ctrl)
161     elif key in [ord('i'), ord('o'), ord('k'), ord('l')]:
162         if key == ord('i'): expTime -= EXP_STEP
163         if key == ord('o'): expTime += EXP_STEP
164         if key == ord('k'): sensIso -= ISO_STEP
165         if key == ord('l'): sensIso += ISO_STEP
166         expTime = clamp(expTime, expMin, expMax)
167         sensIso = clamp(sensIso, sensMin, sensMax)
168         print("Setting manual exposure, time:", expTime, "iso:", sensIso)
169         ctrl = dai.CameraControl()
170         ctrl.setManualExposure(expTime, sensIso)
171         controlQueue.send(ctrl)
172     elif key in [ord('w'), ord('a'), ord('s'), ord('d')]:
173         if key == ord('a'):
174             cropX = cropX - (maxCropX / colorCam.getResolutionWidth()) * STEP_SIZE
175             if cropX < 0: cropX = maxCropX
176         elif key == ord('d'):
177             cropX = cropX + (maxCropX / colorCam.getResolutionWidth()) * STEP_SIZE
178             if cropX > maxCropX: cropX = 0
179         elif key == ord('w'):
180             cropY = cropY - (maxCropY / colorCam.getResolutionHeight()) * STEP_
181             if cropY < 0: cropY = maxCropY
182         elif key == ord('s'):
183             cropY = cropY + (maxCropY / colorCam.getResolutionHeight()) * STEP_
184             if cropY > maxCropY: cropY = 0
185         sendCamConfig = True

```

We're always happy to help with code or other questions you might have.

3.11.15 14.2 - Mono Camera Control

This example shows how to control the device-side crop and camera triggers. Two output is a displayed mono cropped frame, that can be manipulated using the following keys:

1. *a* will move the crop left
2. *d* will move the crop right
3. *w* will move the crop up
4. *s* will move the crop down
5. *e* will trigger autoexposure

6. *i* and *o* will decrease/increase the exposure time
7. *k* and *l* will decrease/increase the sensitivity iso

Demo

Setup

Please run the following command to install the required dependencies

```
python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai
```

For additional information, please follow *installation guide*

Source code

Also available on GitHub

```
1  #!/usr/bin/env python3
2
3  """
4  This example shows usage of mono camera in crop mode with the possibility to move the
5  ↳ crop.
6  Uses 'WASD' controls to move the crop window, 'T' to trigger autofocus, 'IOKL,.' for
7  ↳ manual exposure/focus:
8      Control:      key[dec/inc]  min..max
9      exposure time:   I    O      1..33000 [us]
10     sensitivity iso:  K    L      100..1600
11 To go back to auto controls:
12     'E' - autoexposure
13 """
14
15 import cv2
16 import depthai as dai
17
18 # Step size ('W', 'A', 'S', 'D' controls)
19 stepSize = 0.02
20 # Manual exposure/focus set step
21 expStep = 500 # us
22 isoStep = 50
23
24 # Start defining a pipeline
25 pipeline = dai.Pipeline()
26
27 # Define a source - two mono (grayscale) camera
28 camRight = pipeline.createMonoCamera()
29 camRight.setBoardSocket(dai.CameraBoardSocket.RIGHT)
30 camRight.setResolution(dai.MonoCameraProperties.SensorResolution.THE_720_P)
31 camLeft = pipeline.createMonoCamera()
32 camLeft.setBoardSocket(dai.CameraBoardSocket.LEFT)
33 camLeft.setResolution(dai.MonoCameraProperties.SensorResolution.THE_720_P)
34
35 # Crop range
```

(continues on next page)

(continued from previous page)

```

35 topLeft = dai.Point2f(0.4, 0.4)
36 bottomRight = dai.Point2f(0.6, 0.6)
37
38 manipRight = pipeline.createImageManip()
39 manipRight.initialConfig.setCropRect(topLeft.x, topLeft.y, bottomRight.x, bottomRight.
↳y)
40 manipLeft = pipeline.createImageManip()
41 manipLeft.initialConfig.setCropRect(topLeft.x, topLeft.y, bottomRight.x, bottomRight.
↳y)
42 manipRight.setMaxOutputFrameSize(camRight.getResolutionHeight()*camRight.
↳getResolutionWidth()*3)
43
44 # Camera movement config (wasd)
45 configIn = pipeline.createXLinkIn()
46 configIn.setStreamName('config')
47 configIn.out.link(manipRight.inputConfig)
48 configIn.out.link(manipLeft.inputConfig)
49
50 # Camera control (exp, iso, focus)
51 controlIn = pipeline.createXLinkIn()
52 controlIn.setStreamName('control')
53 controlIn.out.link(camRight.inputControl)
54 controlIn.out.link(camLeft.inputControl)
55
56 # Linking with USB
57 camRight.out.link(manipRight.inputImage)
58 camLeft.out.link(manipLeft.inputImage)
59
60 # Create outputs
61 manipOutRight = pipeline.createXLinkOut()
62 manipOutRight.setStreamName("right")
63 manipRight.out.link(manipOutRight.input)
64
65 manipOutLeft = pipeline.createXLinkOut()
66 manipOutLeft.setStreamName("left")
67 manipLeft.out.link(manipOutLeft.input)
68
69 def clamp(num, v0, v1):
70     return max(v0, min(num, v1))
71
72 # Pipeline defined, now the device is connected to
73 with dai.Device(pipeline) as device:
74     # Start pipeline
75     device.startPipeline()
76
77     # Output queues will be used to get the grayscale frames
78     qRight = device.getOutputQueue(manipOutRight.getStreamName(), maxSize=4,
↳blocking=False)
79     qLeft = device.getOutputQueue(manipOutLeft.getStreamName(), maxSize=4,
↳blocking=False)
80     configQueue = device.getInputQueue(configIn.getStreamName())
81     controlQueue = device.getInputQueue(controlIn.getStreamName())
82
83     def displayFrame(name, frame):
84         cv2.imshow(name, frame)
85
86     sendCamConfig = False

```

(continues on next page)

(continued from previous page)

```

87
88     # Defaults and limits for manual focus/exposure controls
89     expTime = 20000
90     expMin = 1
91     expMax = 33000
92
93     sensIso = 800
94     sensMin = 100
95     sensMax = 1600
96
97     while True:
98         inRight = qRight.get()
99         inLeft = qLeft.get()
100         frameRight = inRight.getCvFrame()
101         frameLeft = inLeft.getCvFrame()
102         displayFrame("right", frameRight)
103         displayFrame("left", frameLeft)
104
105         # Update screen
106         key = cv2.waitKey(1)
107         if key == ord('q'):
108             break
109         elif key == ord('c'):
110             ctrl = dai.CameraControl()
111             ctrl.setCaptureStill(True)
112             controlQueue.send(ctrl)
113         elif key == ord('e'):
114             print("Autoexposure enable")
115             ctrl = dai.CameraControl()
116             ctrl.setAutoExposureEnable()
117             controlQueue.send(ctrl)
118         elif key in [ord('i'), ord('o'), ord('k'), ord('l')]:
119             if key == ord('i'): expTime -= expStep
120             if key == ord('o'): expTime += expStep
121             if key == ord('k'): sensIso -= isoStep
122             if key == ord('l'): sensIso += isoStep
123             expTime = clamp(expTime, expMin, expMax)
124             sensIso = clamp(sensIso, sensMin, sensMax)
125             print("Setting manual exposure, time:", expTime, "iso:", sensIso)
126             ctrl = dai.CameraControl()
127             ctrl.setManualExposure(expTime, sensIso)
128             controlQueue.send(ctrl)
129         elif key == ord('w'):
130             if topLeft.y - stepSize >= 0:
131                 topLeft.y -= stepSize
132                 bottomRight.y -= stepSize
133                 sendCamConfig = True
134         elif key == ord('a'):
135             if topLeft.x - stepSize >= 0:
136                 topLeft.x -= stepSize
137                 bottomRight.x -= stepSize
138                 sendCamConfig = True
139         elif key == ord('s'):
140             if bottomRight.y + stepSize <= 1:
141                 topLeft.y += stepSize
142                 bottomRight.y += stepSize
143                 sendCamConfig = True

```

(continues on next page)

(continued from previous page)

```

144         elif key == ord('d'):
145             if bottomRight.x + stepSize <= 1:
146                 topLeft.x += stepSize
147                 bottomRight.x += stepSize
148                 sendCamConfig = True
149
150
151         if sendCamConfig:
152             cfg = dai.ImageManipConfig()
153             cfg.setCropRect(topLeft.x, topLeft.y, bottomRight.x, bottomRight.y)
154             configQueue.send(cfg)
155             sendCamConfig = False

```

We're always happy to help with code or other questions you might have.

3.11.16 14.3 - Depth Crop Control

This example shows usage of depth camera in crop mode with the possibility to move the crop. You can manipulate the movement of the cropped frame by using the following keys:

1. *a* will move the crop left
2. *d* will move the crop right
3. *w* will move the crop up
4. *s* will move the crop down

Demo

Setup

Please run the following command to install the required dependencies

```

python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai

```

For additional information, please follow [installation guide](#)

Source code

Also available on GitHub

```

1  #!/usr/bin/env python3
2
3  """
4  This example shows usage of depth camera in crop mode with the possibility to move_
   ↳ the crop.
5  Use 'WASD' in order to do it.
6  """
7
8  import cv2
9  import depthai as dai

```

(continues on next page)

(continued from previous page)

```

10 import numpy as np
11
12 stepSize = 0.02
13
14 # Start defining a pipeline
15 pipeline = dai.Pipeline()
16
17 # Define a source - two mono (grayscale) cameras
18 left = pipeline.createMonoCamera()
19 left.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
20 left.setBoardSocket(dai.CameraBoardSocket.LEFT)
21
22 right = pipeline.createMonoCamera()
23 right.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
24 right.setBoardSocket(dai.CameraBoardSocket.RIGHT)
25
26 # Crop range
27 topLeft = dai.Point2f(0.4, 0.4)
28 bottomRight = dai.Point2f(0.6, 0.6)
29
30 manip = pipeline.createImageManip()
31 manip.initialConfig.setCropRect(topLeft.x, topLeft.y, bottomRight.x, bottomRight.y)
32 manip.setMaxOutputFrameSize(right.getResolutionHeight()*right.getResolutionWidth()*3)
33
34
35 # Create a node that will produce the depth map
36 stereo = pipeline.createStereoDepth()
37 stereo.setConfidenceThreshold(200)
38 stereo.setOutputDepth(True)
39
40 left.out.link(stereo.left)
41 right.out.link(stereo.right)
42
43
44 # Control movement
45 controlIn = pipeline.createXLinkIn()
46 controlIn.setStreamName('control')
47 controlIn.out.link(manip.inputConfig)
48
49 # Create outputs
50 xout = pipeline.createXLinkOut()
51 xout.setStreamName("depth")
52 stereo.depth.link(manip.inputImage)
53 manip.out.link(xout.input)
54
55 # Pipeline defined, now the device is connected to
56 with dai.Device(pipeline) as device:
57     # Start pipeline
58     device.startPipeline()
59
60     # Output queue will be used to get the depth frames from the outputs defined above
61     q = device.getOutputQueue(xout.getStreamName(), maxSize=4, blocking=False)
62
63     sendCamConfig = False
64
65     while True:
66         inDepth = q.get() # blocking call, will wait until a new data has arrived

```

(continues on next page)

(continued from previous page)

```

67         # data is originally represented as a flat 1D array, it needs to be converted_
↳into HxW form
68         depthFrame = inDepth.getFrame()
69         # frame is transformed, the color map will be applied to highlight the depth_
↳info
70         depthFrameColor = cv2.normalize(depthFrame, None, 255, 0, cv2.NORM_INF, cv2.
↳CV_8UC1)
71         depthFrameColor = cv2.equalizeHist(depthFrameColor)
72         depthFrameColor = cv2.applyColorMap(depthFrameColor, cv2.COLORMAP_HOT)
73         controlQueue = device.getInputQueue(controlIn.getStreamName())
74
75         # frame is ready to be shown
76         cv2.imshow("depth", depthFrameColor)
77
78         # Update screen
79         key = cv2.waitKey(1)
80         if key == ord('q'):
81             break
82         elif key == ord('w'):
83             if topLeft.y - stepSize >= 0:
84                 topLeft.y -= stepSize
85                 bottomRight.y -= stepSize
86                 sendCamConfig = True
87         elif key == ord('a'):
88             if topLeft.x - stepSize >= 0:
89                 topLeft.x -= stepSize
90                 bottomRight.x -= stepSize
91                 sendCamConfig = True
92         elif key == ord('s'):
93             if bottomRight.y + stepSize <= 1:
94                 topLeft.y += stepSize
95                 bottomRight.y += stepSize
96                 sendCamConfig = True
97         elif key == ord('d'):
98             if bottomRight.x + stepSize <= 1:
99                 topLeft.x += stepSize
100                 bottomRight.x += stepSize
101                 sendCamConfig = True
102
103
104         if sendCamConfig:
105             cfg = dai.ImageManipConfig()
106             cfg.setCropRect(topLeft.x, topLeft.y, bottomRight.x, bottomRight.y)
107             controlQueue.send(cfg)
108             sendCamConfig = False

```

We're always happy to help with code or other questions you might have.

3.11.17 15 - 4K RGB MobileNetSSD

This example shows how to run MobileNetv2SSD on the RGB input frame, and how to display both the RGB preview and the metadata results from the MobileNetv2SSD on the preview. The preview size is set to 4K resolution.

Demo

Setup

Please run the following command to install the required dependencies

```
python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai
```

For additional information, please follow [installation guide](#)

This example also requires MobilenetSSD blob (mobilenet-ssd_openvino_2021.2_5shave.blob file) to work - you can download it from [here](#)

Source code

Also available on GitHub

```
1  #!/usr/bin/env python3
2
3  from pathlib import Path
4  import sys
5  import cv2
6  import depthai as dai
7  import numpy as np
8
9  # Get argument first
10 nnPath = str((Path(__file__).parent / Path('models/mobilenet-ssd_openvino_2021.2_
    ↳5shave.blob')).resolve().absolute())
11 if len(sys.argv) > 1:
12     nnPath = sys.argv[1]
13
14 if not Path(nnPath).exists():
15     import sys
16     raise FileNotFoundError(f'Required file/s not found, please run "{sys.executable}
    ↳install_requirements.py"')
17
18 # Start defining a pipeline
19 pipeline = dai.Pipeline()
20
21 # Define a source - color camera
22 camRgb = pipeline.createColorCamera()
23 camRgb.setPreviewSize(300, 300)    # NN input
24 camRgb.setResolution(dai.ColorCameraProperties.SensorResolution.THE_4_K)
25 camRgb.setInterleaved(False)
26 camRgb.setPreviewKeepAspectRatio(False)
27
28 # Define a neural network that will make predictions based on the source frames
29 nn = pipeline.createMobileNetDetectionNetwork()
```

(continues on next page)

(continued from previous page)

```

30 nn.setConfidenceThreshold(0.5)
31 nn.setBlobPath(nnPath)
32 nn.setNumInferenceThreads(2)
33 nn.input.setBlocking(False)
34 camRgb.preview.link(nn.input)
35
36 # Create outputs
37 xoutVideo = pipeline.createXLinkOut()
38 xoutVideo.setStreamName("video")
39 camRgb.video.link(xoutVideo.input)
40
41 xoutPreview = pipeline.createXLinkOut()
42 xoutPreview.setStreamName("preview")
43 camRgb.preview.link(xoutPreview.input)
44
45 nnOut = pipeline.createXLinkOut()
46 nnOut.setStreamName("nn")
47 nn.out.link(nnOut.input)
48
49 # MobilenetSSD label texts
50 labelMap = ["background", "aeroplane", "bicycle", "bird", "boat", "bottle", "bus",
51 ↪ "car", "cat", "chair", "cow",
52 ↪ "diningtable", "dog", "horse", "motorbike", "person", "pottedplant",
53 ↪ "sheep", "sofa", "train", "tvmonitor"]
54
55 # Pipeline is defined, now we can connect to the device
56 with dai.Device(pipeline) as device:
57     # Start pipeline
58     device.startPipeline()
59
60     # Output queues will be used to get the frames and nn data from the outputs_
61     ↪defined above
62     qVideo = device.getOutputQueue(name="video", maxSize=4, blocking=False)
63     qPreview = device.getOutputQueue(name="preview", maxSize=4, blocking=False)
64     qDet = device.getOutputQueue(name="nn", maxSize=4, blocking=False)
65
66     previewFrame = None
67     videoFrame = None
68     detections = []
69
70     # nn data, being the bounding box locations, are in <0..1> range - they need to_
71     ↪be normalized with frame width/height
72     def frameNorm(frame, bbox):
73         normVals = np.full(len(bbox), frame.shape[0])
74         normVals[::2] = frame.shape[1]
75         return (np.clip(np.array(bbox), 0, 1) * normVals).astype(int)
76
77     def displayFrame(name, frame):
78         for detection in detections:
79             bbox = frameNorm(frame, (detection.xmin, detection.ymin, detection.xmax,
80 ↪detection.ymax))
81             ↪2)
82             cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (255, 0, 0),
83 ↪2)
84             cv2.putText(frame, labelMap[detection.label], (bbox[0] + 10, bbox[1] +
85 ↪20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
86             cv2.putText(frame, f"{int(detection.confidence * 100)}%", (bbox[0] + 10,
87 ↪bbox[1] + 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)

```

(continues on next page)

(continued from previous page)

```

79         cv2.imshow(name, frame)
80
81     cv2.namedWindow("video", cv2.WINDOW_NORMAL)
82     cv2.resizeWindow("video", 1280, 720)
83     print("Resize video window with mouse drag!")
84
85     while True:
86         # Instead of get (blocking), we use tryGet (nonblocking) which will return
87         ↪ the available data or None otherwise
88         inVideo = qVideo.tryGet()
89         inPreview = qPreview.tryGet()
90         inDet = qDet.tryGet()
91
92         if inVideo is not None:
93             videoFrame = inVideo.getCvFrame()
94
95         if inPreview is not None:
96             previewFrame = inPreview.getCvFrame()
97
98         if inDet is not None:
99             detections = inDet.detections
100
101         if videoFrame is not None:
102             displayFrame("video", videoFrame)
103
104         if previewFrame is not None:
105             displayFrame("preview", previewFrame)
106
107         if cv2.waitKey(1) == ord('q'):
108             break

```

We're always happy to help with code or other questions you might have.

3.11.18 16 - Device Queue Event

This example shows how to use `getQueueEvent` function in order to be notified when one of the packets from selected streams arrive

Demo

Setup

Please run the following command to install the required dependencies

```

python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai

```

For additional information, please follow [installation guide](#)

Source code

Also available on [GitHub](#)

```

1  #!/usr/bin/env python3
2
3  """
4  This example demonstrates use of queue events to block a thread until a message
5  arrives to any (of the specified) queue
6  """
7
8  import cv2
9  import depthai as dai
10
11 # Start defining a pipeline
12 pipeline = dai.Pipeline()
13
14 # Create Color and Mono cameras
15 camRgb = pipeline.createColorCamera()
16 camMono = pipeline.createMonoCamera()
17 # Create separate streams for them
18 xoutRgb = pipeline.createXLinkOut()
19 xoutMono = pipeline.createXLinkOut()
20
21 # Set properties
22 xoutRgb.setStreamName("rgb")
23 xoutMono.setStreamName("mono")
24 # Cap color camera to 5 fps
25 camRgb.setFps(5)
26 camRgb.setInterleaved(True)
27 camRgb.setPreviewSize(300, 300)
28
29 # Connect
30 camRgb.preview.link(xoutRgb.input)
31 camMono.out.link(xoutMono.input)
32
33
34 # Pipeline is defined, now we can connect to the device
35 with dai.Device(pipeline) as device:
36     # Start pipeline
37     device.startPipeline()
38
39     # Clear queue events
40     device.getQueueEvents()
41
42     while True:
43         # Block until a message arrives to any of the specified queues
44         queueName = device.getQueueEvent(("rgb", "mono"))
45
46         # Getting that message from queue with name specified by the event
47         # Note: number of events doesn't necessarily match number of messages in_
48         ↪ queues
49         # because queues can be set to non-blocking (overwriting) behavior
50         message = device.getOutputQueue(queueName).get()
51
52         # Display arrived frames
53         if type(message) == dai.ImgFrame:
54             cv2.imshow(queueName, message.getCvFrame())

```

(continues on next page)

(continued from previous page)

```

54
55     if cv2.waitKey(1) == ord('q'):
56         break

```

We're always happy to help with code or other questions you might have.

3.11.19 17 - Video & MobilenetSSD

This example shows how to MobileNetV2SSD on the RGB input frame, which is read from the specified file, and not from the RGB camera, and how to display both the RGB frame and the metadata results from the MobileNetV2SSD on the frame. DepthAI is used here only as a processing unit

Demo

Setup

Please run the following command to install the required dependencies

```

python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai

```

For additional information, please follow [installation guide](#)

This example also requires MobilenetSSD blob (mobilenet-ssd_openvino_2021.2_8shave.blob file) and prerecorded video (construction_vest.mp4 file) to work - you can download them here: [mobilenet-ssd_openvino_2021.2_8shave.blob](#) and [construction_vest.mp4](#)

Source code

Also available on [GitHub](#)

```

1  #!/usr/bin/env python3
2
3  from pathlib import Path
4  import sys
5  import cv2
6  import depthai as dai
7  import numpy as np
8  from time import monotonic
9
10 # Get argument first
11 parentDir = Path(__file__).parent
12 nnPath = str((parentDir / Path('models/mobilenet-ssd_openvino_2021.2_8shave.blob')).
13     ↪ resolve().absolute())
14 videoPath = str((parentDir / Path('models/construction_vest.mp4')).resolve()).
15     ↪ absolute())
16 if len(sys.argv) > 2:
17     nnPath = sys.argv[1]
18     videoPath = sys.argv[2]
19
20 if not Path(nnPath).exists() or not Path(videoPath).exists():
21     import sys

```

(continues on next page)

(continued from previous page)

```

20     raise FileNotFoundError(f'Required file/s not found, please run "{sys.executable} \
↳install_requirements.py"')
21
22 # Start defining a pipeline
23 pipeline = dai.Pipeline()
24
25 # Create xLink input to which host will send frames from the video file
26 xinFrame = pipeline.createXLinkIn()
27 xinFrame.setStreamName("inFrame")
28
29 # Define a neural network that will make predictions based on the source frames
30 nn = pipeline.createMobileNetDetectionNetwork()
31 nn.setConfidenceThreshold(0.5)
32 nn.setBlobPath(nnPath)
33 nn.setNumInferenceThreads(2)
34 nn.input.setBlocking(False)
35 xinFrame.out.link(nn.input)
36
37 # Create output
38 nnOut = pipeline.createXLinkOut()
39 nnOut.setStreamName("nn")
40 nn.out.link(nnOut.input)
41
42 # MobilenetSSD label texts
43 labelMap = ["background", "aeroplane", "bicycle", "bird", "boat", "bottle", "bus",
↳"car", "cat", "chair", "cow",
44             "diningtable", "dog", "horse", "motorbike", "person", "pottedplant",
↳"sheep", "sofa", "train", "tvmonitor"]
45
46
47 # Pipeline is defined, now we can connect to the device
48 with dai.Device(pipeline) as device:
49     # Start pipeline
50     device.startPipeline()
51
52     # Input queue will be used to send video frames to the device.
53     qIn = device.getInputQueue(name="inFrame")
54     # Output queue will be used to get nn data from the video frames.
55     qDet = device.getOutputQueue(name="nn", maxSize=4, blocking=False)
56
57     frame = None
58     detections = []
59
60     # nn data, being the bounding box locations, are in <0..1> range - they need to
↳be normalized with frame width/height
61     def frameNorm(frame, bbox):
62         normVals = np.full(len(bbox), frame.shape[0])
63         normVals[::2] = frame.shape[1]
64         return (np.clip(np.array(bbox), 0, 1) * normVals).astype(int)
65
66
67     def to_planar(arr: np.ndarray, shape: tuple) -> np.ndarray:
68         return cv2.resize(arr, shape).transpose(2, 0, 1).flatten()
69
70     def displayFrame(name, frame):
71         for detection in detections:
72             bbox = frameNorm(frame, (detection.xmin, detection.ymin, detection.xmax,
↳detection.ymax))

```

(continues on next page)

(continued from previous page)

```

73         cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (255, 0, 0),
↪2)
74         cv2.putText(frame, labelMap[detection.label], (bbox[0] + 10, bbox[1] +
↪20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
75         cv2.putText(frame, f"{int(detection.confidence * 100)}%", (bbox[0] + 10,
↪bbox[1] + 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
76         cv2.imshow(name, frame)
77
78     cap = cv2.VideoCapture(videoPath)
79     while cap.isOpened():
80         read_correctly, frame = cap.read()
81         if not read_correctly:
82             break
83
84         img = dai.ImgFrame()
85         img.setData(to_planar(frame, (300, 300)))
86         img.setTimestamp(monotonic())
87         img.setWidth(300)
88         img.setHeight(300)
89         qIn.send(img)
90
91         inDet = qDet.tryGet()
92
93         if inDet is not None:
94             detections = inDet.detections
95
96         if frame is not None:
97             displayFrame("rgb", frame)
98
99         if cv2.waitKey(1) == ord('q'):
100             break

```

We're always happy to help with code or other questions you might have.

3.11.20 18 - RGB Encoding with MobilenetSSD

This example shows how to configure the depthai video encoder in h.265 format to encode the RGB camera input at Full-HD resolution at 30FPS, and transfers the encoded video over XLINK to the host, saving it to disk as a video file. In the same time, a MobileNetv2SSD network is ran on the frames from the same RGB camera that is used for encoding

Pressing Ctrl+C will stop the recording and then convert it using ffmpeg into an mp4 to make it playable. Note that ffmpeg will need to be installed and runnable for the conversion to mp4 to succeed.

Be careful, this example saves encoded video to your host storage. So if you leave it running, you could fill up your storage on your host.

Demo

Setup

Please run the following command to install the required dependencies

```
python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai
```

For additional information, please follow [installation guide](#)

This example also requires MobilenetSDD blob (mobilenet-ssd_openvino_2021.2_6shave.blob file) to work - you can download it from [here](#)

Source code

Also available on GitHub

```

1  #!/usr/bin/env python3
2
3  from pathlib import Path
4  import sys
5  import cv2
6  import depthai as dai
7  import numpy as np
8
9  # Get argument first
10 nnPath = str((Path(__file__).parent / Path('models/mobilenet-ssd_openvino_2021.2_
    ↳6shave.blob')).resolve().absolute())
11 if len(sys.argv) > 1:
12     nnPath = sys.argv[1]
13
14 if not Path(nnPath).exists():
15     import sys
16     raise FileNotFoundError(f'Required file/s not found, please run "{sys.executable}
    ↳install_requirements.py"')
17
18 pipeline = dai.Pipeline()
19
20 cam = pipeline.createColorCamera()
21 cam.setBoardSocket(dai.CameraBoardSocket.RGB)
22 cam.setResolution(dai.ColorCameraProperties.SensorResolution.THE_1080_P)
23 cam.setPreviewSize(300, 300)
24 cam.setInterleaved(False)
25
26 videoEncoder = pipeline.createVideoEncoder()
27 videoEncoder.setDefaultProfilePreset(1920, 1080, 30, dai.VideoEncoderProperties.
    ↳Profile.H265_MAIN)
28 cam.video.link(videoEncoder.input)
29
30 nn = pipeline.createMobileNetDetectionNetwork()
31 nn.setConfidenceThreshold(0.5)
32 nn.setBlobPath(nnPath)
33 nn.setNumInferenceThreads(2)
34 nn.input.setBlocking(False)
35 cam.preview.link(nn.input)

```

(continues on next page)

(continued from previous page)

```

36
37 videoOut = pipeline.createXLinkOut()
38 videoOut.setStreamName('h265')
39 videoEncoder.bitstream.link(videoOut.input)
40
41 xoutRgb = pipeline.createXLinkOut()
42 xoutRgb.setStreamName("rgb")
43 cam.preview.link(xoutRgb.input)
44
45 nnOut = pipeline.createXLinkOut()
46 nnOut.setStreamName("nn")
47 nn.out.link(nnOut.input)
48
49 # MobilenetSSD label texts
50 labelMap = ["background", "aeroplane", "bicycle", "bird", "boat", "bottle", "bus",
51 ↪ "car", "cat", "chair", "cow",
52 ↪ "diningtable", "dog", "horse", "motorbike", "person", "pottedplant",
53 ↪ "sheep", "sofa", "train", "tvmonitor"]
54
55 with dai.Device(pipeline) as device, open('video.h265', 'wb') as videoFile:
56     device.startPipeline()
57
58     queue_size = 8
59     qRgb = device.getOutputQueue("rgb", queue_size)
60     qDet = device.getOutputQueue("nn", queue_size)
61     qRgbEnc = device.getOutputQueue('h265', maxSize=30, blocking=True)
62
63     frame = None
64     detections = []
65
66     def frameNorm(frame, bbox):
67         normVals = np.full(len(bbox), frame.shape[0])
68         normVals[::2] = frame.shape[1]
69         return (np.clip(np.array(bbox), 0, 1) * normVals).astype(int)
70
71     def displayFrame(name, frame):
72         for detection in detections:
73             bbox = frameNorm(frame, (detection.xmin, detection.ymin, detection.xmax, ↪
74 ↪ detection.ymax))
75             cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (255, 0, 0), ↪
76 ↪ 2)
77             cv2.putText(frame, labelMap[detection.label], (bbox[0] + 10, bbox[1] + ↪
78 ↪ 20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
79             cv2.putText(frame, f"{int(detection.confidence * 100)}%", (bbox[0] + 10, ↪
80 ↪ bbox[1] + 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
81             cv2.imshow(name, frame)
82
83     while True:
84         inRgb = qRgb.tryGet()
85         inDet = qDet.tryGet()
86
87         while qRgbEnc.has():
88             qRgbEnc.get().getData().tofile(videoFile)
89
90         if inRgb is not None:

```

(continues on next page)

(continued from previous page)

```

87         frame = inRgb.getCvFrame()
88
89         if inDet is not None:
90             detections = inDet.detections
91
92         if frame is not None:
93             displayFrame("rgb", frame)
94
95         if cv2.waitKey(1) == ord('q'):
96             break
97
98 print("To view the encoded data, convert the stream file (.h265) into a video file (.
99 ↪mp4) using a command below:")
100 print("ffmpeg -framerate 30 -i video.h265 -c copy video.mp4")

```

We're always happy to help with code or other questions you might have.

3.11.21 19 - Mono Camera Control

This example shows how to control two mono cameras, set up a pipeline that outputs grayscale camera images, connects over XLink to transfer these to the host real-time, and displays both using OpenCV.

Setup

Please run the following command to install the required dependencies

```

python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai

```

For additional information, please follow [installation guide](#)

Source code

Also available on GitHub

```

1  #!/usr/bin/env python3
2
3  """
4  Mono camera control demo:
5      Control:      key[dec/inc]  min..max
6      exposure time:  I    O      1..33000 [us]
7      sensitivity iso:  K    L      100..1600
8  Back to autoexposure: 'E'
9  """
10
11 import cv2
12 import depthai as dai
13
14 # Start defining a pipeline
15 pipeline = dai.Pipeline()
16
17 # Define a source - two mono (grayscale) cameras

```

(continues on next page)

(continued from previous page)

```

18 camLeft = pipeline.createMonoCamera()
19 camLeft.setBoardSocket(dai.CameraBoardSocket.LEFT)
20 camLeft.setResolution(dai.MonoCameraProperties.SensorResolution.THE_720_P)
21
22 camRight = pipeline.createMonoCamera()
23 camRight.setBoardSocket(dai.CameraBoardSocket.RIGHT)
24 camRight.setResolution(dai.MonoCameraProperties.SensorResolution.THE_720_P)
25
26 # Create outputs
27 xoutLeft = pipeline.createXLinkOut()
28 xoutLeft.setStreamName('left')
29 camLeft.out.link(xoutLeft.input)
30 xoutRight = pipeline.createXLinkOut()
31 xoutRight.setStreamName('right')
32 camRight.out.link(xoutRight.input)
33
34 # Create and link control input
35 control_in = pipeline.createXLinkIn()
36 control_in.setStreamName('control')
37 control_in.out.link(camLeft.inputControl)
38 control_in.out.link(camRight.inputControl)
39
40
41 def clamp(num, v0, v1): return max(v0, min(num, v1))
42
43
44 # Pipeline is defined, now we can connect to the device
45 with dai.Device(pipeline) as device:
46     # Start pipeline
47     device.startPipeline()
48
49     # Output queues will be used to get the grayscale frames from the outputs defined_
49     ↪ above
50     qLeft = device.getOutputQueue(name="left", maxSize=4, blocking=False)
51     qRight = device.getOutputQueue(name="right", maxSize=4, blocking=False)
52     # Input queue for control commands
53     controlQueue = device.getInputQueue('control')
54
55     # Manual exposure: key-press step, defaults and limits
56     EXP_STEP = 500 # us
57     ISO_STEP = 50
58
59     exp_time = 20000
60     expMin = 1
61     expMax = 33000
62
63     sensIso = 800
64     sensMin = 100
65     sensMax = 1600
66
67     frameLeft = None
68     frameRight = None
69
70     while True:
71         # Instead of get (blocking), we use tryGet (nonblocking) which will return_
71         ↪ the available data or None otherwise
72         inLeft = qLeft.tryGet()

```

(continues on next page)

(continued from previous page)

```

73     inRight = qRight.tryGet()
74
75     if inLeft is not None:
76         frameLeft = inLeft.getCvFrame()
77
78     if inRight is not None:
79         frameRight = inRight.getCvFrame()
80
81     # show the frames if available
82     if frameLeft is not None:
83         cv2.imshow("left", frameLeft)
84     if frameRight is not None:
85         cv2.imshow("right", frameRight)
86
87     key = cv2.waitKey(1)
88     if key == ord('q'):
89         break
90     elif key in [ord('i'), ord('o'), ord('k'), ord('l')]:
91         if key == ord('i'):
92             exp_time -= EXP_STEP
93         if key == ord('o'):
94             exp_time += EXP_STEP
95         if key == ord('k'):
96             sensIso -= ISO_STEP
97         if key == ord('l'):
98             sensIso += ISO_STEP
99         exp_time = clamp(exp_time, expMin, expMax)
100        sensIso = clamp(sensIso, sensMin, sensMax)
101        print("Setting manual exposure, time:", exp_time, "iso:", sensIso)
102        ctrl = dai.CameraControl()
103        ctrl.setManualExposure(exp_time, sensIso)
104        controlQueue.send(ctrl)
105    elif key == ord('e'):
106        print("Autoexposure enable")
107        ctrl = dai.CameraControl()
108        ctrl.setAutoExposureEnable()
109        controlQueue.send(ctrl)

```

We're always happy to help with code or other questions you might have.

3.11.22 20 - Color Rotate Warp

This example shows usage of ImageManip to crop a rotated rectangle area on a frame, or perform various image transforms: rotate, mirror, flip, perspective transform.

Setup

Please run the following command to install the required dependencies

```
python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai
```

For additional information, please follow *installation guide*

Source code

Also available on [GitHub](#)

```
1  #!/usr/bin/env python3
2
3  """
4  This example shows usage of ImageManip to crop a rotated rectangle area on a frame,
5  or perform various image transforms: rotate, mirror, flip, perspective transform.
6  """
7
8  import depthai as dai
9  import cv2
10 import numpy as np
11
12 keyRotateDecr = 'z'
13 keyRotateIncr = 'x'
14 keyResizeInc = 'v'
15 keyWarpTestCycle = 'c'
16
17
18 def printControls():
19     print("=== Controls:")
20     print(keyRotateDecr, "-rotated rectangle crop, decrease rate")
21     print(keyRotateIncr, "-rotated rectangle crop, increase rate")
22     print(keyWarpTestCycle, "-warp 4-point transform, cycle through modes")
23     print(keyResizeInc, "-resize cropped region, or disable resize")
24     print("h -print controls (help)")
25
26
27 rotateRateMax = 5.0
28 rotateRateInc = 0.1
29
30 resizeModeW = 800
31 resizeModeH = 600
32 resizeFactorMax = 5
33
34 '''
35 The crop points are specified in clockwise order,
36 with first point mapped to output top-left, as:
37     P0  ->  P1
38     ^      v
39     P3  <-  P2
40 '''
41 P0 = [0, 0] # top-left
42 P1 = [1, 0] # top-right
43 P2 = [1, 1] # bottom-right
```

(continues on next page)

(continued from previous page)

```

44 P3 = [0, 1] # bottom-left
45 warpTestList = [
46     # points order, normalized coordinates, description
47     # [[[0, 0], [1, 0], [1, 1], [0, 1]], True, "passthrough"],
48     # [[[0, 0], [639, 0], [639, 479], [0, 479]], False, "passthrough (pixels)"],
49     [[P0, P1, P2, P3], True, "1.passthrough"],
50     [[P3, P0, P1, P2], True, "2.rotate 90"],
51     [[P2, P3, P0, P1], True, "3.rotate 180"],
52     [[P1, P2, P3, P0], True, "4.rotate 270"],
53     [[P1, P0, P3, P2], True, "5.horizontal mirror"],
54     [[P3, P2, P1, P0], True, "6.vertical flip"],
55     [[[-0.1, -0.1], [1.1, -0.1], [1.1, 1.1], [-0.1, 1.1]], True, "7.add black borders
↪"],
56     [[[-0.3, 0], [1, 0], [1.3, 1], [0, 1]], True, "8.parallelogram transform"],
57     [[[-0.2, 0], [1.8, 0], [1, 1], [0, 1]], True, "9.trapezoid transform"],
58 ]
59
60 pipeline = dai.Pipeline()
61
62 cam = pipeline.createColorCamera()
63 cam.setPreviewSize(640, 480)
64 cam.setInterleaved(False)
65 camOut = pipeline.createXLinkOut()
66 camOut.setStreamName("preview")
67
68 manip = pipeline.createImageManip()
69 manip.setMaxOutputFrameSize(2000*1500*3)
70 manipOut = pipeline.createXLinkOut()
71 manipOut.setStreamName("manip")
72 manipCfg = pipeline.createXLinkIn()
73 manipCfg.setStreamName("manipCfg")
74
75 cam.preview.link(camOut.input)
76 cam.preview.link(manip.inputImage)
77 manip.out.link(manipOut.input)
78 manipCfg.out.link(manip.inputConfig)
79
80 with dai.Device(pipeline) as device:
81     device.startPipeline()
82
83     qPreview = device.getOutputQueue(name="preview", maxSize=4)
84     qManip = device.getOutputQueue(name="manip", maxSize=4)
85     qManipCfg = device.getInputQueue(name="manipCfg")
86
87     key = -1
88     angleDeg = 0
89     rotateRate = 1.0
90     resizeFactor = 0
91     resizeX = 0
92     resizeY = 0
93     testFourPt = False
94     warpIdx = -1
95
96     printControls()
97
98     while key != ord('q'):
99         if key > 0:

```

(continues on next page)

(continued from previous page)

```

100     if key == ord(keyRotateDecr) or key == ord(keyRotateIncr):
101         if key == ord(keyRotateDecr):
102             if rotateRate > -rotateRateMax:
103                 rotateRate -= rotateRateInc
104         if key == ord(keyRotateIncr):
105             if rotateRate < rotateRateMax:
106                 rotateRate += rotateRateInc
107         testFourPt = False
108         print("Crop rotated rectangle, rate per frame: {:.1f} degrees.".
↪format(rotateRate))
109     elif key == ord(keyResizeInc):
110         resizeFactor += 1
111         if resizeFactor > resizeFactorMax:
112             resizeFactor = 0
113             print("Crop region not resized")
114         else:
115             resizeX = resizeMaxW // resizeFactor
116             resizeY = resizeMaxH // resizeFactor
117             print("Crop region resized to", resizeX, 'x', resizeY)
118     elif key == ord(keyWarpTestCycle):
119         # Disable resizing initially
120         resizeFactor = 0
121         warpIdx = (warpIdx + 1) % len(warpTestList)
122         testFourPt = True
123         testDescription = warpTestList[warpIdx][2]
124         print("Warp 4-point transform:", testDescription)
125     elif key == ord('h'):
126         printControls()
127
128     # Send an updated config with continuous rotate, or after a key press
129     if key >= 0 or (not testFourPt and abs(rotateRate) > 0.0001):
130         cfg = dai.ImageManipConfig()
131         if testFourPt:
132             test = warpTestList[warpIdx]
133             points, normalized = test[0], test[1]
134             # TODO: improve this, should avoid this conversion
135             point2fList = []
136             for p in points:
137                 pt = dai.Point2f()
138                 pt.x, pt.y = p[0], p[1]
139                 point2fList.append(pt)
140             cfg.setWarpTransformFourPoints(point2fList, normalized)
141         else:
142             angleDeg += rotateRate
143             rotatedRect = ((320, 240), (400, 400), angleDeg)
144             rr = dai.RotatedRect()
145             rr.center.x, rr.center.y = rotatedRect[0]
146             rr.size.width, rr.size.height = rotatedRect[1]
147             rr.angle = rotatedRect[2]
148             cfg.setCropRotatedRect(rr, False)
149         if resizeFactor > 0:
150             cfg.setResize(resizeX, resizeY)
151         # cfg.setWarpBorderFillColor(0, 0, 255)
152         # cfg.setWarpBorderReplicatePixels()
153         qManipCfg.send(cfg)
154
155     for q in [qPreview, qManip]:

```

(continues on next page)

(continued from previous page)

```

156     pkt = q.get()
157     name = q.getName()
158     shape = (3, pkt.getHeight(), pkt.getWidth())
159     frame = pkt.getCvFrame()
160     if name == "preview" and not testFourPt:
161         # Draw RotatedRect cropped area on input frame
162         points = np.int0(cv2.boxPoints(rotatedRect))
163         cv2.drawContours(frame, [points], 0, (255, 0, 0), 1)
164         # Mark top-left corner
165         cv2.circle(frame, tuple(points[1]), 10, (255, 0, 0), 2)
166     cv2.imshow(name, frame)
167     key = cv2.waitKey(1)

```

We're always happy to help with code or other questions you might have.

3.11.23 22.1 - RGB & TinyYoloV3 decoding on device

This example shows how to run TinyYoloV3 on the RGB input frame, and how to display both the RGB preview and the metadata results from the TinyYoloV3 on the preview. Decoding is done on Myriad instead on the host.

Configurable, network dependent parameters are required for correct decoding: setNumClasses - number of YOLO classes setCoordinateSize - size of coordinate setAnchors - yolo anchors setAnchorMasks - anchorMasks26, anchorMasks13 (anchorMasks52 - additionally for full YOLOv3) setIouThreshold - intersection over union threshold setConfidenceThreshold - confidence threshold above which objects are detected

Demo

Setup

Please run the following command to install the required dependencies

```

python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai

```

For additional information, please follow [installation guide](#)

This example also requires YoloV3-tiny blob (tiny-yolo-v3_openvino_2021.2_6shave.blob file) to work - you can download it from [here](#)

Source code

Also available on GitHub

```

1  #!/usr/bin/env python3
2
3  """
4  Tiny-yolo-v3 device side decoding demo
5  YOLO v3 Tiny is a real-time object detection model implemented with Keras* from
6  this repository <https://github.com/david8862/keras-YOLOv3-model-set> and converted
7  to TensorFlow* framework. This model was pretrained on COCO* dataset with 80
8  ↪ classes.
9  """

```

(continues on next page)

(continued from previous page)

```

9
10 from pathlib import Path
11 import sys
12 import cv2
13 import depthai as dai
14 import numpy as np
15 import time
16
17 # Tiny yolo v3 label texts
18 labelMap = [
19     "person",      "bicycle",    "car",            "motorbike",     "aeroplane",
20     ↪ "bus",         "train",          "traffic light",  "fire hydrant",  "stop sign",
21     "truck",       "boat",           "parking meter", "bench",         "sheep",
22     ↪ "bird",       "cat",            "dog",           "horse",         "umbrella",
23     ↪ "cow",        "elephant",       "giraffe",       "backpack",      "sports ball",
24     "bear",        "zebra",          "skis",          "snowboard",    "tennis racket",
25     ↪ "handbag",    "tie",            "baseball bat",  "bottle",
26     "suitcase",    "frisbee",        "surfboard",    "baseball glove",
27     ↪ "kite",       "cup",            "skateboard",   "bottle",
28     "wine glass",  "knife",          "spoon",        "tennis racket",
29     ↪ "fork",       "sandwich",       "bowl",         "bottle",
30     "apple",       "broccoli",       "hot dog",      "banana",
31     ↪ "orange",    "carrot",         "pizza",
32     "donut",       "cake",           "pottedplant",  "diningtable",
33     ↪ "chair",     "sofa",           "bed",          "diningtable",
34     "toilet",      "tvmonitor",      "remote",       "cell phone",
35     ↪ "laptop",    "mouse",          "keyboard",     "cell phone",
36     "microwave",   "oven",           "refrigerator", "clock",
37     ↪ "toaster",   "sink",           "book",         "clock",
38     "vase",        "scissors",       "teddy bear",   "toothbrush",
39     ↪ "teddy bear", "hair drier",    "toothbrush"
40 ]
41
42 syncNN = True
43
44 # Get argument first
45 nnPath = str((Path(__file__).parent / Path('models/tiny-yolo-v3_opencv_2021.2_
46 ↪ 6shave.blob')).resolve().absolute())
47
48 if len(sys.argv) > 1:
49     nnPath = sys.argv[1]
50
51 if not Path(nnPath).exists():
52     import sys
53     raise FileNotFoundError(f'Required file/s not found, please run "{sys.executable}
54 ↪ install_requirements.py"')
55
56 # Start defining a pipeline
57 pipeline = dai.Pipeline()
58
59 # Define a source - color camera
60 camRgb = pipeline.createColorCamera()
61 camRgb.setPreviewSize(416, 416)
62 camRgb.setInterleaved(False)
63 camRgb.setFps(40)

```

(continues on next page)

(continued from previous page)

```

53
54 # network specific settings
55 detectionNetwork = pipeline.createYoloDetectionNetwork()
56 detectionNetwork.setConfidenceThreshold(0.5)
57 detectionNetwork.setNumClasses(80)
58 detectionNetwork.setCoordinateSize(4)
59 detectionNetwork.setAnchors(np.array([10,14, 23,27, 37,58, 81,82, 135,169, 344,319]))
60 detectionNetwork.setAnchorMasks({"side26": np.array([1, 2, 3]), "side13": np.array([3,
    ↳ 4, 5])})
61 detectionNetwork.setIouThreshold(0.5)
62
63 detectionNetwork.setBlobPath(nnPath)
64 detectionNetwork.setNumInferenceThreads(2)
65 detectionNetwork.input.setBlocking(False)
66
67 camRgb.preview.link(detectionNetwork.input)
68
69 # Create outputs
70 xoutRgb = pipeline.createXLinkOut()
71 xoutRgb.setStreamName("rgb")
72 if syncNN:
73     detectionNetwork.passthrough.link(xoutRgb.input)
74 else:
75     camRgb.preview.link(xoutRgb.input)
76
77 nnOut = pipeline.createXLinkOut()
78 nnOut.setStreamName("detections")
79 detectionNetwork.out.link(nnOut.input)
80
81
82 # Pipeline is defined, now we can connect to the device
83 with dai.Device(pipeline) as device:
84     # Start pipeline
85     device.startPipeline()
86
87     # Output queues will be used to get the rgb frames and nn data from the outputs_
    ↳ defined above
88     qRgb = device.getOutputQueue(name="rgb", maxSize=4, blocking=False)
89     qDet = device.getOutputQueue(name="detections", maxSize=4, blocking=False)
90
91     frame = None
92     detections = []
93
94     # nn data, being the bounding box locations, are in <0..1> range - they need to_
    ↳ be normalized with frame width/height
95     def frameNorm(frame, bbox):
96         normVals = np.full(len(bbox), frame.shape[0])
97         normVals[::2] = frame.shape[1]
98         return (np.clip(np.array(bbox), 0, 1) * normVals).astype(int)
99
100     def displayFrame(name, frame):
101         for detection in detections:
102             bbox = frameNorm(frame, (detection.xmin, detection.ymin, detection.xmax,
    ↳ detection.ymax))
103             cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (255, 0, 0),
    ↳ 2)
104             cv2.putText(frame, labelMap[detection.label], (bbox[0] + 10, bbox[1] +
    ↳ 20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)

```

(continues on next page)

(continued from previous page)

```

105         cv2.putText(frame, f"{int(detection.confidence * 100)}%", (bbox[0] + 10, ↵
↵bbox[1] + 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
106         cv2.imshow(name, frame)
107
108     startTime = time.monotonic()
109     counter = 0
110
111     while True:
112         if syncNN:
113             inRgb = qRgb.get()
114             inDet = qDet.get()
115         else:
116             inRgb = qRgb.tryGet()
117             inDet = qDet.tryGet()
118
119         if inRgb is not None:
120             frame = inRgb.getCvFrame()
121             cv2.putText(frame, "NN fps: {:.2f}".format(counter / (time.monotonic() - ↵
↵startTime)),
122                 (2, frame.shape[0] - 4), cv2.FONT_HERSHEY_TRIPLEX, 0.4, ↵
↵color=(255, 255, 255))
123
124         if inDet is not None:
125             detections = inDet.detections
126             counter += 1
127
128         if frame is not None:
129             displayFrame("rgb", frame)
130
131         if cv2.waitKey(1) == ord('q'):
132             break

```

We're always happy to help with code or other questions you might have.

3.11.24 22.2 - RGB & TinyYoloV4 decoding on device

This example shows how to run TinyYoloV4 on the RGB input frame, and how to display both the RGB preview and the metadata results from the TinyYoloV4 on the preview. Decoding is done on Myriad instead on the host.

Configurable, network dependent parameters are required for correct decoding: setNumClasses - number of YOLO classes setCoordinateSize - size of coordinate setAnchors - yolo anchors setAnchorMasks - anchorMasks26, anchorMasks13 (anchorMasks52 - additionally for full YOLOv4) setIouThreshold - intersection over union threshold setConfidenceThreshold - confidence threshold above which objects are detected

Demo

Setup

Please run the following command to install the required dependencies

```

python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai

```

For additional information, please follow *installation guide*

This example also requires YOLOv4-tiny blob (tiny-yolo-v4_openvino_2021.2_6shave.blob file) to work - you can download it from [here](#)

Source code

Also available on GitHub

```

1  #!/usr/bin/env python3
2
3  """
4  Tiny-yolo-v4 device side decoding demo
5  The code is the same as for Tiny-yolo-V3, the only difference is the blob file.
6  The blob was compiled following this tutorial: https://github.com/TNTWEN/OpenVINO-
   ↳ YOLOV4
7  """
8
9  from pathlib import Path
10 import sys
11 import cv2
12 import depthai as dai
13 import numpy as np
14 import time
15
16 # tiny yolo v4 label texts
17 labelMap = [
18     "person",      "bicycle",    "car",            "motorbike",     "aeroplane",
19     ↳ "bus",        "train",          "traffic light",  "fire hydrant",  "stop sign",
20     "truck",       "boat",           "parking meter", "bench",         "dog",
21     ↳ "bird",      "cat",            "horse",         "sheep",
22     "cow",         "elephant",       "giraffe",       "backpack",      "umbrella",
23     ↳ "bear",      "zebra",          "skis",          "snowboard",    "sports ball",
24     "suitcase",    "frisbee",        "baseball bat",  "tennis racket", "bottle",
25     ↳ "kite",      "skateboard",     "surfboard",    "cup",
26     "baseball glove", "wine glass",    "fork",          "knife",         "spoon",
27     ↳ "apple",     "sandwich",      "bowl",          "banana",
28     "orange",      "broccoli",      "carrot",        "hot dog",       "pizza",
29     ↳ "donut",     "cake",          "pottedplant",  "bed",           "diningtable",
30     "chair",       "sofa",          "tvmonitor",    "remote",        "keyboard",
31     ↳ "toilet",    "laptop",        "mouse",        "cell phone",
32     "microwave",   "oven",          "refrigerator", "book",          "clock",
33     ↳ "toaster",  "sink",          "scissors",     "teddy bear",   "hair drier",  "toothbrush"
34 ]
35
36 syncNN = True
37
38 # Get argument first
39 nnPath = str((Path(__file__).parent / Path('models/tiny-yolo-v4_openvino_2021.2_
   ↳ 6shave.blob')).resolve().absolute())

```

(continues on next page)

(continued from previous page)

```

37 if len(sys.argv) > 1:
38     nnPath = sys.argv[1]
39
40 if not Path(nnPath).exists():
41     import sys
42     raise FileNotFoundError(f'Required file/s not found, please run "{sys.executable} \
↳install_requirements.py"')
43
44 # Start defining a pipeline
45 pipeline = dai.Pipeline()
46
47 # Define a source - color camera
48 camRgb = pipeline.createColorCamera()
49 camRgb.setPreviewSize(416, 416)
50 camRgb.setInterleaved(False)
51 camRgb.setFps(40)
52
53 # Network specific settings
54 detectionNetwork = pipeline.createYoloDetectionNetwork()
55 detectionNetwork.setConfidenceThreshold(0.5)
56 detectionNetwork.setNumClasses(80)
57 detectionNetwork.setCoordinateSize(4)
58 detectionNetwork.setAnchors(np.array([10,14, 23,27, 37,58, 81,82, 135,169, 344,319]))
59 detectionNetwork.setAnchorMasks({"side26": np.array([1, 2, 3]), "side13": np.array([3,
↳ 4, 5])})
60 detectionNetwork.setIouThreshold(0.5)
61
62 detectionNetwork.setBlobPath(nnPath)
63 detectionNetwork.setNumInferenceThreads(2)
64 detectionNetwork.input.setBlocking(False)
65
66 camRgb.preview.link(detectionNetwork.input)
67
68 # Create outputs
69 xoutRgb = pipeline.createXLinkOut()
70 xoutRgb.setStreamName("rgb")
71 if syncNN:
72     detectionNetwork.passthrough.link(xoutRgb.input)
73 else:
74     camRgb.preview.link(xoutRgb.input)
75
76 nnOut = pipeline.createXLinkOut()
77 nnOut.setStreamName("detections")
78 detectionNetwork.out.link(nnOut.input)
79
80
81 # Pipeline is defined, now we can connect to the device
82 with dai.Device(pipeline) as device:
83     # Start pipeline
84     device.startPipeline()
85
86     # Output queues will be used to get the rgb frames and nn data from the outputs_
↳defined above
87     qRgb = device.getOutputQueue(name="rgb", maxSize=4, blocking=False)
88     qDet = device.getOutputQueue(name="detections", maxSize=4, blocking=False)
89
89     frame = None

```

(continues on next page)

(continued from previous page)

```

91     detections = []
92
93     # nn data, being the bounding box locations, are in <0..1> range - they need to
94     ↪be normalized with frame width/height
95     def frameNorm(frame, bbox):
96         normVals = np.full(len(bbox), frame.shape[0])
97         normVals[::2] = frame.shape[1]
98         return (np.clip(np.array(bbox), 0, 1) * normVals).astype(int)
99
100     def displayFrame(name, frame):
101         for detection in detections:
102             bbox = frameNorm(frame, (detection.xmin, detection.ymin, detection.xmax,
103             ↪detection.ymax))
104             cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (255, 0, 0),
105             ↪2)
106             cv2.putText(frame, labelMap[detection.label], (bbox[0] + 10, bbox[1] +
107             ↪20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
108             cv2.putText(frame, f"{int(detection.confidence * 100)}%", (bbox[0] + 10,
109             ↪bbox[1] + 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
110             cv2.imshow(name, frame)
111
112     startTime = time.monotonic()
113     counter = 0
114
115     while True:
116         if syncNN:
117             inRgb = qRgb.get()
118             inDet = qDet.get()
119         else:
120             inRgb = qRgb.tryGet()
121             inDet = qDet.tryGet()
122
123         if inRgb is not None:
124             frame = inRgb.getCvFrame()
125             cv2.putText(frame, "NN fps: {:.2f}".format(counter / (time.monotonic() -
126             ↪startTime)),
127             ↪(2, frame.shape[0] - 4), cv2.FONT_HERSHEY_TRIPLEX, 0.4,
128             ↪color=(255, 255, 255))
129
130         if inDet is not None:
131             detections = inDet.detections
132             counter += 1
133
134         if frame is not None:
135             displayFrame("rgb", frame)
136
137         if cv2.waitKey(1) == ord('q'):
138             break

```

We're always happy to help with code or other questions you might have.

3.11.25 23 - Auto Exposure on ROI

This example shows how to dynamically set the Auto Exposure (AE) of the RGB camera dynamically, during application runtime, based on bounding box position

Demo

Setup

Please run the following command to install the required dependencies

```
python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai
```

For additional information, please follow [installation guide](#)

This example also requires MobilenetSDD blob (mobilenet-ssd_openvino_2021.2_5shave.blob file) to work - you can download it from [here](#)

Usage

By default, AutoExposure region is adjusted based on neural network output. If desired, the region can be set manually. You can do so by pressing one of the following buttons:

- *w* - move AE region up
- *s* - move AE region down
- *a* - move AE region left
- *d* - move AE region right
- *n* - deactivate manual region (switch back to nn-based roi)

Source code

Also available on [GitHub](#)

```
1  #!/usr/bin/env python3
2
3  from pathlib import Path
4  import sys
5  import cv2
6  import depthai as dai
7  import numpy as np
8
9  # Press WASD to move a manual ROI window for auto-exposure control.
10 # Press N to go back to the region controlled by the NN detections.
11
12 # Get argument first
13 nnPath = str((Path(__file__).parent / Path('models/mobilenet-ssd_openvino_2021.2_
14 ↪6shave.blob')).resolve().absolute())
15 if len(sys.argv) > 1:
16     nnPath = sys.argv[1]
```

(continues on next page)

(continued from previous page)

```

17 if not Path(nnPath).exists():
18     import sys
19     raise FileNotFoundError(f'Required file/s not found, please run "{sys.executable} ↵
↵install_requirements.py"')
20
21 previewSize = (300, 300)
22
23 # Start defining a pipeline
24 pipeline = dai.Pipeline()
25
26 # Define a source - color camera
27 camRgb = pipeline.createColorCamera()
28 camRgb.setPreviewSize(*previewSize)
29 camRgb.setInterleaved(False)
30
31 camControlIn = pipeline.createXLinkIn()
32 camControlIn.setStreamName('camControl')
33 camControlIn.out.link(camRgb.inputControl)
34
35 # Define a neural network that will make predictions based on the source frames
36 nn = pipeline.createMobileNetDetectionNetwork()
37 nn.setConfidenceThreshold(0.5)
38 nn.setBlobPath(nnPath)
39 nn.setNumInferenceThreads(2)
40 nn.input.setBlocking(False)
41 camRgb.preview.link(nn.input)
42
43 # Create outputs
44 xoutRgb = pipeline.createXLinkOut()
45 xoutRgb.setStreamName("rgb")
46 camRgb.preview.link(xoutRgb.input)
47
48 nnOut = pipeline.createXLinkOut()
49 nnOut.setStreamName("nn")
50 nn.out.link(nnOut.input)
51
52 # MobilenetSSD label texts
53 labelMap = ["background", "aeroplane", "bicycle", "bird", "boat", "bottle", "bus",
↵"car", "cat", "chair", "cow",
54             "diningtable", "dog", "horse", "motorbike", "person", "pottedplant",
↵"sheep", "sofa", "train", "tvmonitor"]
55
56
57 def clamp(num, v0, v1):
58     return max(v0, min(num, v1))
59
60
61 def asControl(roi):
62     camControl = dai.CameraControl()
63     camControl.setAutoExposureRegion(*roi)
64     return camControl
65
66
67 class AutoExposureRegion:
68     step = 10
69     position = (0, 0)
70     size = (100, 100)

```

(continues on next page)

(continued from previous page)

```

71 resolution = camRgb.getResolutionSize()
72 maxDims = previewSize[0], previewSize[1]
73
74 def grow(self, x=0, y=0):
75     self.size = (
76         clamp(x + self.size[0], 1, self.maxDims[0]),
77         clamp(y + self.size[1], 1, self.maxDims[1])
78     )
79
80 def move(self, x=0, y=0):
81     self.position = (
82         clamp(x + self.position[0], 0, self.maxDims[0]),
83         clamp(y + self.position[1], 0, self.maxDims[1])
84     )
85
86 def endPosition(self):
87     return (
88         clamp(self.position[0] + self.size[0], 0, self.maxDims[0]),
89         clamp(self.position[1] + self.size[1], 0, self.maxDims[1]),
90     )
91
92 def toRoi(self):
93     roi = np.array([*self.position, *self.size])
94     # Convert to absolute camera coordinates
95     roi = roi * self.resolution[1] // 300
96     roi[0] += (self.resolution[0] - self.resolution[1]) // 2 # x offset for
97 device crop
98     return roi
99
100 @staticmethod
101 def bboxToRoi(bbox):
102     startX, startY = bbox[:2]
103     width, height = bbox[2] - startX, bbox[3] - startY
104     roi = frameNorm(np.empty(camRgb.getResolutionSize()), (startX, startY, width,
105 height))
106     return roi
107
108 # Pipeline is defined, now we can connect to the device
109 with dai.Device(pipeline) as device:
110     # Start pipeline
111     device.startPipeline()
112
113     # Output queues will be used to get the rgb frames and nn data from the outputs
114 defined above
115     qControl = device.getInputQueue(name="camControl")
116     qRgb = device.getOutputQueue(name="rgb", maxSize=4, blocking=False)
117     qDet = device.getOutputQueue(name="nn", maxSize=4, blocking=False)
118     frame = None
119     detections = []
120
121     nnRegion = True
122     region = AutoExposureRegion()
123
124     # nn data (bounding box locations) are in <0..1> range - they need to be
125 normalized with frame width/height
126     def frameNorm(frame, bbox):

```

(continues on next page)

(continued from previous page)

```

124     normVals = np.full(len(bbox), frame.shape[0])
125     normVals[::2] = frame.shape[1]
126     return (np.clip(np.array(bbox), 0, 1) * normVals).astype(int)
127
128     def displayFrame(name, frame):
129         for detection in detections:
130             bbox = frameNorm(frame, (detection.xmin, detection.ymin, detection.xmax,
131 ↪ detection.ymax))
132             cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (255, 0, 0),
133 ↪ 2)
134             cv2.putText(frame, labelMap[detection.label], (bbox[0] + 10, bbox[1] +
135 ↪ 20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
136             cv2.putText(frame, f"{int(detection.confidence * 100)}%", (bbox[0] + 10,
137 ↪ bbox[1] + 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
138             if not nnRegion:
139                 cv2.rectangle(frame, region.position, region.endPosition(), (0, 255, 0),
140 ↪ 2)
141             cv2.imshow(name, frame)
142
143     while True:
144         # Instead of get (blocking), we use tryGet (nonblocking) which will return
145 ↪ the available data or None otherwise
146         inRgb = qRgb.tryGet()
147         inDet = qDet.tryGet()
148
149         if inRgb is not None:
150             frame = inRgb.getCvFrame()
151
152         if inDet is not None:
153             detections = inDet.detections
154
155             if nnRegion and len(detections) > 0:
156                 bbox = (detections[0].xmin, detections[0].ymin, detections[0].xmax,
157 ↪ detections[0].ymax)
158                 qControl.send(asControl(AutoExposureRegion.bboxToRoi(bbox)))
159
160         if frame is not None:
161             displayFrame("rgb", frame)
162
163         key = cv2.waitKey(1)
164         if key == ord('n'):
165             print("AE ROI controlled by NN")
166             nnRegion = True
167         elif key in [ord('w'), ord('a'), ord('s'), ord('d'), ord('+'), ord('-')]:
168             nnRegion = False
169             if key == ord('a'):
170                 region.move(x=-region.step)
171             if key == ord('d'):
172                 region.move(x=region.step)
173             if key == ord('w'):
174                 region.move(y=-region.step)
175             if key == ord('s'):
176                 region.move(y=region.step)
177             if key == ord('+'):
178                 region.grow(x=10, y=10)
179                 region.step = region.step + 1
180             if key == ord('-'):

```

(continues on next page)

(continued from previous page)

```

174         region.grow(x=-10, y=-10)
175         region.step = max(region.step - 1, 1)
176         print(f"Setting static AE ROI: {region.toRoi()} (on frame: {[*region.
↪ position, *region.endPosition()]})")
177         qControl.send(asControl(region.toRoi()))
178         elif key == ord('q'):
179             break

```

We're always happy to help with code or other questions you might have.

3.11.26 24 - OpenCV support

This example shows API which exposes both numpy and OpenCV compatible image types for easier usage. It uses ColorCamera node to retrieve both BGR interleaved 'preview' and NV12 encoded 'video' frames. Both are displayed using functions *getFrame* and *getCvFrame*.

Demo

Setup

Please run the following command to install the required dependencies

```

python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai

```

For additional information, please follow [installation guide](#)

Source code

Also available on [GitHub](#)

```

1  #!/usr/bin/env python3
2
3  import cv2
4  import depthai as dai
5
6  # Start defining a pipeline
7  pipeline = dai.Pipeline()
8
9  # Define a source - color camera
10 camRgb = pipeline.createColorCamera()
11 camRgb.setPreviewSize(300, 300)
12 camRgb.setBoardSocket(dai.CameraBoardSocket.RGB)
13 camRgb.setResolution(dai.ColorCameraProperties.SensorResolution.THE_1080_P)
14 camRgb.setInterleaved(True)
15 camRgb.setColorOrder(dai.ColorCameraProperties.ColorOrder.BGR)
16
17 # Create output
18 xoutVideo = pipeline.createXLinkOut()
19 xoutVideo.setStreamName("video")
20 xoutPreview = pipeline.createXLinkOut()
21 xoutPreview.setStreamName("preview")

```

(continues on next page)

(continued from previous page)

```

22
23 camRgb.preview.link(xoutPreview.input)
24 camRgb.video.link(xoutVideo.input)
25
26 # Pipeline is defined, now we can connect to the device
27 with dai.Device(pipeline) as device:
28     # Start pipeline
29     device.startPipeline()
30
31     while True:
32         # Get preview and video frames
33         preview = device.getOutputQueue('preview').get()
34         video = device.getOutputQueue('video').get()
35
36         # Show 'preview' frame as is (already in correct format, no copy is made)
37         cv2.imshow("preview", preview.getFrame())
38         # Get BGR frame from NV12 encoded video frame to show with opencv
39         cv2.imshow("video", video.getCvFrame())
40
41         if cv2.waitKey(1) == ord('q'):
42             break

```

We're always happy to help with code or other questions you might have.

3.11.27 25 - System information

This example shows how to get system information (memory usage, cpu usage and temperature) from the board.

Demo

Example script output

```

Drr used / total - 0.13 / 414.80 MiB
Cmx used / total - 2.24 / 2.50 MiB
LeonCss heap used / total - 4.17 / 46.41 MiB
LeonMss heap used / total - 2.87 / 27.58 MiB
Chip temperature - average: 38.59, css: 39.81, mss: 37.71, upa0: 38.65, upa1: 38.18
Cpu usage - Leon OS: 7.08%, Leon RT: 1.48 %
-----
Drr used / total - 0.13 / 414.80 MiB
Cmx used / total - 2.24 / 2.50 MiB
LeonCss heap used / total - 4.17 / 46.41 MiB
LeonMss heap used / total - 2.87 / 27.58 MiB
Chip temperature - average: 38.59, css: 39.58, mss: 37.94, upa0: 38.18, upa1: 38.65
Cpu usage - Leon OS: 1.55%, Leon RT: 0.30 %
-----
Drr used / total - 0.13 / 414.80 MiB
Cmx used / total - 2.24 / 2.50 MiB
LeonCss heap used / total - 4.17 / 46.41 MiB
LeonMss heap used / total - 2.87 / 27.58 MiB
Chip temperature - average: 38.94, css: 40.04, mss: 38.18, upa0: 39.35, upa1: 38.18
Cpu usage - Leon OS: 0.56%, Leon RT: 0.06 %
-----
Drr used / total - 0.13 / 414.80 MiB

```

(continues on next page)

(continued from previous page)

```

Cmx used / total - 2.24 / 2.50 MiB
LeonCss heap used / total - 4.17 / 46.41 MiB
LeonMss heap used / total - 2.87 / 27.58 MiB
Chip temperature - average: 39.46, css: 40.28, mss: 38.88, upa0: 39.81, upa1: 38.88
Cpu usage - Leon OS: 0.51%, Leon RT: 0.06 %
-----

```

Setup

Please run the following command to install the required dependencies

```

python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai

```

For additional information, please follow [installation guide](#)

Source code

Also available on [GitHub](#)

```

1  #!/usr/bin/env python3
2
3  import cv2
4  import depthai as dai
5
6
7  def print_sys_info(info):
8      m = 1024 * 1024 # MiB
9      print(f"Drr used / total - {info._DDRMemoryUsage.used / m:.2f} / {info.
10     ↳DDRMemoryUsage.total / m:.2f} MiB")
11     print(f"Cmx used / total - {info.cmxMemoryUsage.used / m:.2f} / {info.
12     ↳cmxMemoryUsage.total / m:.2f} MiB")
13     print(f"LeonCss heap used / total - {info.leonCssMemoryUsage.used / m:.2f} /
14     ↳{info.leonCssMemoryUsage.total / m:.2f} MiB")
15     print(f"LeonMss heap used / total - {info.leonMssMemoryUsage.used / m:.2f} /
16     ↳{info.leonMssMemoryUsage.total / m:.2f} MiB")
17     t = info.chipTemperature
18     print(f"Chip temperature - average: {t.average:.2f}, css: {t.css:.2f}, mss: {t.
19     ↳mss:.2f}, upa0: {t.upa:.2f}, upa1: {t.dss:.2f}")
20     print(f"Cpu usage - Leon OS: {info.leonCssCpuUsage.average * 100:.2f}%, Leon RT:
21     ↳{info.leonMssCpuUsage.average * 100:.2f} %")
22     print("-----")
23
24
25  # Start defining a pipeline
26  pipeline = dai.Pipeline()
27
28  sys_logger = pipeline.createSystemLogger()
29  sys_logger.setRate(1) # 1 Hz
30
31  # Create output
32  linkOut = pipeline.createXLinkOut()
33  linkOut.setStreamName("sysinfo")

```

(continues on next page)

(continued from previous page)

```

28 sys_logger.out.link(linkOut.input)
29
30 # Pipeline is defined, now we can connect to the device
31 with dai.Device(pipeline) as device:
32     # Start pipeline
33     device.startPipeline()
34
35     # Output queue will be used to get the system info
36     q_sysinfo = device.getOutputQueue(name="sysinfo", maxSize=4, blocking=False)
37
38     while True:
39         info = q_sysinfo.get() # Blocking call, will wait until a new data has
↪ arrived
40         print_sys_info(info)
41
42         if cv2.waitKey(1) == ord('q'):
43             break
44

```

We're always happy to help with code or other questions you might have.

3.11.28 26.1 - RGB & MobilenetSSD with spatial data

This example shows how to run MobileNetV2SSD on the RGB input frame, and how to display both the RGB preview, detections, depth map and spatial information (X,Y,Z). It's similar to example '21_mobilenet_decoding_on_device' except it has spatial data. X,Y,Z coordinates are relative to the center of depth map.

setConfidenceThreshold - confidence threshold above which objects are detected

Demo

Setup

Please run the following command to install the required dependencies

```

python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai

```

For additional information, please follow [installation guide](#)

This example also requires MobilenetSSD blob (mobilenet-ssd_openvino_2021.2_6shave.blob file) to work - you can download it from [here](#)

Source code

Also available on [GitHub](#)

```

1  #!/usr/bin/env python3
2
3  from pathlib import Path
4  import sys
5  import cv2
6  import depthai as dai
7  import numpy as np
8  import time
9
10 '''
11 Spatial detection network demo.
12     Performs inference on RGB camera and retrieves spatial location coordinates: x,y,
13     ↪z relative to the center of depth map.
14 '''
15
16 # MobilenetSSD label texts
17 labelMap = ["background", "aeroplane", "bicycle", "bird", "boat", "bottle", "bus",
18     ↪"car", "cat", "chair", "cow",
19     "diningtable", "dog", "horse", "motorbike", "person", "pottedplant",
20     ↪"sheep", "sofa", "train", "tvmonitor"]
21
22 syncNN = True
23
24 # Get argument first
25 nnBlobPath = str((Path(__file__).parent / Path('models/mobilenet-ssd_openvino_2021.2_
26     ↪6shave.blob')).resolve().absolute())
27
28 if len(sys.argv) > 1:
29     nnBlobPath = sys.argv[1]
30
31 if not Path(nnBlobPath).exists():
32     import sys
33     raise FileNotFoundError(f'Required file/s not found, please run "{sys.executable}
34     ↪install_requirements.py"')
35
36 # Start defining a pipeline
37 pipeline = dai.Pipeline()
38
39 # Define a source - color camera
40 colorCam = pipeline.createColorCamera()
41 spatialDetectionNetwork = pipeline.createMobileNetSpatialDetectionNetwork()
42 monoLeft = pipeline.createMonoCamera()
43 monoRight = pipeline.createMonoCamera()
44 stereo = pipeline.createStereoDepth()
45
46 xoutRgb = pipeline.createXLinkOut()
47 xoutNN = pipeline.createXLinkOut()
48 xoutBoundingBoxDepthMapping = pipeline.createXLinkOut()
49 xoutDepth = pipeline.createXLinkOut()
50
51 xoutRgb.setStreamName("rgb")
52 xoutNN.setStreamName("detections")
53 xoutBoundingBoxDepthMapping.setStreamName("boundingBoxDepthMapping")
54 xoutDepth.setStreamName("depth")

```

(continues on next page)

(continued from previous page)

```

50
51 colorCam.setPreviewSize(300, 300)
52 colorCam.setResolution(dai.ColorCameraProperties.SensorResolution.THE_1080_P)
53 colorCam.setInterleaved(False)
54 colorCam.setColorOrder(dai.ColorCameraProperties.ColorOrder.BGR)
55
56 monoLeft.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
57 monoLeft.setBoardSocket(dai.CameraBoardSocket.LEFT)
58 monoRight.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
59 monoRight.setBoardSocket(dai.CameraBoardSocket.RIGHT)
60
61 # Setting node configs
62 stereo.setOutputDepth(True)
63 stereo.setConfidenceThreshold(255)
64
65 spatialDetectionNetwork.setBlobPath(nnBlobPath)
66 spatialDetectionNetwork.setConfidenceThreshold(0.5)
67 spatialDetectionNetwork.input.setBlocking(False)
68 spatialDetectionNetwork.setBoundingBoxScaleFactor(0.5)
69 spatialDetectionNetwork.setDepthLowerThreshold(100)
70 spatialDetectionNetwork.setDepthUpperThreshold(5000)
71
72 # Create outputs
73
74 monoLeft.out.link(stereo.left)
75 monoRight.out.link(stereo.right)
76
77 colorCam.preview.link(spatialDetectionNetwork.input)
78 if syncNN:
79     spatialDetectionNetwork.passthrough.link(xoutRgb.input)
80 else:
81     colorCam.preview.link(xoutRgb.input)
82
83 spatialDetectionNetwork.out.link(xoutNN.input)
84 spatialDetectionNetwork.boundingBoxMapping.link(xoutBoundingBoxDepthMapping.input)
85
86 stereo.depth.link(spatialDetectionNetwork.inputDepth)
87 spatialDetectionNetwork.passthroughDepth.link(xoutDepth.input)
88
89 # Pipeline is defined, now we can connect to the device
90 with dai.Device(pipeline) as device:
91     # Start pipeline
92     device.startPipeline()
93
94     # Output queues will be used to get the rgb frames and nn data from the outputs_
95     ↪ defined above
96     previewQueue = device.getOutputQueue(name="rgb", maxSize=4, blocking=False)
97     detectionNNQueue = device.getOutputQueue(name="detections", maxSize=4,
98     ↪ blocking=False)
99     xoutBoundingBoxDepthMapping = device.getOutputQueue(name="boundingBoxDepthMapping
100     ↪ ", maxSize=4, blocking=False)
101     depthQueue = device.getOutputQueue(name="depth", maxSize=4, blocking=False)
102
103     frame = None
104     detections = []
105
106     startTime = time.monotonic()

```

(continues on next page)

(continued from previous page)

```

104     counter = 0
105     fps = 0
106     color = (255, 255, 255)
107
108     while True:
109         inPreview = previewQueue.get()
110         inNN = detectionNNQueue.get()
111         depth = depthQueue.get()
112
113         counter+=1
114         current_time = time.monotonic()
115         if (current_time - startTime) > 1 :
116             fps = counter / (current_time - startTime)
117             counter = 0
118             startTime = current_time
119
120         frame = inPreview.getCvFrame()
121         depthFrame = depth.getFrame()
122
123         depthFrameColor = cv2.normalize(depthFrame, None, 255, 0, cv2.NORM_INF, cv2.
↳CV_8UC1)
124         depthFrameColor = cv2.equalizeHist(depthFrameColor)
125         depthFrameColor = cv2.applyColorMap(depthFrameColor, cv2.COLORMAP_HOT)
126         detections = inNN.detections
127         if len(detections) != 0:
128             boundingBoxMapping = xoutBoundingBoxDepthMapping.get()
129             roiDatas = boundingBoxMapping.getConfigData()
130
131             for roiData in roiDatas:
132                 roi = roiData.roi
133                 roi = roi.denormalize(depthFrameColor.shape[1], depthFrameColor.
↳shape[0])
134                 topLeft = roi.topLeft()
135                 bottomRight = roi.bottomRight()
136                 xmin = int(topLeft.x)
137                 ymin = int(topLeft.y)
138                 xmax = int(bottomRight.x)
139                 ymax = int(bottomRight.y)
140
141                 cv2.rectangle(depthFrameColor, (xmin, ymin), (xmax, ymax), color, cv2.
↳FONT_HERSHEY_SCRIPT_SIMPLEX)
142
143
144         # If the frame is available, draw bounding boxes on it and show the frame
145         height = frame.shape[0]
146         width = frame.shape[1]
147         for detection in detections:
148             # Denormalize bounding box
149             x1 = int(detection.xmin * width)
150             x2 = int(detection.xmax * width)
151             y1 = int(detection.ymin * height)
152             y2 = int(detection.ymax * height)
153             try:
154                 label = labelMap[detection.label]
155             except:
156                 label = detection.label
157             cv2.putText(frame, str(label), (x1 + 10, y1 + 20), cv2.FONT_HERSHEY_
↳TRIPLEX, 0.5, color)

```

(continues on next page)

(continued from previous page)

```

158         cv2.putText(frame, "{:.2f}".format(detection.confidence*100), (x1 + 10,
↪y1 + 35), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
159         cv2.putText(frame, f"X: {int(detection.spatialCoordinates.x)} mm", (x1 +
↪10, y1 + 50), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
160         cv2.putText(frame, f"Y: {int(detection.spatialCoordinates.y)} mm", (x1 +
↪10, y1 + 65), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
161         cv2.putText(frame, f"Z: {int(detection.spatialCoordinates.z)} mm", (x1 +
↪10, y1 + 80), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
162
163         cv2.rectangle(frame, (x1, y1), (x2, y2), color, cv2.FONT_HERSHEY_SIMPLEX)
164
165         cv2.putText(frame, "NN fps: {:.2f}".format(fps), (2, frame.shape[0] - 4), cv2.
↪FONT_HERSHEY_TRIPLEX, 0.4, color)
166         cv2.imshow("depth", depthFrameColor)
167         cv2.imshow("rgb", frame)
168
169         if cv2.waitKey(1) == ord('q'):
170             break

```

We're always happy to help with code or other questions you might have.

3.11.29 26.2 - Mono & MobilenetSSD with spatial data

This example shows how to run MobileNetV2SSD on the rectified right input frame, and how to display both the preview, detections, depth map and spatial information (X,Y,Z). It's similar to example '21_mobilenet_decoding_on_device' except it has spatial data. X,Y,Z coordinates are relative to the center of depth map.

setConfidenceThreshold - confidence threshold above which objects are detected

Demo

Setup

Please run the following command to install the required dependencies

```

python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai

```

For additional information, please follow [installation guide](#)

This example also requires MobilenetSSD blob (mobilenet-ssd_openvino_2021.2_6shave.blob file) to work - you can download it from [here](#)

Source code

Also available on GitHub

```

1  #!/usr/bin/env python3
2
3  from pathlib import Path
4  import sys
5  import cv2
6  import depthai as dai
7  import numpy as np
8  import time
9
10 '''
11 Mobilenet SSD device side decoding demo
12 The "mobilenet-ssd" model is a Single-Shot multibox Detection (SSD) network intended
13 to perform object detection. This model is implemented using the Caffe* framework.
14 For details about this model, check out the repository <https://github.com/
↳chuanqi305/MobileNet-SSD>.
15 '''
16
17 # MobilenetSSD label texts
18 labelMap = ["background", "aeroplane", "bicycle", "bird", "boat", "bottle", "bus",
↳"car", "cat", "chair", "cow",
19             "diningtable", "dog", "horse", "motorbike", "person", "pottedplant",
↳"sheep", "sofa", "train", "tvmonitor"]
20
21 syncNN = True
22 flipRectified = True
23
24 # Get argument first
25 nnPath = str((Path(__file__).parent / Path('models/mobilenet-ssd_openvino_2021.2_
↳6shave.blob')).resolve().absolute())
26 if len(sys.argv) > 1:
27     nnPath = sys.argv[1]
28
29 if not Path(nnPath).exists():
30     import sys
31     raise FileNotFoundError(f'Required file/s not found, please run "{sys.executable}
↳install_requirements.py"')
32
33 # Start defining a pipeline
34 pipeline = dai.Pipeline()
35
36
37 manip = pipeline.createImageManip()
38 manip.initialConfig.setResize(300, 300)
39 # The NN model expects BGR input. By default ImageManip output type would be same as
↳input (gray in this case)
40 manip.initialConfig.setFrameType(dai.ImgFrame.Type.BGR888p)
41 # manip.setKeepAspectRatio(False)
42
43 # Define a neural network that will make predictions based on the source frames
44 spatialDetectionNetwork = pipeline.createMobileNetSpatialDetectionNetwork()
45 spatialDetectionNetwork.setConfidenceThreshold(0.5)
46 spatialDetectionNetwork.setBlobPath(nnPath)
47 spatialDetectionNetwork.input.setBlocking(False)
48 spatialDetectionNetwork.setBoundingBoxScaleFactor(0.5)

```

(continues on next page)

(continued from previous page)

```

49 spatialDetectionNetwork.setDepthLowerThreshold(100)
50 spatialDetectionNetwork.setDepthUpperThreshold(5000)
51
52 manip.out.link(spatialDetectionNetwork.input)
53
54 # Create outputs
55 xoutManip = pipeline.createXLinkOut()
56 xoutManip.setStreamName("right")
57 if syncNN:
58     spatialDetectionNetwork.passthrough.link(xoutManip.input)
59 else:
60     manip.out.link(xoutManip.input)
61
62 depthRoiMap = pipeline.createXLinkOut()
63 depthRoiMap.setStreamName("boundingBoxDepthMapping")
64
65 xoutDepth = pipeline.createXLinkOut()
66 xoutDepth.setStreamName("depth")
67
68 nnOut = pipeline.createXLinkOut()
69 nnOut.setStreamName("detections")
70 spatialDetectionNetwork.out.link(nnOut.input)
71 spatialDetectionNetwork.boundingBoxMapping.link(depthRoiMap.input)
72
73 monoLeft = pipeline.createMonoCamera()
74 monoRight = pipeline.createMonoCamera()
75 stereo = pipeline.createStereoDepth()
76 monoLeft.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
77 monoLeft.setBoardSocket(dai.CameraBoardSocket.LEFT)
78 monoRight.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
79 monoRight.setBoardSocket(dai.CameraBoardSocket.RIGHT)
80 stereo.setOutputDepth(True)
81 stereo.setConfidenceThreshold(255)
82 stereo.setOutputRectified(True)
83
84 stereo.rectifiedRight.link(manip.inputImage)
85
86 monoLeft.out.link(stereo.left)
87 monoRight.out.link(stereo.right)
88
89 stereo.depth.link(spatialDetectionNetwork.inputDepth)
90 spatialDetectionNetwork.passthroughDepth.link(xoutDepth.input)
91
92 # Pipeline is defined, now we can connect to the device
93 with dai.Device(pipeline) as device:
94     # Start pipeline
95     device.startPipeline()
96
97     # Output queues will be used to get the rgb frames and nn data from the outputs_
98     ↳ defined above
99     previewQueue = device.getOutputQueue(name="right", maxSize=4, blocking=False)
100     detectionNNQueue = device.getOutputQueue(name="detections", maxSize=4,
101     ↳ blocking=False)
102     depthRoiMap = device.getOutputQueue(name="boundingBoxDepthMapping", maxSize=4,
103     ↳ blocking=False)
104     depthQueue = device.getOutputQueue(name="depth", maxSize=4, blocking=False)

```

(continues on next page)

(continued from previous page)

```

103     rectifiedRight = None
104     detections = []
105
106     startTime = time.monotonic()
107     counter = 0
108     fps = 0
109     color = (255, 255, 255)
110
111     while True:
112         inRectified = previewQueue.get()
113         det = detectionNNQueue.get()
114         depth = depthQueue.get()
115
116         counter += 1
117         currentTime = time.monotonic()
118         if (currentTime - startTime) > 1:
119             fps = counter / (currentTime - startTime)
120             counter = 0
121             startTime = currentTime
122
123         rectifiedRight = inRectified.getCvFrame()
124
125         depthFrame = depth.getFrame()
126
127         depthFrameColor = cv2.normalize(depthFrame, None, 255, 0, cv2.NORM_INF, cv2.
↪CV_8UC1)
128         depthFrameColor = cv2.equalizeHist(depthFrameColor)
129         depthFrameColor = cv2.applyColorMap(depthFrameColor, cv2.COLORMAP_HOT)
130         detections = det.detections
131         if len(detections) != 0:
132             boundingBoxMapping = depthRoiMap.get()
133             roiDatas = boundingBoxMapping.getConfigData()
134
135             for roiData in roiDatas:
136                 roi = roiData.roi
137                 roi = roi.denormalize(depthFrameColor.shape[1], depthFrameColor.
↪shape[0])
138                 topLeft = roi.topLeft()
139                 bottomRight = roi.bottomRight()
140                 xmin = int(topLeft.x)
141                 ymin = int(topLeft.y)
142                 xmax = int(bottomRight.x)
143                 ymax = int(bottomRight.y)
144                 cv2.rectangle(depthFrameColor, (xmin, ymin), (xmax, ymax), color, cv2.
↪FONT_HERSHEY_SCRIPT_SIMPLEX)
145
146                 if flipRectified:
147                     rectifiedRight = cv2.flip(rectifiedRight, 1)
148
149                 # If the rectifiedRight is available, draw bounding boxes on it and show the
↪rectifiedRight
150                 height = rectifiedRight.shape[0]
151                 width = rectifiedRight.shape[1]
152                 for detection in detections:
153                     if flipRectified:
154                         swap = detection.xmin
155                         detection.xmin = 1 - detection.xmax

```

(continues on next page)

(continued from previous page)

```

156         detection.xmax = 1 - swap
157         # Denormalize bounding box
158         x1 = int(detection.xmin * width)
159         x2 = int(detection.xmax * width)
160         y1 = int(detection.ymin * height)
161         y2 = int(detection.ymax * height)
162
163         try:
164             label = labelMap[detection.label]
165         except:
166             label = detection.label
167
168         cv2.putText(rectifiedRight, str(label), (x1 + 10, y1 + 20), cv2.FONT_
↪HERSHEY_TRIPLEX, 0.5, color)
169         cv2.putText(rectifiedRight, "{:.2f}".format(detection.confidence*100),
↪(x1 + 10, y1 + 35), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
170         cv2.putText(rectifiedRight, f"X: {int(detection.spatialCoordinates.x)} mm
↪", (x1 + 10, y1 + 50), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
171         cv2.putText(rectifiedRight, f"Y: {int(detection.spatialCoordinates.y)} mm
↪", (x1 + 10, y1 + 65), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
172         cv2.putText(rectifiedRight, f"Z: {int(detection.spatialCoordinates.z)} mm
↪", (x1 + 10, y1 + 80), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
173
174         cv2.rectangle(rectifiedRight, (x1, y1), (x2, y2), color, cv2.FONT_HERSHEY_
↪SIMPLEX)
175
176         cv2.putText(rectifiedRight, "NN fps: {:.2f}".format(fps), (2, rectifiedRight.
↪shape[0] - 4), cv2.FONT_HERSHEY_TRIPLEX, 0.4, color)
177         cv2.imshow("depth", depthFrameColor)
178         cv2.imshow("rectified right", rectifiedRight)
179
180         if cv2.waitKey(1) == ord('q'):
181             break

```

We're always happy to help with code or other questions you might have.

3.11.30 26.3 - RGB & TinyYolo with spatial data

This example shows how to run TinyYoloV3 and v4 on the RGB input frame, and how to display both the RGB preview, detections, depth map and spatial information (X,Y,Z). It's similar to example '26_1_spatial_mobilenet' except it is running TinyYolo network. X,Y,Z coordinates are relative to the center of depth map.

setNumClasses - number of YOLO classes setCoordinateSize - size of coordinate setAnchors - yolo anchors setAnchorMasks - anchorMasks26, anchorMasks13 (anchorMasks52 - additionally for full YOLOv4) setIouThreshold - intersection over union threshold setConfidenceThreshold - confidence threshold above which objects are detected

Demo

Setup

Please run the following command to install the required dependencies

```
python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai
```

For additional information, please follow *installation guide*

This example also requires YOLOv4-tiny blob (tiny-yolo-v4_openvino_2021.2_6shave.blob file) to work - you can download it from [here](#)

YOLOv3-tiny blob (tiny-yolo-v3_openvino_2021.2_6shave.blob file) can be used too - you can download it from [here](#)

Source code

Also available on GitHub

```

1  #!/usr/bin/env python3
2
3  from pathlib import Path
4  import sys
5  import cv2
6  import depthai as dai
7  import numpy as np
8  import time
9
10
11  '''
12  Spatial Tiny-yolo example
13  Performs inference on RGB camera and retrieves spatial location coordinates: x,y,z_
14  ↳relative to the center of depth map.
15  Can be used for tiny-yolo-v3 or tiny-yolo-v4 networks
16  '''
17
18  # Tiny yolo v3/4 label texts
19  labelMap = [
20      "person",      "bicycle",    "car",            "motorbike",     "aeroplane",
21      ↳"bus",         "train",        "traffic light",  "fire hydrant",  "stop sign",
22      "truck",       "boat",         "parking meter", "bench",         "dog",
23      ↳"bird",        "cat",          "horse",         "sheep",
24      "cow",         "elephant",     "giraffe",       "backpack",      "umbrella",
25      ↳"bear",        "zebra",        "skis",          "snowboard",    "sports ball",
26      "suitcase",    "frisbee",     "kite",          "baseball bat",  "baseball glove",
27      ↳"wine glass",  "cup",         "skateboard",    "surfboard",    "tennis racket",
28      "fork",        "knife",       "bottle",        "bottle",
29      ↳"apple",       "sandwich",    "spoon",         "bowl",         "pizza",
30      "orange",     "broccoli",   "carrot",        "hot dog",
31      ↳"donut",      "cake",       "pottedplant",  "bed",          "diningtable",
32      "chair",      "sofa",       "toilet",       "tvmonitor",

```

(continues on next page)

(continued from previous page)

```

27     "laptop",          "mouse",          "remote",          "keyboard",        "cell phone",
    ↪ "microwave",        "oven",            "refrigerator",    "book",            "clock",
28     "toaster",         "sink",            "refrigerator",    "book",            "clock",
    ↪ "vase",              "scissors",
29     "teddy bear",      "hair drier",      "toothbrush"
30 ]
31
32 syncNN = True
33
34 # Get argument first
35 nnBlobPath = str((Path(__file__).parent / Path('models/tiny-yolo-v4_openvino_2021.2_
    ↪ 6shave.blob')).resolve().absolute())
36 if len(sys.argv) > 1:
37     nnBlobPath = sys.argv[1]
38
39 if not Path(nnBlobPath).exists():
40     import sys
41     raise FileNotFoundError(f'Required file/s not found, please run "{sys.executable}
    ↪ install_requirements.py"')
42
43 # Start defining a pipeline
44 pipeline = dai.Pipeline()
45
46 # Define a source - color camera
47 colorCam = pipeline.createColorCamera()
48 spatialDetectionNetwork = pipeline.createYoloSpatialDetectionNetwork()
49 monoLeft = pipeline.createMonoCamera()
50 monoRight = pipeline.createMonoCamera()
51 stereo = pipeline.createStereoDepth()
52
53 xoutRgb = pipeline.createXLinkOut()
54 xoutNN = pipeline.createXLinkOut()
55 xoutBoundingBoxDepthMapping = pipeline.createXLinkOut()
56 xoutDepth = pipeline.createXLinkOut()
57
58 xoutRgb.setStreamName("rgb")
59 xoutNN.setStreamName("detections")
60 xoutBoundingBoxDepthMapping.setStreamName("boundingBoxDepthMapping")
61 xoutDepth.setStreamName("depth")
62
63
64 colorCam.setPreviewSize(416, 416)
65 colorCam.setResolution(dai.ColorCameraProperties.SensorResolution.THE_1080_P)
66 colorCam.setInterleaved(False)
67 colorCam.setColorOrder(dai.ColorCameraProperties.ColorOrder.BGR)
68
69 monoLeft.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
70 monoLeft.setBoardSocket(dai.CameraBoardSocket.LEFT)
71 monoRight.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
72 monoRight.setBoardSocket(dai.CameraBoardSocket.RIGHT)
73
74 # setting node configs
75 stereo.setOutputDepth(True)
76 stereo.setConfidenceThreshold(255)
77
78 spatialDetectionNetwork.setBlobPath(nnBlobPath)
79 spatialDetectionNetwork.setConfidenceThreshold(0.5)

```

(continues on next page)

(continued from previous page)

```

80 spatialDetectionNetwork.input.setBlocking(False)
81 spatialDetectionNetwork.setBoundingBoxScaleFactor(0.5)
82 spatialDetectionNetwork.setDepthLowerThreshold(100)
83 spatialDetectionNetwork.setDepthUpperThreshold(5000)
84 # Yolo specific parameters
85 spatialDetectionNetwork.setNumClasses(80)
86 spatialDetectionNetwork.setCoordinateSize(4)
87 spatialDetectionNetwork.setAnchors(np.array([10,14, 23,27, 37,58, 81,82, 135,169, 344,
88 ↪319]))
89 spatialDetectionNetwork.setAnchorMasks({ "side26": np.array([1,2,3]), "side13": np.
90 ↪array([3,4,5]) })
91 spatialDetectionNetwork.setIouThreshold(0.5)
92
93 # Create outputs
94 monoLeft.out.link(stereo.left)
95 monoRight.out.link(stereo.right)
96
97 colorCam.preview.link(spatialDetectionNetwork.input)
98 if syncNN:
99     spatialDetectionNetwork.passthrough.link(xoutRgb.input)
100 else:
101     colorCam.preview.link(xoutRgb.input)
102
103 spatialDetectionNetwork.out.link(xoutNN.input)
104 spatialDetectionNetwork.boundingBoxMapping.link(xoutBoundingBoxDepthMapping.input)
105
106 stereo.depth.link(spatialDetectionNetwork.inputDepth)
107 spatialDetectionNetwork.passthroughDepth.link(xoutDepth.input)
108
109 # Pipeline is defined, now we can connect to the device
110 with dai.Device(pipeline) as device:
111     # Start pipeline
112     device.startPipeline()
113
114     # Output queues will be used to get the rgb frames and nn data from the outputs_
115     ↪defined above
116     previewQueue = device.getOutputQueue(name="rgb", maxSize=4, blocking=False)
117     detectionNNQueue = device.getOutputQueue(name="detections", maxSize=4,
118     ↪blocking=False)
119     xoutBoundingBoxDepthMapping = device.getOutputQueue(name="boundingBoxDepthMapping
120     ↪", maxSize=4, blocking=False)
121     depthQueue = device.getOutputQueue(name="depth", maxSize=4, blocking=False)
122
123     frame = None
124     detections = []
125
126     startTime = time.monotonic()
127     counter = 0
128     fps = 0
129     color = (255, 255, 255)
130
131     while True:
132         inPreview = previewQueue.get()
133         inNN = detectionNNQueue.get()
134         depth = depthQueue.get()

```

(continues on next page)

(continued from previous page)

```

132     counter+=1
133     current_time = time.monotonic()
134     if (current_time - startTime) > 1 :
135         fps = counter / (current_time - startTime)
136         counter = 0
137         startTime = current_time
138
139     frame = inPreview.getCvFrame()
140     depthFrame = depth.getFrame()
141
142     depthFrameColor = cv2.normalize(depthFrame, None, 255, 0, cv2.NORM_INF, cv2.
↳CV_8UC1)
143     depthFrameColor = cv2.equalizeHist(depthFrameColor)
144     depthFrameColor = cv2.applyColorMap(depthFrameColor, cv2.COLORMAP_HOT)
145     detections = inNN.detections
146     if len(detections) != 0:
147         boundingBoxMapping = xoutBoundingBoxDepthMapping.get()
148         roiDatas = boundingBoxMapping.getConfigData()
149
150         for roiData in roiDatas:
151             roi = roiData.roi
152             roi = roi.denormalize(depthFrameColor.shape[1], depthFrameColor.
↳shape[0])
153             topLeft = roi.topLeft()
154             bottomRight = roi.bottomRight()
155             xmin = int(topLeft.x)
156             ymin = int(topLeft.y)
157             xmax = int(bottomRight.x)
158             ymax = int(bottomRight.y)
159
160             cv2.rectangle(depthFrameColor, (xmin, ymin), (xmax, ymax), color, cv2.
↳FONT_HERSHEY_SCRIPT_SIMPLEX)
161
162     # If the frame is available, draw bounding boxes on it and show the frame
163     height = frame.shape[0]
164     width = frame.shape[1]
165     for detection in detections:
166         # Denormalize bounding box
167         x1 = int(detection.xmin * width)
168         x2 = int(detection.xmax * width)
169         y1 = int(detection.ymin * height)
170         y2 = int(detection.ymax * height)
171         try:
172             label = labelMap[detection.label]
173         except:
174             label = detection.label
175         cv2.putText(frame, str(label), (x1 + 10, y1 + 20), cv2.FONT_HERSHEY_
↳TRIPLEX, 0.5, color)
176         cv2.putText(frame, "{:.2f}".format(detection.confidence*100), (x1 + 10,
↳y1 + 35), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
177         cv2.putText(frame, f"X: {int(detection.spatialCoordinates.x)} mm", (x1 +
↳10, y1 + 50), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
178         cv2.putText(frame, f"Y: {int(detection.spatialCoordinates.y)} mm", (x1 +
↳10, y1 + 65), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
179         cv2.putText(frame, f"Z: {int(detection.spatialCoordinates.z)} mm", (x1 +
↳10, y1 + 80), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)

```

(continues on next page)

(continued from previous page)

```

181         cv2.rectangle(frame, (x1, y1), (x2, y2), color, cv2.FONT_HERSHEY_SIMPLEX)
182
183     cv2.putText(frame, "NN fps: {:.2f}".format(fps), (2, frame.shape[0] - 4), cv2.
184     FONT_HERSHEY_TRIPLEX, 0.4, color)
185     cv2.imshow("depth", depthFrameColor)
186     cv2.imshow("rgb", frame)
187
188     if cv2.waitKey(1) == ord('q'):
189         break

```

We're always happy to help with code or other questions you might have.

3.11.31 27 - Spatial location calculator

This example shows how to retrieve spatial location data (X,Y,Z) on a runtime configurable ROI. You can move the ROI using WASD keys. X,Y,Z coordinates are relative to the center of depth map.

setConfidenceThreshold - confidence threshold above which objects are detected

Demo

Setup

Please run the following command to install the required dependencies

```

python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai

```

For additional information, please follow [installation guide](#)

Source code

Also available on [GitHub](#)

```

1  #!/usr/bin/env python3
2
3  import cv2
4  import depthai as dai
5
6  stepSize = 0.05
7
8  # Start defining a pipeline
9  pipeline = dai.Pipeline()
10
11 # Define a source - two mono (grayscale) cameras
12 monoLeft = pipeline.createMonoCamera()
13 monoRight = pipeline.createMonoCamera()
14 stereo = pipeline.createStereoDepth()
15 spatialLocationCalculator = pipeline.createSpatialLocationCalculator()
16
17 xoutDepth = pipeline.createXLinkOut()

```

(continues on next page)

(continued from previous page)

```

18 xoutSpatialData = pipeline.createXLinkOut()
19 xinSpatialCalcConfig = pipeline.createXLinkIn()
20
21 xoutDepth.setStreamName("depth")
22 xoutSpatialData.setStreamName("spatialData")
23 xinSpatialCalcConfig.setStreamName("spatialCalcConfig")
24
25 # MonoCamera
26 monoLeft.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
27 monoLeft.setBoardSocket(dai.CameraBoardSocket.LEFT)
28 monoRight.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
29 monoRight.setBoardSocket(dai.CameraBoardSocket.RIGHT)
30
31 outputDepth = True
32 outputRectified = False
33 lrcheck = False
34 subpixel = False
35
36 # StereoDepth
37 stereo.setOutputDepth(outputDepth)
38 stereo.setOutputRectified(outputRectified)
39 stereo.setConfidenceThreshold(255)
40
41 stereo.setLeftRightCheck(lrcheck)
42 stereo.setSubpixel(subpixel)
43
44 monoLeft.out.link(stereo.left)
45 monoRight.out.link(stereo.right)
46
47 spatialLocationCalculator.passthroughDepth.link(xoutDepth.input)
48 stereo.depth.link(spatialLocationCalculator.inputDepth)
49
50 topLeft = dai.Point2f(0.4, 0.4)
51 bottomRight = dai.Point2f(0.6, 0.6)
52
53 spatialLocationCalculator.setWaitForConfigInput(False)
54 config = dai.SpatialLocationCalculatorConfigData()
55 config.depthThresholds.lowerThreshold = 100
56 config.depthThresholds.upperThreshold = 10000
57 config.roi = dai.Rect(topLeft, bottomRight)
58 spatialLocationCalculator.initialConfig.addROI(config)
59 spatialLocationCalculator.out.link(xoutSpatialData.input)
60 xinSpatialCalcConfig.out.link(spatialLocationCalculator.inputConfig)
61
62 # Pipeline is defined, now we can connect to the device
63 with dai.Device(pipeline) as device:
64     device.startPipeline()
65
66     # Output queue will be used to get the depth frames from the outputs defined above
67     depthQueue = device.getOutputQueue(name="depth", maxSize=4, blocking=False)
68     spatialCalcQueue = device.getOutputQueue(name="spatialData", maxSize=4,
69     ↪blocking=False)
69     spatialCalcConfigInQueue = device.getInputQueue("spatialCalcConfig")
70
71     color = (255, 255, 255)
72
73     print("Use WASD keys to move ROI!")

```

(continues on next page)

(continued from previous page)

```

74
75     while True:
76         inDepth = depthQueue.get() # Blocking call, will wait until a new data has_
↳arrived
77         inDepthAvg = spatialCalcQueue.get() # Blocking call, will wait until a new_
↳data has arrived
78
79         depthFrame = inDepth.getFrame()
80         depthFrameColor = cv2.normalize(depthFrame, None, 255, 0, cv2.NORM_INF, cv2.
↳CV_8UC1)
81         depthFrameColor = cv2.equalizeHist(depthFrameColor)
82         depthFrameColor = cv2.applyColorMap(depthFrameColor, cv2.COLORMAP_HOT)
83
84         spatialData = inDepthAvg.getSpatialLocations()
85         for depthData in spatialData:
86             roi = depthData.config.roi
87             roi = roi.denormalize(width=depthFrameColor.shape[1],
↳height=depthFrameColor.shape[0])
88             xmin = int(roi.topLeft().x)
89             ymin = int(roi.topLeft().y)
90             xmax = int(roi.bottomRight().x)
91             ymax = int(roi.bottomRight().y)
92
93             fontType = cv2.FONT_HERSHEY_TRIPLEX
94             cv2.rectangle(depthFrameColor, (xmin, ymin), (xmax, ymax), color, cv2.
↳FONT_HERSHEY_SCRIPT_SIMPLEX)
95             cv2.putText(depthFrameColor, f"X: {int(depthData.spatialCoordinates.x)} mm
↳", (xmin + 10, ymin + 20), fontType, 0.5, color)
96             cv2.putText(depthFrameColor, f"Y: {int(depthData.spatialCoordinates.y)} mm
↳", (xmin + 10, ymin + 35), fontType, 0.5, color)
97             cv2.putText(depthFrameColor, f"Z: {int(depthData.spatialCoordinates.z)} mm
↳", (xmin + 10, ymin + 50), fontType, 0.5, color)
98
99
100         cv2.imshow("depth", depthFrameColor)
101
102         newConfig = False
103         key = cv2.waitKey(1)
104         if key == ord('q'):
105             break
106         elif key == ord('w'):
107             if topLeft.y - stepSize >= 0:
108                 topLeft.y -= stepSize
109                 bottomRight.y -= stepSize
110                 newConfig = True
111         elif key == ord('a'):
112             if topLeft.x - stepSize >= 0:
113                 topLeft.x -= stepSize
114                 bottomRight.x -= stepSize
115                 newConfig = True
116         elif key == ord('s'):
117             if bottomRight.y + stepSize <= 1:
118                 topLeft.y += stepSize
119                 bottomRight.y += stepSize
120                 newConfig = True
121         elif key == ord('d'):
122             if bottomRight.x + stepSize <= 1:

```

(continues on next page)

(continued from previous page)

```

123         topLeft.x += stepSize
124         bottomRight.x += stepSize
125         newConfig = True
126
127     if newConfig:
128         config.roi = dai.Rect(topLeft, bottomRight)
129         cfg = dai.SpatialLocationCalculatorConfig()
130         cfg.addROI(config)
131         spatialCalcConfigInQueue.send(cfg)

```

We're always happy to help with code or other questions you might have.

3.11.32 28 - Camera video high resolution

This example shows how to use high resolution video at low latency. Compared to *01 - RGB Preview*, this demo outputs NV12 frames whereas preview frames are BGR and are not suited for larger resolution (eg. 2000x1000). Preview is more suitable for either NN or visualization purposes.

Demo

Setup

Please run the following command to install the required dependencies

```

python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai

```

For additional information, please follow [installation guide](#)

Source code

Also available on [GitHub](#)

```

1  #!/usr/bin/env python3
2
3  import cv2
4  import depthai as dai
5  import numpy as np
6
7  # Start defining a pipeline
8  pipeline = dai.Pipeline()
9
10 # Define a source - color camera
11 colorCam = pipeline.createColorCamera()
12 colorCam.setBoardSocket(dai.CameraBoardSocket.RGB)
13 colorCam.setResolution(dai.ColorCameraProperties.SensorResolution.THE_1080_P)
14 colorCam.setVideoSize(1920, 1080)
15
16 # Create output
17 xoutVideo = pipeline.createXLinkOut()
18 xoutVideo.setStreamName("video")
19 xoutVideo.input.setBlocking(False)

```

(continues on next page)

(continued from previous page)

```

20 xoutVideo.input.setQueueSize(1)
21
22 colorCam.video.link(xoutVideo.input)
23
24 # Pipeline is defined, now we can connect to the device
25 with dai.Device(pipeline) as device:
26     # Start pipeline
27     device.startPipeline()
28     video = device.getOutputQueue(name="video", maxSize=1, blocking=False)
29
30     while True:
31         # Get preview and video frames
32         videoIn = video.get()
33
34         # Get BGR frame from NV12 encoded video frame to show with opencv
35         # Visualizing the frame on slower hosts might have overhead
36         cv2.imshow("video", videoIn.getCvFrame())
37
38         if cv2.waitKey(1) == ord('q'):
39             break

```

We're always happy to help with code or other questions you might have.

3.11.33 29.1 - Object tracker on RGB camera

This example shows how to run MobileNetV2SSD on the RGB input frame, and perform object tracking on persons.

Demo

Setup

Please run the following command to install the required dependencies

```

python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai

```

For additional information, please follow [installation guide](#)

This example also requires MobilenetSDD blob (mobilenet.blob file) to work - you can download it from [here](#)

Source code

Also available on GitHub

```

1 #!/usr/bin/env python3
2
3 from pathlib import Path
4 import cv2
5 import depthai as dai
6 import numpy as np
7 import time
8 import argparse

```

(continues on next page)

(continued from previous page)

```

9
10 labelMap = ["background", "aeroplane", "bicycle", "bird", "boat", "bottle", "bus",
11 ↪ "car", "cat", "chair", "cow",
12 ↪ "diningtable", "dog", "horse", "motorbike", "person", "pottedplant",
13 ↪ "sheep", "sofa", "train", "tvmonitor"]
14
15 nnPathDefault = str((Path(__file__).parent / Path('models/mobilenet-ssd_openvino_2021.
16 ↪ 2_6shave.blob')).resolve().absolute())
17
18 parser = argparse.ArgumentParser()
19 parser.add_argument('nnPath', nargs='?', help="Path to mobilenet detection network_
20 ↪ blob", default=nnPathDefault)
21
22 parser.add_argument('-ff', '--full_frame', action="store_true", help="Perform_
23 ↪ tracking on full RGB frame", default=False)
24
25 args = parser.parse_args()
26
27
28 fullFrameTracking = args.full_frame
29
30 # Start defining a pipeline
31 pipeline = dai.Pipeline()
32
33
34 colorCam = pipeline.createColorCamera()
35 detectionNetwork = pipeline.createMobileNetDetectionNetwork()
36 objectTracker = pipeline.createObjectTracker()
37 trackerOut = pipeline.createXLinkOut()
38
39
40 xlinkOut = pipeline.createXLinkOut()
41
42 xlinkOut.setStreamName("preview")
43 trackerOut.setStreamName("tracklets")
44
45
46 colorCam.setPreviewSize(300, 300)
47 colorCam.setResolution(dai.ColorCameraProperties.SensorResolution.THE_1080_P)
48 colorCam.setInterleaved(False)
49 colorCam.setColorOrder(dai.ColorCameraProperties.ColorOrder.BGR)
50 colorCam.setFps(40)
51
52 # setting node configs
53 detectionNetwork.setBlobPath(args.nnPath)
54 detectionNetwork.setConfidenceThreshold(0.5)
55 detectionNetwork.input.setBlocking(False)
56
57 # Link plugins CAM . NN . XLINK
58 colorCam.preview.link(detectionNetwork.input)
59 objectTracker.passthroughTrackerFrame.link(xlinkOut.input)
60
61
62 objectTracker.setDetectionLabelsToTrack([15]) # track only person
63 # possible tracking types: ZERO_TERM_COLOR_HISTOGRAM, ZERO_TERM_IMAGELESS
64 objectTracker.setTrackerType(dai.TrackerType.ZERO_TERM_COLOR_HISTOGRAM)
65 # take the smallest ID when new object is tracked, possible options: SMALLEST_ID,
66 ↪ UNIQUE_ID
67 objectTracker.setTrackerIdAssignmentPolicy(dai.TrackerIdAssignmentPolicy.SMALLEST_ID)
68
69
70 if fullFrameTracking:
71     colorCam.video.link(objectTracker.inputTrackerFrame)

```

(continues on next page)

(continued from previous page)

```

60 else:
61     detectionNetwork.passthrough.link(objectTracker.inputTrackerFrame)
62
63 detectionNetwork.passthrough.link(objectTracker.inputDetectionFrame)
64 detectionNetwork.out.link(objectTracker.inputDetections)
65 objectTracker.out.link(trackerOut.input)
66
67
68 # Pipeline defined, now the device is connected to
69 with dai.Device(pipeline) as device:
70
71     # Start the pipeline
72     device.startPipeline()
73
74     preview = device.getOutputQueue("preview", 4, False)
75     tracklets = device.getOutputQueue("tracklets", 4, False)
76
77     startTime = time.monotonic()
78     counter = 0
79     fps = 0
80     frame = None
81
82     while(True):
83         imgFrame = preview.get()
84         track = tracklets.get()
85
86         counter+=1
87         current_time = time.monotonic()
88         if (current_time - startTime) > 1 :
89             fps = counter / (current_time - startTime)
90             counter = 0
91             startTime = current_time
92
93         color = (255, 0, 0)
94         frame = imgFrame.getCvFrame()
95         trackletsData = track.tracklets
96         for t in trackletsData:
97             roi = t.roi.denormalize(frame.shape[1], frame.shape[0])
98             x1 = int(roi.topLeft().x)
99             y1 = int(roi.topLeft().y)
100             x2 = int(roi.bottomRight().x)
101             y2 = int(roi.bottomRight().y)
102
103             try:
104                 label = labelMap[t.label]
105             except:
106                 label = t.label
107
108             statusMap = {dai.Tracklet.TrackingStatus.NEW : "NEW", dai.Tracklet.
↪TrackingStatus.TRACKED : "TRACKED", dai.Tracklet.TrackingStatus.LOST : "LOST"}
109             cv2.putText(frame, str(label), (x1 + 10, y1 + 20), cv2.FONT_HERSHEY_
↪TRIPLEX, 0.5, color)
110             cv2.putText(frame, f"ID: {[t.id]}", (x1 + 10, y1 + 35), cv2.FONT_HERSHEY_
↪TRIPLEX, 0.5, color)
111             cv2.putText(frame, statusMap[t.status], (x1 + 10, y1 + 50), cv2.FONT_
↪HERSHEY_TRIPLEX, 0.5, color)
112             cv2.rectangle(frame, (x1, y1), (x2, y2), color, cv2.FONT_HERSHEY_SIMPLEX)

```

(continues on next page)

(continued from previous page)

```

113         cv2.putText(frame, "NN fps: {:.2f}".format(fps), (2, frame.shape[0] - 4), cv2.
114         ↪FONT_HERSHEY_TRIPLEX, 0.4, color)
115
116         cv2.imshow("tracker", frame)
117
118         if cv2.waitKey(1) == ord('q'):
119             break

```

We're always happy to help with code or other questions you might have.

3.11.34 29.2 - Spatial object tracker on RGB camera

This example shows how to run MobileNetV2SSD on the RGB input frame, and perform spatial object tracking on persons.

Demo

Setup

Please run the following command to install the required dependencies

```

python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai

```

For additional information, please follow [installation guide](#)

This example also requires MobilenetSDD blob (mobilenet.blob file) to work - you can download it from [here](#)

Source code

Also available on [GitHub](#)

```

1  #!/usr/bin/env python3
2
3  from pathlib import Path
4  import cv2
5  import depthai as dai
6  import numpy as np
7  import time
8  import argparse
9
10 labelMap = ["background", "aeroplane", "bicycle", "bird", "boat", "bottle", "bus",
11 ↪"car", "cat", "chair", "cow",
12             "diningtable", "dog", "horse", "motorbike", "person", "pottedplant",
13 ↪"sheep", "sofa", "train", "tvmonitor"]
14
15 nnPathDefault = str((Path(__file__).parent / Path('models/mobilenet-ssd_openvino_2021.
16 ↪2_6shave.blob')).resolve().absolute())
17 parser = argparse.ArgumentParser()
18 parser.add_argument('nnPath', nargs='?', help="Path to mobilenet detection network_
19 ↪blob", default=nnPathDefault)

```

(continues on next page)

(continued from previous page)

```

16 parser.add_argument('-ff', '--full_frame', action="store_true", help="Perform_
    ↳tracking on full RGB frame", default=False)
17
18 args = parser.parse_args()
19
20
21 fullFrameTracking = args.full_frame
22
23 # Start defining a pipeline
24 pipeline = dai.Pipeline()
25
26 colorCam = pipeline.createColorCamera()
27 spatialDetectionNetwork = pipeline.createMobileNetSpatialDetectionNetwork()
28 monoLeft = pipeline.createMonoCamera()
29 monoRight = pipeline.createMonoCamera()
30 stereo = pipeline.createStereoDepth()
31 objectTracker = pipeline.createObjectTracker()
32
33 xoutRgb = pipeline.createXLinkOut()
34 trackerOut = pipeline.createXLinkOut()
35
36 xoutRgb.setStreamName("preview")
37 trackerOut.setStreamName("tracklets")
38
39 colorCam.setPreviewSize(300, 300)
40 colorCam.setResolution(dai.ColorCameraProperties.SensorResolution.THE_1080_P)
41 colorCam.setInterleaved(False)
42 colorCam.setColorOrder(dai.ColorCameraProperties.ColorOrder.BGR)
43
44 monoLeft.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
45 monoLeft.setBoardSocket(dai.CameraBoardSocket.LEFT)
46 monoRight.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
47 monoRight.setBoardSocket(dai.CameraBoardSocket.RIGHT)
48
49 # setting node configs
50 stereo.setOutputDepth(True)
51 stereo.setConfidenceThreshold(255)
52
53 spatialDetectionNetwork.setBlobPath(args.nnPath)
54 spatialDetectionNetwork.setConfidenceThreshold(0.5)
55 spatialDetectionNetwork.input.setBlocking(False)
56 spatialDetectionNetwork.setBoundingBoxScaleFactor(0.5)
57 spatialDetectionNetwork.setDepthLowerThreshold(100)
58 spatialDetectionNetwork.setDepthUpperThreshold(5000)
59
60 # Create outputs
61
62 monoLeft.out.link(stereo.left)
63 monoRight.out.link(stereo.right)
64
65 # Link plugins CAM . NN . XLINK
66 colorCam.preview.link(spatialDetectionNetwork.input)
67 objectTracker.passthroughTrackerFrame.link(xoutRgb.input)
68
69
70 objectTracker.setDetectionLabelsToTrack([15]) # track only person
71 # possible tracking types: ZERO_TERM_COLOR_HISTOGRAM, ZERO_TERM_IMAGELESS

```

(continues on next page)

(continued from previous page)

```

72 objectTracker.setTrackerType(dai.TrackerType.ZERO_TERM_COLOR_HISTOGRAM)
73 # take the smallest ID when new object is tracked, possible options: SMALLEST_ID, ↵
74   ↵UNIQUE_ID
75 objectTracker.setTrackerIdAssignmentPolicy(dai.TrackerIdAssignmentPolicy.SMALLEST_ID)
76
77 objectTracker.out.link(trackerOut.input)
78 if fullFrameTracking:
79     colorCam.setPreviewKeepAspectRatio(False)
80     colorCam.video.link(objectTracker.inputTrackerFrame)
81     objectTracker.inputTrackerFrame.setBlocking(False)
82     # do not block the pipeline if it's too slow on full frame
83     objectTracker.inputTrackerFrame.setQueueSize(2)
84 else:
85     spatialDetectionNetwork.passthrough.link(objectTracker.inputTrackerFrame)
86
87 spatialDetectionNetwork.passthrough.link(objectTracker.inputDetectionFrame)
88 spatialDetectionNetwork.out.link(objectTracker.inputDetections)
89
90 stereo.depth.link(spatialDetectionNetwork.inputDepth)
91
92 # Pipeline defined, now the device is connected to
93 with dai.Device(pipeline) as device:
94
95     # Start the pipeline
96     device.startPipeline()
97
98     preview = device.getOutputQueue("preview", 4, False)
99     tracklets = device.getOutputQueue("tracklets", 4, False)
100
101     startTime = time.monotonic()
102     counter = 0
103     fps = 0
104     frame = None
105
106     while(True):
107         imgFrame = preview.get()
108         track = tracklets.get()
109
110         counter+=1
111         current_time = time.monotonic()
112         if (current_time - startTime) > 1 :
113             fps = counter / (current_time - startTime)
114             counter = 0
115             startTime = current_time
116
117         color = (255, 0, 0)
118         frame = imgFrame.getCvFrame()
119         trackletsData = track.tracklets
120         for t in trackletsData:
121             roi = t.roi.denormalize(frame.shape[1], frame.shape[0])
122             x1 = int(roi.topLeft().x)
123             y1 = int(roi.topLeft().y)
124             x2 = int(roi.bottomRight().x)
125             y2 = int(roi.bottomRight().y)
126
127             try:

```

(continues on next page)

(continued from previous page)

```

128         label = labelMap[t.label]
129     except:
130         label = t.label
131         statusMap = {dai.Tracklet.TrackingStatus.NEW : "NEW", dai.Tracklet.
↪TrackingStatus.TRACKED : "TRACKED", dai.Tracklet.TrackingStatus.LOST : "LOST"}
132         cv2.putText(frame, str(label), (x1 + 10, y1 + 20), cv2.FONT_HERSHEY_
↪TRIPLEX, 0.5, color)
133         cv2.putText(frame, f"ID: {[t.id]}", (x1 + 10, y1 + 35), cv2.FONT_HERSHEY_
↪TRIPLEX, 0.5, color)
134         cv2.putText(frame, statusMap[t.status], (x1 + 10, y1 + 50), cv2.FONT_
↪HERSHEY_TRIPLEX, 0.5, color)
135         cv2.rectangle(frame, (x1, y1), (x2, y2), color, cv2.FONT_HERSHEY_SIMPLEX)
136
137         cv2.putText(frame, f"X: {int(t.spatialCoordinates.x)} mm", (x1 + 10, y1 +
↪65), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
138         cv2.putText(frame, f"Y: {int(t.spatialCoordinates.y)} mm", (x1 + 10, y1 +
↪80), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
139         cv2.putText(frame, f"Z: {int(t.spatialCoordinates.z)} mm", (x1 + 10, y1 +
↪95), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
140
141         cv2.putText(frame, "NN fps: {:.2f}".format(fps), (2, frame.shape[0] - 4), cv2.
↪FONT_HERSHEY_TRIPLEX, 0.4, color)
142
143         cv2.imshow("tracker", frame)
144
145         if cv2.waitKey(1) == ord('q'):
146             break

```

We're always happy to help with code or other questions you might have.

3.11.35 29.3 - Object tracker on video

This example shows how to run MobileNetV2SSD on video input frame, and perform object tracking on persons.

Demo

Setup

Please run the following command to install the required dependencies

```

python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai

```

For additional information, please follow [installation guide](#)

This example also requires MobilenetSDD based person-detection blob (person-detection-0201_openvino_2021.3_7shave.blob file) to work - you can download it from [here](#)

Source code

Also available on [GitHub](#)

```

1  #!/usr/bin/env python3
2
3  from pathlib import Path
4  import cv2
5  import depthai as dai
6  import numpy as np
7  import time
8  import argparse
9
10 labelMap = ["person", ""]
11
12 nnPathDefault = str((Path(__file__).parent / Path('models/person-detection-0201_
↳ openvino_2021.3_7shave.blob')).resolve().absolute())
13 videoPathDefault = str((Path(__file__).parent / Path('models/construction_vest.mp4')).
↳ resolve().absolute())
14 parser = argparse.ArgumentParser()
15 parser.add_argument('-nnPath', help="Path to mobilenet detection network blob",
↳ default=nnPathDefault)
16 parser.add_argument('-v', '--videoPath', help="Path to video frame",
↳ default=videoPathDefault)
17
18 args = parser.parse_args()
19
20 # Start defining a pipeline
21 pipeline = dai.Pipeline()
22
23 # Create neural network input
24 xinFrame = pipeline.createXLinkIn()
25 xinFrame.setStreamName("inFrame")
26 xinFrame.setMaxDataSize(1920*1080*3)
27
28 detectionNetwork = pipeline.createMobileNetDetectionNetwork()
29 objectTracker = pipeline.createObjectTracker()
30 trackerOut = pipeline.createXLinkOut()
31
32 xlinkOut = pipeline.createXLinkOut()
33
34 xlinkOut.setStreamName("trackerFrame")
35 trackerOut.setStreamName("tracklets")
36
37 # Create a node to convert the grayscale frame into the nn-acceptable form
38 manip = pipeline.createImageManip()
39 manip.initialConfig.setResizeThumbnail(384, 384)
40 # manip.initialConfig.setResize(384, 384)
41 # manip.initialConfig.setKeepAspectRatio(False) #squash the image to not lose FOV
42 # The NN model expects BGR input. By default ImageManip output type would be same as
↳ input (gray in this case)
43 manip.initialConfig.setFrameType(dai.ImgFrame.Type.BGR888p)
44 xinFrame.out.link(manip.inputImage)
45 manip.inputImage.setBlocking(True)
46
47 manipOut = pipeline.createXLinkOut()
48 manipOut.setStreamName("manip")
49 manip.out.link(manipOut.input)

```

(continues on next page)

(continued from previous page)

```

50
51 nnOut = pipeline.createXLinkOut()
52 nnOut.setStreamName("nn")
53 detectionNetwork.out.link(nnOut.input)
54
55
56 # setting node configs
57 detectionNetwork.setBlobPath(args.nnPath)
58 detectionNetwork.setConfidenceThreshold(0.5)
59
60 manip.out.link(detectionNetwork.input)
61 detectionNetwork.input.setBlocking(True)
62 objectTracker.passthroughTrackerFrame.link(xlinkOut.input)
63
64
65 objectTracker.setDetectionLabelsToTrack([0]) # track only person
66 # possible tracking types: ZERO_TERM_COLOR_HISTOGRAM, ZERO_TERM_IMAGELESS
67 objectTracker.setTrackerType(dai.TrackerType.ZERO_TERM_COLOR_HISTOGRAM)
68 # take the smallest ID when new object is tracked, possible options: SMALLEST_ID, ↵
69 ↵UNIQUE_ID
70 objectTracker.setTrackerIdAssignmentPolicy(dai.TrackerIdAssignmentPolicy.SMALLEST_ID)
71
72 xinFrame.out.link(objectTracker.inputTrackerFrame)
73 objectTracker.inputTrackerFrame.setBlocking(True)
74 detectionNetwork.passthrough.link(objectTracker.inputDetectionFrame)
75 objectTracker.inputDetectionFrame.setBlocking(True)
76 detectionNetwork.out.link(objectTracker.inputDetections)
77 objectTracker.inputDetections.setBlocking(True)
78 objectTracker.out.link(trackerOut.input)
79
80 # Pipeline defined, now the device is connected to
81 with dai.Device(pipeline) as device:
82
83     # Start the pipeline
84     device.startPipeline()
85
86     qIn = device.getInputQueue(name="inFrame")
87     trackerFrameQ = device.getOutputQueue("trackerFrame", 4)
88     tracklets = device.getOutputQueue("tracklets", 4)
89     qManip = device.getOutputQueue("manip", maxSize=4)
90     qDet = device.getOutputQueue("nn", maxSize=4)
91
92     startTime = time.monotonic()
93     counter = 0
94     fps = 0
95     detections = []
96     frame = None
97
98     def to_planar(arr: np.ndarray, shape: tuple) -> np.ndarray:
99         return cv2.resize(arr, shape).transpose(2, 0, 1).flatten()
100
101     # nn data, being the bounding box locations, are in <0..1> range - they need to ↵
102     ↵be normalized with frame width/height
103     def frameNorm(frame, bbox):
104         normVals = np.full(len(bbox), frame.shape[0])
105         normVals[::2] = frame.shape[1]

```

(continues on next page)

(continued from previous page)

```

105     return (np.clip(np.array(bbox), 0, 1) * normVals).astype(int)
106
107     def displayFrame(name, frame):
108         for detection in detections:
109             bbox = frameNorm(frame, (detection.xmin, detection.ymin, detection.xmax,
110 ↪ detection.ymax))
111             cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (255, 0, 0),
112 ↪ 2)
113             cv2.putText(frame, labelMap[detection.label], (bbox[0] + 10, bbox[1] +
114 ↪ 20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
115             cv2.putText(frame, f"{int(detection.confidence * 100)}%", (bbox[0] + 10,
116 ↪ bbox[1] + 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
117             cv2.imshow(name, frame)
118
119     cap = cv2.VideoCapture(args.videoPath)
120     baseTs = time.monotonic()
121     simulatedFps = 30
122     inputFrameShape = (1280, 720)
123
124     while cap.isOpened():
125         read_correctly, frame = cap.read()
126         if not read_correctly:
127             break
128
129         img = dai.ImgFrame()
130         img.setType(dai.ImgFrame.Type.BGR888p)
131         img.setData(to_planar(frame, inputFrameShape))
132         img.setTimestamp(baseTs)
133         baseTs += 1/simulatedFps
134
135         img.setWidth(inputFrameShape[0])
136         img.setHeight(inputFrameShape[1])
137         qIn.send(img)
138
139         trackFrame = trackerFrameQ.tryGet()
140         if trackFrame is None:
141             continue
142
143         track = tracklets.get()
144         manip = qManip.get()
145         inDet = qDet.get()
146
147         counter+=1
148         current_time = time.monotonic()
149         if (current_time - startTime) > 1 :
150             fps = counter / (current_time - startTime)
151             counter = 0
152             startTime = current_time
153
154         detections = inDet.detections
155         manipFrame = manip.getCvFrame()
156         displayFrame("nn", manipFrame)
157
158         color = (255, 0, 0)
159         trackerFrame = trackFrame.getCvFrame()
160         trackletsData = track.tracklets
161         for t in trackletsData:

```

(continues on next page)

(continued from previous page)

```

158         roi = t.roi.denormalize(trackerFrame.shape[1], trackerFrame.shape[0])
159         x1 = int(roi.topLeft().x)
160         y1 = int(roi.topLeft().y)
161         x2 = int(roi.bottomRight().x)
162         y2 = int(roi.bottomRight().y)
163
164         try:
165             label = labelMap[t.label]
166         except:
167             label = t.label
168
169         statusMap = {dai.Tracklet.TrackingStatus.NEW : "NEW", dai.Tracklet.
↪TrackingStatus.TRACKED : "TRACKED", dai.Tracklet.TrackingStatus.LOST : "LOST"}
170         cv2.putText(trackerFrame, str(label), (x1 + 10, y1 + 20), cv2.FONT_
↪HERSHEY_TRIPLEX, 0.5, color)
171         cv2.putText(trackerFrame, f"ID: {[t.id]}", (x1 + 10, y1 + 35), cv2.FONT_
↪HERSHEY_TRIPLEX, 0.5, color)
172         cv2.putText(trackerFrame, statusMap[t.status], (x1 + 10, y1 + 50), cv2.
↪FONT_HERSHEY_TRIPLEX, 0.5, color)
173         cv2.rectangle(trackerFrame, (x1, y1), (x2, y2), color, cv2.FONT_HERSHEY_
↪SIMPLEX)
174
175         cv2.putText(trackerFrame, "Fps: {:.2f}".format(fps), (2, trackerFrame.
↪shape[0] - 4), cv2.FONT_HERSHEY_TRIPLEX, 0.4, color)
176
177         cv2.imshow("tracker", trackerFrame)
178
179         if cv2.waitKey(1) == ord('q'):
180             break

```

We're always happy to help with code or other questions you might have.

3.11.36 30 - Stereo Depth from host

This example shows depth map from host using stereo images. There are 3 depth modes which you can select inside the code: left-right check, extended (for closer distance), subpixel (for longer distance). Otherwise a median with kernel_7x7 is activated.

Setup

Please run the following command to install the required dependencies

```

python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai

```

For additional information, please follow [installation guide](#)

This example also requires dataset folder - you can download it from [here](#)

Source code

Also available on [GitHub](#)

```

1  #!/usr/bin/env python3
2
3  import cv2
4  import numpy as np
5  import depthai as dai
6  from time import sleep
7  import datetime
8  import argparse
9  from pathlib import Path
10
11 datasetDefault = str((Path(__file__).parent / Path('models/dataset')).resolve().
    ↳absolute())
12 parser = argparse.ArgumentParser()
13 parser.add_argument('-dataset', nargs='?', help="Path to recorded frames",
    ↳default=datasetDefault)
14 args = parser.parse_args()
15
16 if not Path(datasetDefault).exists():
17     import sys
18     raise FileNotFoundError(f'Required file/s not found, please run "{sys.executable}
    ↳install_requirements.py"')
19
20
21 # StereoDepth config options.
22 out_depth = False # Disparity by default
23 out_rectified = True # Output and display rectified streams
24 lrcheck = True # Better handling for occlusions
25 extended = False # Closer-in minimum depth, disparity range is doubled
26 subpixel = True # Better accuracy for longer distance, fractional disparity 32-
    ↳levels
27 median = dai.StereoDepthProperties.MedianFilter.KERNEL_7x7
28
29 # Sanitize some incompatible options
30 if lrcheck or extended or subpixel:
31     median = dai.StereoDepthProperties.MedianFilter.MEDIAN_OFF
32
33 print("StereoDepth config options: ")
34 print("Left-Right check: ", lrcheck)
35 print("Extended disparity: ", extended)
36 print("Subpixel: ", subpixel)
37 print("Median filtering: ", median)
38
39 right_intrinsic = [[860.0, 0.0, 640.0], [0.0, 860.0, 360.0], [0.0, 0.0, 1.0]]
40
41
42 def create_stereo_depth_pipeline():
43     print("Creating Stereo Depth pipeline: ", end='')
44
45     print("XLINK IN -> STEREO -> XLINK OUT")
46     pipeline = dai.Pipeline()
47
48     camLeft = pipeline.createXLinkIn()
49     camRight = pipeline.createXLinkIn()
50     stereo = pipeline.createStereoDepth()

```

(continues on next page)

(continued from previous page)

```

51     xoutLeft = pipeline.createXLinkOut()
52     xoutRight = pipeline.createXLinkOut()
53     xoutDepth = pipeline.createXLinkOut()
54     xoutDisparity = pipeline.createXLinkOut()
55     xoutRectifLeft = pipeline.createXLinkOut()
56     xoutRectifRight = pipeline.createXLinkOut()
57
58     camLeft.setStreamName('in_left')
59     camRight.setStreamName('in_right')
60
61     stereo.setOutputDepth(out_depth)
62     stereo.setOutputRectified(out_rectified)
63     stereo.setConfidenceThreshold(200)
64     stereo.setRectifyEdgeFillColor(0) # Black, to better see the cutout
65     stereo.setMedianFilter(median) # KERNEL_7x7 default
66     stereo.setLeftRightCheck(lrcheck)
67     stereo.setExtendedDisparity(extended)
68     stereo.setSubpixel(subpixel)
69
70     stereo.setEmptyCalibration() # Set if the input frames are already rectified
71     stereo.setInputResolution(1280, 720)
72
73     xoutLeft.setStreamName('left')
74     xoutRight.setStreamName('right')
75     xoutDepth.setStreamName('depth')
76     xoutDisparity.setStreamName('disparity')
77     xoutRectifLeft.setStreamName('rectified_left')
78     xoutRectifRight.setStreamName('rectified_right')
79
80     camLeft.out.link(stereo.left)
81     camRight.out.link(stereo.right)
82     stereo.syncedLeft.link(xoutLeft.input)
83     stereo.syncedRight.link(xoutRight.input)
84     stereo.depth.link(xoutDepth.input)
85     stereo.disparity.link(xoutDisparity.input)
86     stereo.rectifiedLeft.link(xoutRectifLeft.input)
87     stereo.rectifiedRight.link(xoutRectifRight.input)
88
89     streams = ['left', 'right']
90     if out_rectified:
91         streams.extend(['rectified_left', 'rectified_right'])
92     streams.extend(['disparity', 'depth'])
93
94     return pipeline, streams
95
96
97 def convert_to_cv2_frame(name, image):
98     baseline = 75 #mm
99     focal = right_intrinsic[0][0]
100     max_disp = 96
101     disp_type = np.uint8
102     disp_levels = 1
103     if (extended):
104         max_disp *= 2
105     if (subpixel):
106         max_disp *= 32
107         disp_type = np.uint16

```

(continues on next page)

(continued from previous page)

```

108     disp_levels = 32
109
110     data, w, h = image.getData(), image.getWidth(), image.getHeight()
111     if name == 'depth':
112         # this contains FP16 with (lrccheck or extended or subpixel)
113         frame = np.array(data).astype(np.uint8).view(np.uint16).reshape((h, w))
114     elif name == 'disparity':
115         disp = np.array(data).astype(np.uint8).view(np.uint16).reshape((h, w))
116
117         # Compute depth from disparity
118         with np.errstate(divide='ignore'):
119             depth = (disp_levels * baseline * focal / disp).astype(np.uint16)
120
121         if 1: # Optionally, extend disparity range to better visualize it
122             frame = (disp * 255. / max_disp).astype(np.uint8)
123
124         if 1: # Optionally, apply a color map
125             frame = cv2.applyColorMap(frame, cv2.COLORMAP_HOT)
126
127     else: # mono streams / single channel
128         frame = np.array(data).reshape((h, w)).astype(np.uint8)
129         if name.startswith('rectified_'):
130             frame = cv2.flip(frame, 1)
131         if name == 'rectified_right':
132             last_rectif_right = frame
133     return frame
134
135
136 pipeline, streams = create_stereo_depth_pipeline()
137
138 print("Creating DepthAI device")
139 with dai.Device(pipeline) as device:
140     print("Starting pipeline")
141     device.startPipeline()
142
143     inStreams = ['in_right', 'in_left']
144     inStreamsCameraID = [dai.CameraBoardSocket.RIGHT, dai.CameraBoardSocket.LEFT]
145     in_q_list = []
146     for s in inStreams:
147         q = device.getInputQueue(s)
148         in_q_list.append(q)
149
150     # Create a receive queue for each stream
151     q_list = []
152     for s in streams:
153         q = device.getOutputQueue(s, 8, blocking=False)
154         q_list.append(q)
155
156     # Need to set a timestamp for input frames, for the sync stage in Stereo node
157     timestamp_ms = 0
158     index = 0
159     while True:
160         # Handle input streams, if any
161         if in_q_list:
162             dataset_size = 2 # Number of image pairs
163             frame_interval_ms = 500
164             for i, q in enumerate(in_q_list):

```

(continues on next page)

(continued from previous page)

```

165         path = args.dataset + '/' + str(index) + '/' + q.getName() + '.png'
166         data = cv2.imread(path, cv2.IMREAD_GRAYSCALE).reshape(720*1280)
167         tstamp = datetime.timedelta(seconds = timestamp_ms // 1000,
168                                     milliseconds = timestamp_ms % 1000)
169         img = dai.ImgFrame()
170         img.setData(data)
171         img.setTimestamp(tstamp)
172         img.setInstanceNum(inStreamsCameraID[i])
173         img.setType(dai.ImgFrame.Type.RAW8)
174         img.setWidth(1280)
175         img.setHeight(720)
176         q.send(img)
177         if timestamp_ms == 0: # Send twice for first iteration
178             q.send(img)
179         print("Sent frame: {:25s}".format(path), 'timestamp_ms:', timestamp_
180 ms)
181         timestamp_ms += frame_interval_ms
182         index = (index + 1) % dataset_size
183         sleep(frame_interval_ms / 1000)
184     # Handle output streams
185     for q in q_list:
186         if q.getName() in ['left', 'right', 'depth']: continue
187         frame = convert_to_cv2_frame(q.getName(), q.get())
188         cv2.imshow(q.getName(), frame)
189     if cv2.waitKey(1) == ord('q'):
190         break

```

We're always happy to help with code or other questions you might have.

Code samples are used for automated testing. They are also a great starting point for the gen2 API.

List of code samples

- *01 - RGB Preview* - Displays a small preview of the RGB camera
- *02 - Mono Preview* - Displays right/left mono cameras
- *03 - Depth Preview* - Displays colorized stereo disparity
- *04 - RGB Encoding* - Encodes RGB (1080P, 30FPS) into .h265 and saves it on the host
- *05 - RGB & Mono Encoding* - Encodes RGB (1080P, 30FPS) and both mono streams (720P, 30FPS) into .h265/.h264 and saves them on the host
- *06 - RGB Full Resolution Saver* - Saves full resolution RGB images (4k) on the host (.jpeg)
- *07 - Mono Full Resolution Saver* - Saves mono (720P) images to the host (.png)
- *08 - RGB & MobilenetSSD* - Runs MobileNetSSD on RGB frames and displays detections on the frame
- *09 - Mono & MobilenetSSD* - Runs MobileNetSSD on mono frames and displays detections on the frame
- *10 - Mono & MobilenetSSD & Depth* - Runs MobileNetSSD on mono frames and displays detections on mono/disparity frames
- *11 - RGB & Encoding & Mono & MobilenetSSD* - Runs MobileNetSSD on mono frames and displays detections on the frame + encodes RGB to .h265
- *12 - RGB Encoding & Mono with MobilenetSSD & Depth* - A combination of **04** and **10** code samples

- *13 - Encoding Max Limit* - Encodes RGB (4k 25FPS) and both mono streams (720P, 25FPS) into `.h265/.h264` and saves them on the host
- *14.1 - Color Camera Control* - Demonstrates how to control the color camera (crop, focus, exposure, sensitivity) from the host
- *14.2 - Mono Camera Control* - Demonstrates how to control the mono camera (crop, exposure, sensitivity) from the host
- *14.3 - Depth Crop Control* - Demonstrates how to control cropping of depth frames from the host
- *15 - 4K RGB MobileNetSSD* - Runs MobileNetSSD on RGB frames and displays detections on both preview and 4k frames
- *16 - Device Queue Event* - Demonstrates how to use device queue events
- *17 - Video & MobilenetSSD* - Runs MobileNetSSD on the video from the host
- *18 - RGB Encoding with MobilenetSSD* - Runs MobileNetSSD on RGB frames and encodes FULL-HD RGB into `.h265` and saves it on the host
- *19 - Mono Camera Control* - Demonstrates how to control the mono camera (exposure, sensitivity) from the host
- *20 - Color Rotate Warp* - Demonstrates how to rotate, mirror, flip or perform perspective transform on a frame
- *22.1 - RGB & TinyYoloV3 decoding on device* - Runs YoloV3 on RGB frames and displays detections on the frame
- *22.2 - RGB & TinyYoloV4 decoding on device* - Runs YoloV4 on RGB frames and displays detections on the frame
- *23 - Auto Exposure on ROI* - Demonstrates how to use auto exposure based on the selected ROI
- *24 - OpenCV support* - Demonstrates how to retrieve an image frame as an OpenCV frame
- *25 - System information* - Displays device system information (memory/cpu usage, temperature)
- *26.1 - RGB & MobilenetSSD with spatial data* - Displays RGB frames with MobileNet detections and spatial coordinates on them
- *26.2 - Mono & MobilenetSSD with spatial data* - Displays mono frames with MobileNet detections and spatial coordinates on them
- *26.3 - RGB & TinyYolo with spatial data* - Displays RGB frames with Yolo detections and spatial coordinates on them
- *27 - Spatial location calculator* - Demonstrates how to use the spatial location calculator
- *28 - Camera video high resolution* - Demonstrates how to use the video output of the color camera
- *29.1 - Object tracker on RGB camera* - Performs object tracking from the color camera
- *29.2 - Spatial object tracker on RGB camera* - Performs object tracking and also provides spatial coordinates
- *29.3 - Object tracker on video* - Performs object tracking from the video
- *30 - Stereo Depth from host* - Generates stereo depth frame from a set of mono images from the host

3.12 Python API Reference

Classes:

<i>ADatatype</i>	Abstract message
<i>Asset</i>	Asset is identified with string key and can store arbitrary binary data
<i>AssetManager</i>	AssetManager can store assets and serialize
<i>Buffer</i>	Base message - buffer of binary data
<i>CameraBoardSocket</i>	Which Camera socket to use.
<i>CameraControl</i>	CameraControl message Specifies various camera control commands like:
<i>CameraImageOrientation</i>	Camera sensor image orientation / pixel readout.
<i>ChipTemperature</i>	Chip temperature information.
<i>ColorCamera</i>	ColorCamera node.
<i>ColorCameraProperties</i>	Specify ColorCamera options such as camera ID, ...
<i>CpuUsage</i>	CpuUsage structure
<i>DataInputQueue</i>	Access to send messages through XLink stream
<i>DataOutputQueue</i>	Access to receive messages coming from XLink stream
<i>DetectionNetwork</i>	DetectionNetwork.
<i>DetectionNetworkProperties</i>	Properties for DetectionNetwork
<i>Device</i>	Represents the DepthAI device with the methods to interact with it.
<i>DeviceBootloader</i>	Represents the DepthAI bootloader with the methods to interact with it.
<i>DeviceDesc</i>	
<i>DeviceInfo</i>	
<i>GlobalProperties</i>	Specify properties which apply for whole pipeline
<i>ImageManip</i>	ImageManip node.
<i>ImageManipConfig</i>	ImageManipConfig message.
<i>ImgDetection</i>	
<i>ImgDetections</i>	ImgDetections message.
<i>ImgFrame</i>	ImgFrame message.
<i>LogLevel</i>	Members:
<i>MemoryInfo</i>	MemoryInfo structure
<i>MobileNetDetectionNetwork</i>	MobileNetDetectionNetwork node.
<i>MobileNetSpatialDetectionNetwork</i>	MobileNetSpatialDetectionNetwork.
<i>MonoCamera</i>	MonoCamera node.
<i>MonoCameraProperties</i>	Specify MonoCamera options such as camera ID, ...
<i>NNData</i>	NNData message.
<i>NeuralNetwork</i>	NeuralNetwork node.
<i>NeuralNetworkProperties</i>	Specify NeuralNetwork options such as blob path, ...
<i>Node</i>	Abstract Node
<i>ObjectTracker</i>	ObjectTracker node.
<i>ObjectTrackerProperties</i>	Properties for ObjectTracker
<i>OpenVINO</i>	Support for basic OpenVINO related actions like version identification of neural network blobs,...
<i>Pipeline</i>	Represents the pipeline, set of nodes and connections between them
<i>Point2f</i>	Point2f structure
<i>Point3f</i>	Point3f structure

continues on next page

Table 1 – continued from previous page

<i>RawBuffer</i>	
<i>RawCameraControl</i>	
<i>RawImageManipConfig</i>	
<i>RawImgDetections</i>	
<i>RawImgFrame</i>	
<i>RawNNData</i>	
<i>RawSpatialImgDetections</i>	
<i>RawSystemInformation</i>	System information of device
<i>RawTracklets</i>	
<i>Rect</i>	Rect structure
<i>RotatedRect</i>	
<i>SPIOut</i>	SPIOut node.
<i>Size2f</i>	
<i>SpatialDetectionNetwork</i>	SpatialDetectionNetwork node.
<i>SpatialDetectionNetworkProperties</i>	Properties for SpatialDetectionNetwork
<i>SpatialImgDetection</i>	Spatial image detection structure
<i>SpatialImgDetections</i>	SpatialImgDetections message.
<i>SpatialLocationCalculator</i>	SpatialLocationCalculator node.
<i>SpatialLocationCalculatorConfig</i>	SpatialLocationCalculatorConfig message.
<i>SpatialLocationCalculatorConfigData</i>	
<i>SpatialLocationCalculatorConfigThresholds</i>	Spatial location configuration thresholds structure
<i>SpatialLocationCalculatorData</i>	SpatialLocationCalculatorData message.
<i>SpatialLocationCalculatorProperties</i>	Specify SpatialLocationCalculator options
<i>SpatialLocations</i>	Spatial location information structure
<i>StereoDepth</i>	StereoDepth node.
<i>StereoDepthProperties</i>	Specify StereoDepth options
<i>SystemInformation</i>	SystemInformation message.
<i>SystemLogger</i>	SystemLogger node.
<i>SystemLoggerProperties</i>	
<i>TensorInfo</i>	
<i>Timestamp</i>	
<i>TrackerIdAssignmentPolicy</i>	Members:
<i>TrackerType</i>	Members:
<i>Tracklet</i>	Tracklet structure
<i>Tracklets</i>	Tracklets message.
<i>VideoEncoder</i>	VideoEncoder node.
<i>VideoEncoderProperties</i>	Specify VideoEncoder options such as profile, bitrate, ...
<i>XLinkConnection</i>	
<i>XLinkDeviceState</i>	Members:
<i>XLinkIn</i>	XLinkIn node.
<i>XLinkOut</i>	XLinkOut node.
<i>XLinkPlatform</i>	Members:
<i>XLinkProtocol</i>	Members:
<i>YoloDetectionNetwork</i>	YoloDetectionNetwork node.
<i>YoloSpatialDetectionNetwork</i>	YoloSpatialDetectionNetwork.

class depthai.**ADatatype**

Bases: pybind11_builtins.pybind11_object

Abstract message

Methods:

<code>__init__(*args, **kwargs)</code>	Initialize self.
<code>getRaw(self)</code>	

`__init__ (*args, **kwargs)`
Initialize self. See help(type(self)) for accurate signature.

`getRaw (self: depthai.ADatatype) → depthai.RawBuffer`

class depthai.Asset

Bases: pybind11_builtins.pybind11_object

Asset is identified with string key and can store arbitrary binary data

Methods:

<code>__init__(*args, **kwargs)</code>	Overloaded function.
--	----------------------

Attributes:

<code>alignment</code>
<code>data</code>
<code>key</code>

`__init__ (*args, **kwargs)`
Overloaded function.

1. `__init__(self: depthai.Asset) -> None`
2. `__init__(self: depthai.Asset, arg0: str) -> None`

property alignment

property data

property key

class depthai.AssetManager

Bases: pybind11_builtins.pybind11_object

AssetManager can store assets and serialize

Methods:

<code>__init__(self)</code>	
<code>add(*args, **kwargs)</code>	Overloaded function.
<code>addExisting(self, assets)</code>	Adds all assets in an array to the AssetManager
<code>get(*args, **kwargs)</code>	Overloaded function.
<code>getAll(*args, **kwargs)</code>	Overloaded function.
<code>remove(self, key)</code>	Removes asset with key
<code>set(self, key, asset)</code>	Adds or overwrites existing asset with a specified key.
<code>size(self)</code>	

returns Number of asset stored in the AssetManager

__init__ (*self*: depthai.AssetManager) → None

add (*args, **kwargs)

Overloaded function.

1. add(*self*: depthai.AssetManager, asset: depthai.Asset) -> None

Adds an asset object to AssetManager.

Parameter asset: Asset to add

2. add(*self*: depthai.AssetManager, key: str, asset: depthai.Asset) -> None

Adds an asset object to AssetManager with a specified key. Key value will be assigned to an Asset as well

If asset with key already exists, the function throws an error

Parameter key: Key under which the asset should be stored

Parameter asset: Asset to store

addExisting (*self*: depthai.AssetManager, assets: List[depthai.Asset]) → None

Adds all assets in an array to the AssetManager

Parameter assets: Vector of assets to add

get (*args, **kwargs)

Overloaded function.

1. get(*self*: depthai.AssetManager, key: str) -> depthai.Asset

Returns Asset assigned to the specified key or throws an error otherwise

2. get(*self*: depthai.AssetManager, key: str) -> depthai.Asset

Returns Asset assigned to the specified key or throws an error otherwise

getAll (*args, **kwargs)

Overloaded function.

1. getAll(*self*: depthai.AssetManager) -> List[depthai.Asset]

Returns All asset stored in the AssetManager

2. getAll(*self*: depthai.AssetManager) -> List[depthai.Asset]

Returns All asset stored in the AssetManager

remove (*self*: depthai.AssetManager, key: str) → None

Removes asset with key

Parameter key: Key of asset to remove

set (*self*: depthai.AssetManager, key: str, asset: depthai.Asset) → None

Adds or overwrites existing asset with a specified key.

Parameter key: Key under which the asset should be stored

Parameter asset: Asset to store

size (*self*: depthai.AssetManager) → int

Returns Number of asset stored in the AssetManager

class depthai.**Buffer**

Bases: *depthai.ADatatype*

Base message - buffer of binary data

Methods:

<code>__init__(self)</code>	Creates Buffer message
<code>getData(self)</code>	returns Reference to internal buffer
<code>setData(*args, **kwargs)</code>	Overloaded function.

`__init__(self: depthai.Buffer) → None`

Creates Buffer message

`getData(self: object) → numpy.ndarray[numpy.uint8]`

Returns Reference to internal buffer

`setData(*args, **kwargs)`

Overloaded function.

1. setData(self: depthai.Buffer, arg0: List[int]) -> None

Parameter data: Copies data to internal buffer

2. setData(self: depthai.Buffer, arg0: numpy.ndarray[numpy.uint8]) -> None

Parameter data: Copies data to internal buffer

class depthai.**CameraBoardSocket**

Bases: *pybind11_builtins.pybind11_object*

Which Camera socket to use.

AUTO denotes that the decision will be made by device

Members:

AUTO

RGB

LEFT

RIGHT

Attributes:

<i>AUTO</i>
<i>LEFT</i>
<i>RGB</i>
<i>RIGHT</i>
<i>name</i>
<i>value</i>

Methods:

```
__init__(self, value)
```

```
AUTO = <CameraBoardSocket.AUTO: -1>
```

```
LEFT = <CameraBoardSocket.LEFT: 1>
```

```
RGB = <CameraBoardSocket.RGB: 0>
```

```
RIGHT = <CameraBoardSocket.RIGHT: 2>
```

```
__init__(self: depthai.CameraBoardSocket, value: int) → None
```

property name

property value

```
class depthai.CameraControl
```

Bases: *depthai.Buffer*

CameraControl message Specifies various camera control commands like:

- Still capture
- Auto focus
- Anti banding
- Auto white balance
- Scene
- Effect
- ...

Classes:

<i>AntiBandingMode</i>	Members:
<i>AutoFocusMode</i>	Members:
<i>AutoWhiteBalanceMode</i>	Members:
<i>EffectMode</i>	Members:
<i>SceneMode</i>	Members:

Methods:

<code>__init__(self)</code>	Construct CameraControl message
<code>getCaptureStill(self)</code>	Check whether command to capture a still is set
<code>setAntiBandingMode(self, mode)</code>	Set a command to specify auto banding mode
<code>setAutoExposureCompensation(self, compensation)</code>	Set a command to specify auto exposure compensation
<code>setAutoExposureEnable(self)</code>	Set a command to enable auto exposure
<code>setAutoExposureLock(self, lock)</code>	Set a command to specify lock auto exposure
<code>setAutoExposureRegion(self, startX, startY, ...)</code>	Set a command to specify auto exposure region in pixels
<code>setAutoFocusMode(self, mode)</code>	Set a command to specify autofocus mode
<code>setAutoFocusRegion(self, startX, startY, ...)</code>	Set a command to specify focus region in pixels
<code>setAutoFocusTrigger(self)</code>	Set a command to trigger autofocus
<code>setAutoWhiteBalanceLock(self, lock)</code>	Set a command to specify auto white balance lock
<code>setAutoWhiteBalanceMode(self, mode)</code>	Set a command to specify auto white balance mode

continues on next page

Table 10 – continued from previous page

<i>setBrightness</i> (self, value)	Set a command to specify auto white balance lock
<i>setCaptureStill</i> (self, capture)	Set a command to capture a still image
<i>setChromaDenoise</i> (self, value)	Set a command to specify chroma denoise value
<i>setContrast</i> (self, value)	Set a command to specify auto white balance lock
<i>setEffectMode</i> (self, mode)	Set a command to specify effect mode
<i>setLumaDenoise</i> (self, value)	Set a command to specify luma denoise value
<i>setManualExposure</i> (self, exposureTimeUs, ...)	Set a command to manually specify exposure
<i>setManualFocus</i> (self, lensPosition)	Set a command to specify manual focus position
<i>setNoiseReductionStrength</i> (self, value)	Set a command to specify noise reduction strength
<i>setSaturation</i> (self, value)	Set a command to specify saturation value
<i>setSceneMode</i> (self, mode)	Set a command to specify scene mode
<i>setSharpness</i> (self, value)	Set a command to specify sharpness value
<i>setStartStreaming</i> (self)	Set a command to start streaming
<i>setStopStreaming</i> (self)	Set a command to stop streaming

class AntiBandingModeBases: `pybind11_builtins.pybind11_object`

Members:

OFF

MAINS_50_HZ

MAINS_60_HZ

AUTO

Attributes:

<i>AUTO</i>
<i>MAINS_50_HZ</i>
<i>MAINS_60_HZ</i>
<i>OFF</i>
<i>name</i>
<i>value</i>

Methods:

<i>__init__</i> (self, value)

AUTO = <AntiBandingMode.AUTO: 3>**MAINS_50_HZ** = <AntiBandingMode.MAINS_50_HZ: 1>**MAINS_60_HZ** = <AntiBandingMode.MAINS_60_HZ: 2>**OFF** = <AntiBandingMode.OFF: 0>*__init__*(self: `depthai.RawCameraControl.AntiBandingMode`, value: `int`) → None

property name

property value

class AutoFocusMode

Bases: `pybind11_builtins.pybind11_object`

Members:

OFF
AUTO
MACRO
CONTINUOUS_VIDEO
CONTINUOUS_PICTURE
EDOF

Attributes:

<i>AUTO</i>
<i>CONTINUOUS_PICTURE</i>
<i>CONTINUOUS_VIDEO</i>
<i>EDOF</i>
<i>MACRO</i>
<i>OFF</i>
<i>name</i>
<i>value</i>

Methods:

<i>__init__</i> (self, value)

```
AUTO = <AutoFocusMode.AUTO: 1>
CONTINUOUS_PICTURE = <AutoFocusMode.CONTINUOUS_PICTURE: 4>
CONTINUOUS_VIDEO = <AutoFocusMode.CONTINUOUS_VIDEO: 3>
EDOF = <AutoFocusMode.EDOF: 5>
MACRO = <AutoFocusMode.MACRO: 2>
OFF = <AutoFocusMode.OFF: 0>

__init__(self: depthai.RawCameraControl.AutoFocusMode, value: int) → None

property name
property value
```

class AutoWhiteBalanceMode

Bases: `pybind11_builtins.pybind11_object`

Members:

OFF
AUTO
INCANDESCENT
FLUORESCENT
WARM_FLUORESCENT
DAYLIGHT

CLOUDY_DAYLIGHT

TWILIGHT

SHADE

Attributes:

<i>AUTO</i>
<i>CLOUDY_DAYLIGHT</i>
<i>DAYLIGHT</i>
<i>FLUORESCENT</i>
<i>INCANDESCENT</i>
<i>OFF</i>
<i>SHADE</i>
<i>TWILIGHT</i>
<i>WARM_FLUORESCENT</i>
<i>name</i>
<i>value</i>

Methods:

<code>__init__(self, value)</code>

AUTO = <AutoWhiteBalanceMode.AUTO: 1>**CLOUDY_DAYLIGHT** = <AutoWhiteBalanceMode.CLOUDY_DAYLIGHT: 6>**DAYLIGHT** = <AutoWhiteBalanceMode.DAYLIGHT: 5>**FLUORESCENT** = <AutoWhiteBalanceMode.FLUORESCENT: 3>**INCANDESCENT** = <AutoWhiteBalanceMode.INCANDESCENT: 2>**OFF** = <AutoWhiteBalanceMode.OFF: 0>**SHADE** = <AutoWhiteBalanceMode.SHADE: 8>**TWILIGHT** = <AutoWhiteBalanceMode.TWILIGHT: 7>**WARM_FLUORESCENT** = <AutoWhiteBalanceMode.WARM_FLUORESCENT: 4>`__init__(self: depthai.RawCameraControl.AutoWhiteBalanceMode, value: int) → None`

property name

property value

class EffectModeBases: `pybind11_builtins.pybind11_object`

Members:

OFF

MONO

NEGATIVE

SOLARIZE

SEPIA

POSTERIZE
WHITEBOARD
BLACKBOARD
AQUA

Attributes:

<i>AQUA</i>
<i>BLACKBOARD</i>
<i>MONO</i>
<i>NEGATIVE</i>
<i>OFF</i>
<i>POSTERIZE</i>
<i>SEPIA</i>
<i>SOLARIZE</i>
<i>WHITEBOARD</i>
<i>name</i>
<i>value</i>

Methods:

<i>__init__</i> (self, value)

```
AQUA = <EffectMode.AQUA: 8>
BLACKBOARD = <EffectMode.BLACKBOARD: 7>
MONO = <EffectMode.MONO: 1>
NEGATIVE = <EffectMode.NEGATIVE: 2>
OFF = <EffectMode.OFF: 0>
POSTERIZE = <EffectMode.POSTERIZE: 5>
SEPIA = <EffectMode.SEPIA: 4>
SOLARIZE = <EffectMode.SOLARIZE: 3>
WHITEBOARD = <EffectMode.WHITEBOARD: 6>
```

```
__init__(self: depthai.RawCameraControl.EffectMode, value: int) → None
```

property name
property value

class SceneMode

Bases: `pybind11_builtins.pybind11_object`

Members:

UNSUPPORTED
FACE_PRIORITY
ACTION
PORTRAIT

LANDSCAPE
NIGHT
NIGHT_PORTRAIT
THEATRE
BEACH
SNOW
SUNSET
STEADYPHOTO
FIREWORKS
SPORTS
PARTY
CANDLELIGHT
BARCODE

Attributes:

<i>ACTION</i>
<i>BARCODE</i>
<i>BEACH</i>
<i>CANDLELIGHT</i>
<i>FACE_PRIORITY</i>
<i>FIREWORKS</i>
<i>LANDSCAPE</i>
<i>NIGHT</i>
<i>NIGHT_PORTRAIT</i>
<i>PARTY</i>
<i>PORTRAIT</i>
<i>SNOW</i>
<i>SPORTS</i>
<i>STEADYPHOTO</i>
<i>SUNSET</i>
<i>THEATRE</i>
<i>UNSUPPORTED</i>
<i>name</i>
<i>value</i>

Methods:

<i>__init__</i> (self, value)

ACTION = <SceneMode.ACTION: 2>
BARCODE = <SceneMode.BARCODE: 16>
BEACH = <SceneMode.BEACH: 8>
CANDLELIGHT = <SceneMode.CANDLELIGHT: 15>
FACE_PRIORITY = <SceneMode.FACE_PRIORITY: 1>

```
FIREWORKS = <SceneMode.FIREWORKS: 12>
LANDSCAPE = <SceneMode.LANDSCAPE: 4>
NIGHT = <SceneMode.NIGHT: 5>
NIGHT_PORTRAIT = <SceneMode.NIGHT_PORTRAIT: 6>
PARTY = <SceneMode.PARTY: 14>
PORTRAIT = <SceneMode.PORTRAIT: 3>
SNOW = <SceneMode.SNOW: 9>
SPORTS = <SceneMode.SPORTS: 13>
STEADYPHOTO = <SceneMode.STEADYPHOTO: 11>
SUNSET = <SceneMode.SUNSET: 10>
THEATRE = <SceneMode.THEATRE: 7>
UNSUPPORTED = <SceneMode.UNSUPPORTED: 0>
```

`__init__` (*self*: `depthai.RawCameraControl.SceneMode`, *value*: `int`) → `None`

property name

property value

`__init__` (*self*: `depthai.CameraControl`) → `None`
Construct CameraControl message

`getCaptureStill` (*self*: `depthai.CameraControl`) → `bool`
Check whether command to capture a still is set

Returns True if capture still command is set

`setAntiBandingMode` (*self*: `depthai.CameraControl`, *mode*: `depthai.RawCameraControl.AntiBandingMode`) → `None`
Set a command to specify auto banding mode

Parameter mode: Auto banding mode to use

`setAutoExposureCompensation` (*self*: `depthai.CameraControl`, *compensation*: `int`) → `None`
Set a command to specify auto exposure compenstaion

Parameter compensation: Compensation value between -128..127

`setAutoExposureEnable` (*self*: `depthai.CameraControl`) → `None`
Set a command to enable auto exposure

`setAutoExposureLock` (*self*: `depthai.CameraControl`, *lock*: `bool`) → `None`
Set a command to specify lock auto exposure

Parameter lock: Auto exposure lock mode enabled or disabled

`setAutoExposureRegion` (*self*: `depthai.CameraControl`, *startX*: `int`, *startY*: `int`, *width*: `int`, *height*: `int`) → `None`
Set a command to specify auto exposure region in pixels

Parameter startX: X coordinate of top left corner of region

Parameter startY: Y coordinate of top left corner of region

Parameter width: Region width

Parameter height: Region height

setAutoFocusMode (*self*: depthai.CameraControl, *mode*: depthai.RawCameraControl.AutoFocusMode)

→ None
Set a command to specify autofocus mode

setAutoFocusRegion (*self*: depthai.CameraControl, *startX*: int, *startY*: int, *width*: int, *height*: int)

→ None
Set a command to specify focus region in pixels

Parameter startX: X coordinate of top left corner of region

Parameter startY: Y coordinate of top left corner of region

Parameter width: Region width

Parameter height: Region height

setAutoFocusTrigger (*self*: depthai.CameraControl) → None

Set a command to trigger autofocus

setAutoWhiteBalanceLock (*self*: depthai.CameraControl, *lock*: bool) → None

Set a command to specify auto white balance lock

Parameter lock: Auto white balance lock mode enabled or disabled

setAutoWhiteBalanceMode (*self*: depthai.CameraControl, *mode*: depthai.RawCameraControl.AutoWhiteBalanceMode) → None

Set a command to specify auto white balance mode

Parameter mode: Auto white balance mode to use

setBrightness (*self*: depthai.CameraControl, *value*: int) → None

Set a command to specify auto white balance lock

Parameter lock: Auto white balance lock mode enabled or disabled

setCaptureStill (*self*: depthai.CameraControl, *capture*: bool) → None

Set a command to capture a still image

setChromaDenoise (*self*: depthai.CameraControl, *value*: int) → None

Set a command to specify chroma denoise value

Parameter value: Chroma denoise

setContrast (*self*: depthai.CameraControl, *value*: int) → None

Set a command to specify auto white balance lock

Parameter lock: Auto white balance lock mode enabled or disabled

setEffectMode (*self*: depthai.CameraControl, *mode*: depthai.RawCameraControl.EffectMode) → None

Set a command to specify effect mode

Parameter mode: Effect mode

setLumaDenoise (*self*: depthai.CameraControl, *value*: int) → None

Set a command to specify luma denoise value

Parameter value: Luma denoise

setManualExposure (*self*: depthai.CameraControl, *exposureTimeUs*: int, *sensitivityIso*: int) → None

Set a command to manually specify exposure

Parameter exposureTimeUs: Exposure time in microseconds

Parameter sensitivityIso: Sensitivity as ISO value

setManualFocus (*self*: [depthai.CameraControl](#), *lensPosition*: *int*) → [None](#)

Set a command to specify manual focus position

Parameter lensPosition: specify lens position 0..255

setNoiseReductionStrength (*self*: [depthai.CameraControl](#), *value*: *int*) → [None](#)

Set a command to specify noise reduction strength

Parameter value: Noise reduction strength

setSaturation (*self*: [depthai.CameraControl](#), *value*: *int*) → [None](#)

Set a command to specify saturation value

Parameter value: Saturation

setSceneMode (*self*: [depthai.CameraControl](#), *mode*: [depthai.RawCameraControl.SceneMode](#)) → [None](#)

Set a command to specify scene mode

Parameter mode: Scene mode

setSharpness (*self*: [depthai.CameraControl](#), *value*: *int*) → [None](#)

Set a command to specify sharpness value

Parameter value: Sharpness

setStartStreaming (*self*: [depthai.CameraControl](#)) → [None](#)

Set a command to start streaming

setStopStreaming (*self*: [depthai.CameraControl](#)) → [None](#)

Set a command to stop streaming

class [depthai.CameraImageOrientation](#)

Bases: [pybind11_builtins.pybind11_object](#)

Camera sensor image orientation / pixel readout. This exposes direct sensor settings. 90 or 270 degrees rotation is not available.

AUTO denotes that the decision will be made by device (e.g. on OAK-1/megaAI: ROTATE_180_DEG).

Members:

AUTO

NORMAL

HORIZONTAL_MIRROR

VERTICAL_FLIP

ROTATE_180_DEG

Attributes:

AUTO

HORIZONTAL_MIRROR

NORMAL

ROTATE_180_DEG

VERTICAL_FLIP

name

value

Methods:

```
__init__(self, value)
```

```
AUTO = <CameraImageOrientation.AUTO: -1>
```

```
HORIZONTAL_MIRROR = <CameraImageOrientation.HORIZONTAL_MIRROR: 1>
```

```
NORMAL = <CameraImageOrientation.NORMAL: 0>
```

```
ROTATE_180_DEG = <CameraImageOrientation.ROTATE_180_DEG: 3>
```

```
VERTICAL_FLIP = <CameraImageOrientation.VERTICAL_FLIP: 2>
```

```
__init__(self: depthai.CameraImageOrientation, value: int) → None
```

property name

property value

class `depthai.ChipTemperature`

Bases: `pybind11_builtins.pybind11_object`

Chip temperature information.

Multiple temperature measurement points and their average

Methods:

```
__init__(self)
```

Attributes:

average

css

dss

mss

upa

```
__init__(self: depthai.ChipTemperature) → None
```

property average

property css

property dss

property mss

property upa

class `depthai.ColorCamera`

Bases: `depthai.Node`

ColorCamera node. For use with color sensors.

Classes:

Properties

alias of `depthai.ColorCameraProperties`

Methods:

<code>__init__(*args, **kwargs)</code>	Initialize self.
<code>getBoardSocket(self)</code>	Retrieves which board socket to use
<code>getCamId(self)</code>	
<code>getColorOrder(self)</code>	Get color order of preview output frames.
<code>getFp16(self)</code>	Get fp16 (0..255) data of preview output frames
<code>getFps(self)</code>	Get rate at which camera should produce frames
<code>getImageOrientation(self)</code>	Get camera image orientation
<code>getInterleaved(self)</code>	Get planar or interleaved data of preview output frames
<code>getPreviewHeight(self)</code>	Get preview height
<code>getPreviewKeepAspectRatio(self)</code>	See also: setPreviewKeepAspectRatio
<code>getPreviewSize(self)</code>	Get preview size as tuple
<code>getPreviewWidth(self)</code>	Get preview width
<code>getResolution(self)</code>	Get sensor resolution
<code>getResolutionHeight(self)</code>	Get sensor resolution height
<code>getResolutionSize(self)</code>	Get sensor resolution as size
<code>getResolutionWidth(self)</code>	Get sensor resolution width
<code>getSensorCrop(self)</code>	returns Sensor top left crop coordinates
<code>getSensorCropX(self)</code>	Get sensor top left x crop coordinate
<code>getSensorCropY(self)</code>	Get sensor top left y crop coordinate
<code>getStillHeight(self)</code>	Get still height
<code>getStillSize(self)</code>	Get still size as tuple
<code>getStillWidth(self)</code>	Get still width
<code>getVideoHeight(self)</code>	Get video height
<code>getVideoSize(self)</code>	Get video size as tuple
<code>getVideoWidth(self)</code>	Get video width
<code>getWaitForConfigInput(self)</code>	See also: setWaitForConfigInput
<code>sensorCenterCrop(self)</code>	Specify sensor center crop.
<code>setBoardSocket(self, boardSocket)</code>	Specify which board socket to use
<code>setCamId(self, arg0)</code>	
<code>setColorOrder(self, colorOrder)</code>	Set color order of preview output images.
<code>setFp16(self, fp16)</code>	Set fp16 (0..255) data type of preview output frames
<code>setFps(self, fps)</code>	Set rate at which camera should produce frames
<code>setImageOrientation(self, boardSocket)</code>	Set camera image orientation
<code>setInterleaved(self, interleaved)</code>	Set planar or interleaved data of preview output frames
<code>setPreviewKeepAspectRatio(self, keep)</code>	Specifies whether preview output should preserve aspect ratio, after downscaling from video size or not.
<code>setPreviewSize(self, width, height)</code>	Set preview output size
<code>setResolution(self, resolution)</code>	Set sensor resolution
<code>setSensorCrop(self, x, y)</code>	Specifies sensor crop rectangle
<code>setStillSize(self, width, height)</code>	Set still output size
<code>setVideoSize(self, width, height)</code>	Set video output size
<code>setWaitForConfigInput(self, wait)</code>	Specify to wait until inputConfig receives a configuration message, before sending out a frame.

Attributes:

<i>initialControl</i>	Initial control options to apply to sensor
<i>inputConfig</i>	Input for ImageManipConfig message, which can modify crop paremeters in runtime
<i>inputControl</i>	Input for CameraControl message, which can modify camera parameters in runtime
<i>preview</i>	Outputs ImgFrame message that carries BGR/RGB planar/interleaved encoded frame data.
<i>still</i>	Outputs ImgFrame message that carries NV12 encoded (YUV420, UV plane interleaved) frame data.
<i>video</i>	Outputs ImgFrame message that carries NV12 encoded (YUV420, UV plane interleaved) frame data.

Propertiesalias of *depthai.ColorCameraProperties* **Classes:**

ColorOrder	For 24 bit color these can be either RGB or BGR
SensorResolution	Select the camera sensor resolution

Methods:

<code>__init__(*args, **kwargs)</code>	Initialize self.
--	------------------

Attributes:

boardSocket
colorOrder
fps
initialControl
interleaved
previewHeight
previewWidth
resolution
sensorCropX
sensorCropY
stillHeight
stillWidth
videoHeight
videoWidth

`__init__(*args, **kwargs)`
Initialize self. See `help(type(self))` for accurate signature.

getBoardSocket (*self*: *depthai.ColorCamera*) → *dai::CameraBoardSocket*
Retrieves which board socket to use

Returns Board socket to use

getCamId (*self*: *depthai.ColorCamera*) → *int*

getColorOrder (*self*: *depthai.ColorCamera*) → *dai::ColorCameraProperties::ColorOrder*
Get color order of preview output frames. RGB or BGR

getFp16 (*self*: [depthai.ColorCamera](#)) → bool

Get fp16 (0..255) data of preview output frames

getFps (*self*: [depthai.ColorCamera](#)) → float

Get rate at which camera should produce frames

Returns Rate in frames per second

getImageOrientation (*self*: [depthai.ColorCamera](#)) → [dai::CameraImageOrientation](#)

Get camera image orientation

getInterleaved (*self*: [depthai.ColorCamera](#)) → bool

Get planar or interleaved data of preview output frames

getPreviewHeight (*self*: [depthai.ColorCamera](#)) → int

Get preview height

getPreviewKeepAspectRatio (*self*: [depthai.ColorCamera](#)) → bool

See also:

[setPreviewKeepAspectRatio](#)

Returns Preview keep aspect ratio option

getPreviewSize (*self*: [depthai.ColorCamera](#)) → [Tuple\[int, int\]](#)

Get preview size as tuple

getPreviewWidth (*self*: [depthai.ColorCamera](#)) → int

Get preview width

getResolution (*self*: [depthai.ColorCamera](#)) → [dai::ColorCameraProperties::SensorResolution](#)

Get sensor resolution

getResolutionHeight (*self*: [depthai.ColorCamera](#)) → int

Get sensor resolution height

getResolutionSize (*self*: [depthai.ColorCamera](#)) → [Tuple\[int, int\]](#)

Get sensor resolution as size

getResolutionWidth (*self*: [depthai.ColorCamera](#)) → int

Get sensor resolution width

getSensorCrop (*self*: [depthai.ColorCamera](#)) → [Tuple\[float, float\]](#)

Returns Sensor top left crop coordinates

getSensorCropX (*self*: [depthai.ColorCamera](#)) → float

Get sensor top left x crop coordinate

getSensorCropY (*self*: [depthai.ColorCamera](#)) → float

Get sensor top left y crop coordinate

getStillHeight (*self*: [depthai.ColorCamera](#)) → int

Get still height

getStillSize (*self*: [depthai.ColorCamera](#)) → [Tuple\[int, int\]](#)

Get still size as tuple

getStillWidth (*self*: [depthai.ColorCamera](#)) → int

Get still width

getVideoHeight (*self*: [depthai.ColorCamera](#)) → int
Get video height

getVideoSize (*self*: [depthai.ColorCamera](#)) → Tuple[int, int]
Get video size as tuple

getVideoWidth (*self*: [depthai.ColorCamera](#)) → int
Get video width

getWaitForConfigInput (*self*: [depthai.ColorCamera](#)) → bool

See also:

[setWaitForConfigInput](#)

Returns True if wait for inputConfig message, false otherwise

property initialControl
Initial control options to apply to sensor

property inputConfig
Input for ImageManipConfig message, which can modify crop paremeters in runtime
Default queue is non-blocking with size 8

property inputControl
Input for CameraControl message, which can modify camera parameters in runtime
Default queue is blocking with size 8

property preview
Outputs ImgFrame message that carries BGR/RGB planar/interleaved encoded frame data.
Suitable for use with NeuralNetwork node

sensorCenterCrop (*self*: [depthai.ColorCamera](#)) → None
Specify sensor center crop. Resolution size / video size

setBoardSocket (*self*: [depthai.ColorCamera](#), *boardSocket*: [dai::CameraBoardSocket](#)) → None
Specify which board socket to use

Parameter boardSocket: Board socket to use

setCamId (*self*: [depthai.ColorCamera](#), *arg0*: int) → None

setColorOrder (*self*: [depthai.ColorCamera](#), *colorOrder*: [dai::ColorCameraProperties::ColorOrder](#)) → None
Set color order of preview output images. RGB or BGR

setFp16 (*self*: [depthai.ColorCamera](#), *fp16*: bool) → None
Set fp16 (0..255) data type of preview output frames

setFps (*self*: [depthai.ColorCamera](#), *fps*: float) → None
Set rate at which camera should produce frames

Parameter fps: Rate in frames per second

setImageOrientation (*self*: [depthai.ColorCamera](#), *boardSocket*: [dai::CameraImageOrientation](#)) → None
Set camera image orientation

setInterleaved (*self*: [depthai.ColorCamera](#), *interleaved*: bool) → None
Set planar or interleaved data of preview output frames

setPreviewKeepAspectRatio (*self*: `depthai.ColorCamera`, *keep*: `bool`) → `None`

Specifies whether preview output should preserve aspect ratio, after downscaling from video size or not.

Parameter keep: If true, a larger crop region will be considered to still be able to create the final image in the specified aspect ratio. Otherwise video size is resized to fit preview size

setPreviewSize (*self*: `depthai.ColorCamera`, *width*: `int`, *height*: `int`) → `None`

Set preview output size

setResolution (*self*: `depthai.ColorCamera`, *resolution*: `dai::ColorCameraProperties::SensorResolution`) → `None`

Set sensor resolution

setSensorCrop (*self*: `depthai.ColorCamera`, *x*: `float`, *y*: `float`) → `None`

Specifies sensor crop rectangle

Parameter x: Top left X coordinate

Parameter y: Top left Y coordinate

setStillSize (*self*: `depthai.ColorCamera`, *width*: `int`, *height*: `int`) → `None`

Set still output size

setVideoSize (*self*: `depthai.ColorCamera`, *width*: `int`, *height*: `int`) → `None`

Set video output size

setWaitForConfigInput (*self*: `depthai.ColorCamera`, *wait*: `bool`) → `None`

Specify to wait until inputConfig receives a configuration message, before sending out a frame.

Parameter wait: True to wait for inputConfig message, false otherwise

property still

Outputs ImgFrame message that carries NV12 encoded (YUV420, UV plane interleaved) frame data.

The message is sent only when a CameraControl message arrives to inputControl with captureStill command set.

property video

Outputs ImgFrame message that carries NV12 encoded (YUV420, UV plane interleaved) frame data.

Suitable for use with VideoEncoder node

class `depthai.ColorCameraProperties`

Bases: `pybind11_builtins.pybind11_object`

Specify ColorCamera options such as camera ID, ...

Classes:

<code>ColorOrder</code>	For 24 bit color these can be either RGB or BGR
<code>SensorResolution</code>	Select the camera sensor resolution

Methods:

<code>__init__</code> (*args, **kwargs)	Initialize self.
---	------------------

Attributes:

<code>boardSocket</code>	
<code>colorOrder</code>	

continues on next page

Table 33 – continued from previous page

<i>fps</i>
<i>initialControl</i>
<i>interleaved</i>
<i>previewHeight</i>
<i>previewWidth</i>
<i>resolution</i>
<i>sensorCropX</i>
<i>sensorCropY</i>
<i>stillHeight</i>
<i>stillWidth</i>
<i>videoHeight</i>
<i>videoWidth</i>

class ColorOrderBases: `pybind11_builtins.pybind11_object`

For 24 bit color these can be either RGB or BGR

Members:

BGR

RGB

Attributes:

<i>BGR</i>
<i>RGB</i>
<i>name</i>
<i>value</i>

Methods:

<code>__init__(self, value)</code>

BGR = <ColorOrder.BGR: 0>**RGB = <ColorOrder.RGB: 1>**`__init__(self: depthai.ColorCameraProperties.ColorOrder, value: int) → None`**property name****property value****class SensorResolution**Bases: `pybind11_builtins.pybind11_object`

Select the camera sensor resolution

Members:

THE_1080_P

THE_4_K

THE_12_MP

Attributes:

`THE_1080_P`

`THE_12_MP`

`THE_4_K`

`name`

`value`

Methods:

`__init__(self, value)`

`THE_1080_P = <SensorResolution.THE_1080_P: 0>``THE_12_MP = <SensorResolution.THE_12_MP: 2>``THE_4_K = <SensorResolution.THE_4_K: 1>``__init__(self: depthai.ColorCameraProperties.SensorResolution, value: int) → None``property name``property value``__init__(*args, **kwargs)`

Initialize self. See help(type(self)) for accurate signature.

`property boardSocket``property colorOrder``property fps``property initialControl``property interleaved``property previewHeight``property previewWidth``property resolution``property sensorCropX``property sensorCropY``property stillHeight``property stillWidth``property videoHeight``property videoWidth`**class** `depthai.CpuUsage`Bases: `pybind11_builtins.pybind11_object`

CpuUsage structure

Average usage in percent and time span of the average (since last query)

Methods:

`__init__(self)`

Attributes:

average

msTime

__init__ (*self*: depthai.CpuUsage) → None**property average****property msTime****class** depthai.DataInputQueue

Bases: pybind11_builtins.pybind11_object

Access to send messages through XLink stream

Methods:

__init__ (*args, **kwargs)	Initialize self.
<i>getBlocking</i> (self)	Gets current queue behavior when full (maxSize)
<i>getMaxSize</i> (self, arg0)	Gets queue maximum size
<i>getName</i> (self)	Gets queues name
<i>send</i> (*args, **kwargs)	Overloaded function.
<i>setBlocking</i> (self, blocking)	Sets queue behavior when full (maxSize)
<i>setMaxSize</i> (self, maxSize)	Sets queue maximum size

__init__ (*args, **kwargs)

Initialize self. See help(type(self)) for accurate signature.

getBlocking (*self*: depthai.DataInputQueue) → bool

Gets current queue behavior when full (maxSize)

Returns true if blocking, false otherwise**getMaxSize** (*self*: depthai.DataInputQueue, *arg0*: int) → int

Gets queue maximum size

Returns Maximum queue size**getName** (*self*: depthai.DataInputQueue) → str

Gets queues name

Returns Queue name**send** (*args, **kwargs)

Overloaded function.

1. **send**(self: depthai.DataInputQueue, msg: depthai.ADatatype) -> None

Adds a message to the queue, which will be picked up and sent to the device. Can either block if 'blocking' behavior is true or overwrite oldest

Parameter msg: Message to add to the queue2. **send**(self: depthai.DataInputQueue, rawMsg: depthai.RawBuffer) -> None

Adds a raw message to the queue, which will be picked up and sent to the device. Can either block if 'blocking' behavior is true or overwrite oldest

Parameter rawMsg: Message to add to the queue

setBlocking (*self*: depthai.DataInputQueue, *blocking*: *bool*) → *None*

Sets queue behavior when full (maxSize)

Parameter blocking: Specifies if block or overwrite the oldest message in the queue

setMaxSize (*self*: depthai.DataInputQueue, *maxSize*: *int*) → *None*

Sets queue maximum size

Parameter maxSize: Specifies maximum number of messages in the queue

class depthai.DataOutputQueue

Bases: pybind11_builtins.pybind11_object

Access to receive messages coming from XLink stream

Methods:

<code>__init__(*args, **kwargs)</code>	Initialize self.
<code>addCallback(*args, **kwargs)</code>	Overloaded function.
<code>get(self)</code>	Block until a message is available.
<code>getAll(self)</code>	Block until at least one message in the queue.
<code>getBlocking(self)</code>	Gets current queue behavior when full (maxSize)
<code>getMaxSize(self, arg0)</code>	Gets queue maximum size
<code>getName(self)</code>	Gets queues name
<code>has(self)</code>	Check whether front of the queue has a message (isn't empty)
<code>removeCallback(self, callbackId)</code>	Removes a callback
<code>setBlocking(self, blocking)</code>	Sets queue behavior when full (maxSize)
<code>setMaxSize(self, maxSize)</code>	Sets queue maximum size
<code>tryGet(self)</code>	Try to retrieve message from queue.
<code>tryGetAll(self)</code>	Try to retrieve all messages in the queue.

`__init__ (*args, **kwargs)`

Initialize self. See help(type(self)) for accurate signature.

addCallback (**args, **kwargs*)

Overloaded function.

1. addCallback(*self*: depthai.DataOutputQueue, *callback*: std::function<void (std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >, std::shared_ptr<dai::ADatatype>>)> -> int

Adds a callback on message received

Parameter callback: Callback function with queue name and message pointer

Returns Callback id

2. addCallback(*self*: depthai.DataOutputQueue, *callback*: std::function<void (std::shared_ptr<dai::ADatatype>>)> -> int

Adds a callback on message received

Parameter callback: Callback function with message pointer

Returns Callback id

3. addCallback(*self*: depthai.DataOutputQueue, *callback*: std::function<void ()>) -> int

Adds a callback on message received

Parameter callback: Callback function without any parameters

Returns Callback id

get (*self*: [depthai.DataOutputQueue](#)) → [depthai.ADatatype](#)

Block until a message is available.

Returns Message or nullptr if no message available

getAll (*self*: [depthai.DataOutputQueue](#)) → List[[depthai.ADatatype](#)]

Block until at least one message in the queue. Then return all messages from the queue.

Returns Vector of messages

getBlocking (*self*: [depthai.DataOutputQueue](#)) → bool

Gets current queue behavior when full (maxSize)

Returns true if blocking, false otherwise

getMaxSize (*self*: [depthai.DataOutputQueue](#), *arg0*: int) → int

Gets queue maximum size

Returns Maximum queue size

getName (*self*: [depthai.DataOutputQueue](#)) → str

Gets queues name

Returns Queue name

has (*self*: [depthai.DataOutputQueue](#)) → bool

Check whether front of the queue has a message (isn't empty)

Returns true if queue isn't empty, false otherwise

removeCallback (*self*: [depthai.DataOutputQueue](#), *callbackId*: int) → bool

Removes a callback

Parameter callbackId: Id of callback to be removed

Returns true if callback was removed, false otherwise

setBlocking (*self*: [depthai.DataOutputQueue](#), *blocking*: bool) → None

Sets queue behavior when full (maxSize)

Parameter blocking: Specifies if block or overwrite the oldest message in the queue

setMaxSize (*self*: [depthai.DataOutputQueue](#), *maxSize*: int) → None

Sets queue maximum size

Parameter maxSize: Specifies maximum number of messages in the queue

tryGet (*self*: [depthai.DataOutputQueue](#)) → [depthai.ADatatype](#)

Try to retrieve message from queue. If no message available, return immediately with nullptr

Returns Message or nullptr if no message available

tryGetAll (*self*: [depthai.DataOutputQueue](#)) → List[[depthai.ADatatype](#)]

Try to retrieve all messages in the queue.

Returns Vector of messages

class `depthai.DetectionNetwork`

Bases: `depthai.NeuralNetwork`

DetectionNetwork. Base for different network specializations

Classes:

<i>Properties</i>	alias of <i>depthai.DetectionNetworkProperties</i>
-------------------	--

Methods:

<code>__init__(*args, **kwargs)</code>	Initialize self.
<code>setConfidenceThreshold(self, thresh)</code>	Specifies confidence threshold at which to filter the rest of the detections.

Attributes:

<i>input</i>	Input message with data to be inferred upon Default queue is blocking with size 5
<i>out</i>	Outputs ImgDetections message that carries parsed detection results.
<i>passthrough</i>	Passthrough message on which the inference was performed.

Properties

alias of `depthai.DetectionNetworkProperties` **Methods:**

<code>__init__(*args, **kwargs)</code>	Initialize self.
--	------------------

Attributes:

<code>anchorMasks</code>
<code>anchors</code>
<code>classes</code>
<code>confidenceThreshold</code>
<code>coordinates</code>
<code>iouThreshold</code>
<code>nnFamily</code>

`__init__(*args, **kwargs)`
Initialize self. See `help(type(self))` for accurate signature.

property input

Input message with data to be inferred upon Default queue is blocking with size 5

property out

Outputs ImgDetections message that carries parsed detection results.

property passthrough

Passthrough message on which the inference was performed.

Suitable for when input queue is set to non-blocking behavior.

setConfidenceThreshold (*self*: [depthai.DetectionNetwork](#), *thresh*: *float*) → *None*

Specifies confidence threshold at which to filter the rest of the detections.

Parameter thresh: Detection confidence must be greater than specified threshold to be added to the list

class [depthai.DetectionNetworkProperties](#)

Bases: [depthai.NeuralNetworkProperties](#)

Properties for DetectionNetwork

Methods:

<code>__init__(*args, **kwargs)</code>	Initialize self.
--	------------------

Attributes:

<code>anchorMasks</code>
<code>anchors</code>
<code>classes</code>
<code>confidenceThreshold</code>
<code>coordinates</code>
<code>iouThreshold</code>
<code>nnFamily</code>

`__init__(*args, **kwargs)`

Initialize self. See help(type(self)) for accurate signature.

property `anchorMasks`

property `anchors`

property `classes`

property `confidenceThreshold`

property `coordinates`

property `iouThreshold`

property `nnFamily`

class [depthai.Device](#)

Bases: `pybind11_builtins.pybind11_object`

Represents the DepthAI device with the methods to interact with it.

Methods:

<code>__init__(*args, **kwargs)</code>	Overloaded function.
<code>addLogCallback(self, callback)</code>	Add a callback for device logging.
<code>close(self)</code>	Closes the connection to device.
<code>getAllAvailableDevices()</code>	Returns all connected devices
<code>getAnyAvailableDevice(*args, **kwargs)</code>	Overloaded function.
<code>getChipTemperature(self)</code>	Retrieves current chip temperature as measured by device
<code>getCmxMemoryUsage(self)</code>	Retrieves current CMX memory information from device

continues on next page

Table 49 – continued from previous page

<i>getDdrMemoryUsage</i> (self)	Retrieves current DDR memory information from device
<i>getDeviceByMxId</i> (mxId)	Finds a device by MX ID.
<i>getEmbeddedDeviceBinary</i> (usb2Mode, version)	Gets device firmware binary for a specific OpenVINO version
<i>getFirstAvailableDevice</i> ()	Gets first available device.
<i>getInputQueue</i> (*args, **kwargs)	Overloaded function.
<i>getInputQueueNames</i> (self)	Get all available input queue names
<i>getLeonCssCpuUsage</i> (self)	Retrieves average CSS Leon CPU usage
<i>getLeonCssHeapUsage</i> (self)	Retrieves current CSS Leon CPU heap information from device
<i>getLeonMssCpuUsage</i> (self)	Retrieves average MSS Leon CPU usage
<i>getLeonMssHeapUsage</i> (self)	Retrieves current MSS Leon CPU heap information from device
<i>getLogLevel</i> (self)	Gets current logging severity level of the device.
<i>getLogOutputLevel</i> (self)	Gets logging level which decides printing level to standard output.
<i>getOutputQueue</i> (*args, **kwargs)	Overloaded function.
<i>getOutputQueueNames</i> (self)	Get all available output queue names
<i>getQueueEvent</i> (*args, **kwargs)	Overloaded function.
<i>getQueueEvents</i> (*args, **kwargs)	Overloaded function.
<i>getSystemInformationLoggingRate</i> (self)	Gets current rate of system information logging (“info” severity) in Hz.
<i>isPipelineRunning</i> (self)	Checks if devices pipeline is already running
<i>removeLogCallback</i> (self, callbackId)	Removes a callback
<i>setLogLevel</i> (self, level)	Sets the devices logging severity level.
<i>setLogOutputLevel</i> (self, level)	Sets logging level which decides printing level to standard output.
<i>setSystemInformationLoggingRate</i> (self, rateHz)	Sets rate of system information logging (“info” severity).
<i>startPipeline</i> (self)	Starts the execution of the devices pipeline

__init__ (*args, **kwargs)
Overloaded function.

1. **__init__**(self: depthai.Device, pipeline: depthai.Pipeline) -> None

Connects to any available device with a DEFAULT_SEARCH_TIME timeout.

Parameter pipeline:

- Pipeline to be executed on the device

2. **__init__**(self: depthai.Device, pipeline: depthai.Pipeline, usb2Mode: bool) -> None

Connects to any available device with a DEFAULT_SEARCH_TIME timeout.

Parameter pipeline:

- Pipeline to be executed on the device

Parameter usb2Mode:

- Boot device using USB2 mode firmware

3. **__init__**(self: depthai.Device, pipeline: depthai.Pipeline, pathToCmd: str) -> None

Connects to any available device with a DEFAULT_SEARCH_TIME timeout.

Parameter pipeline:

- Pipeline to be executed on the device

Parameter pathToCmd:

- Path to custom device firmware

4. `__init__(self: depthai.Device, pipeline: depthai.Pipeline, deviceDesc: depthai.DeviceInfo, usb2Mode: bool = False) -> None`

Connects to any available device with a DEFAULT_SEARCH_TIME timeout.

Parameter pipeline:

- Pipeline to be executed on the device

Parameter pathToCmd:

- Path to custom device firmware

5. `__init__(self: depthai.Device, pipeline: depthai.Pipeline, deviceDesc: depthai.DeviceInfo, pathToCmd: str) -> None`

Connects to device specified by devInfo.

Parameter pipeline:

- Pipeline to be executed on the device

Parameter devInfo:

- DeviceInfo which specifies which device to connect to

Parameter usb2Mode:

- Boot device using USB2 mode firmware

addLogCallback (*self: depthai.Device, callback: std::function<void (dai::LogMessage)>*) \rightarrow `int`
Add a callback for device logging. The callback will be called from a separate thread with the LogMessage being passed.

Parameter callback:

- Callback to call whenever a log message arrives

Returns Id which can be used to later remove the callback

close (*self: depthai.Device*) \rightarrow `None`

Closes the connection to device. Better alternative is the usage of context manager: *with depthai.Device(pipeline) as device:*

static getAllAvailableDevices () \rightarrow List[*depthai.DeviceInfo*]

Returns all connected devices

Returns vector of connected devices

static getAnyAvailableDevice (*args, **kwargs)

Overloaded function.

1. `getAnyAvailableDevice(timeout: datetime.timedelta) -> Tuple[bool, depthai.DeviceInfo]`

Waits for any available device with a timeout

Parameter timeout:

- duration of time to wait for the any device

Returns a tuple of bool and DeviceInfo. Bool specifies if device was found. DeviceInfo specifies the found device

2. `getAnyAvailableDevice()` -> Tuple[bool, depthai.DeviceInfo]

Gets any available device

Returns a tuple of bool and DeviceInfo. Bool specifies if device was found. DeviceInfo specifies the found device

getChipTemperature (*self*: depthai.Device) -> dai::ChipTemperature
Retrieves current chip temperature as measured by device

Returns Temperature of various onboard sensors

getCmxMemoryUsage (*self*: depthai.Device) -> dai::MemoryInfo
Retrieves current CMX memory information from device

Returns Used, remaining and total cmx memory

getDdrMemoryUsage (*self*: depthai.Device) -> dai::MemoryInfo
Retrieves current DDR memory information from device

Returns Used, remaining and total ddr memory

static getDeviceByMxId (*mxId*: str) -> Tuple[bool, depthai.DeviceInfo]
Finds a device by MX ID. Example: 14442C10D13EABCE00

Parameter mxId:

- MyraidX ID which uniquely specifies a device

Returns a tuple of bool and DeviceInfo. Bool specifies if device was found. DeviceInfo specifies the found device

static getEmbeddedDeviceBinary (*usb2Mode*: bool, *version*: depthai.OpenVINO.Version = <Version.VERSION_2021_3: 6>) -> List[int]
Gets device firmware binary for a specific OpenVINO version

Parameter usb2Mode:

- USB2 mode firmware

Parameter version:

- Version of OpenVINO which firmware will support

Returns firmware binary

static getFirstAvailableDevice () -> Tuple[bool, depthai.DeviceInfo]
Gets first available device. Device can be either in XLINK_UNBOOTED or XLINK_BOOTLOADER state

Returns a tuple of bool and DeviceInfo. Bool specifies if device was found. DeviceInfo specifies the found device

getInputQueue (*args, **kwargs)
Overloaded function.

1. `getInputQueue(self: depthai.Device, name: str) -> dai::DataInputQueue`

Gets an input queue corresponding to stream name. If it doesn't exist it throws

Parameter name: Queue/stream name, set in XLinkIn node

Returns Smart pointer to DataInputQueue

2. `getInputQueue(self: depthai.Device, name: str, maxSize: int, blocking: bool = True) -> dai::DataInputQueue`

Gets an input queue corresponding to stream name. If it doesn't exist it throws. Also sets queue options

Parameter name: Queue/stream name, set in XLinkOut node

Parameter maxSize: Maximum number of messages in queue

Parameter blocking: Queue behavior once full. True: blocking, false: overwriting of oldest messages.
Default: true

Returns Smart pointer to DataInputQueue

getInputQueueNames (*self*: `depthai.Device`) → List[str]

Get all available input queue names

Returns Vector of input queue names

getLeonCssCpuUsage (*self*: `depthai.Device`) → dai::CpuUsage

Retrieves average CSS Leon CPU usage

Returns Average CPU usage and sampling duration

getLeonCssHeapUsage (*self*: `depthai.Device`) → dai::MemoryInfo

Retrieves current CSS Leon CPU heap information from device

Returns Used, remaining and total heap memory

getLeonMssCpuUsage (*self*: `depthai.Device`) → dai::CpuUsage

Retrieves average MSS Leon CPU usage

Returns Average CPU usage and sampling duration

getLeonMssHeapUsage (*self*: `depthai.Device`) → dai::MemoryInfo

Retrieves current MSS Leon CPU heap information from device

Returns Used, remaining and total heap memory

getLogLevel (*self*: `depthai.Device`) → dai::LogLevel

Gets current logging severity level of the device.

Returns Logging severity level

getLogOutputLevel (*self*: `depthai.Device`) → dai::LogLevel

Gets logging level which decides printing level to standard output.

Returns Standard output printing severity

getOutputQueue (*args, **kwargs)

Overloaded function.

1. `getOutputQueue(self: depthai.Device, name: str) -> dai::DataOutputQueue`

Gets an output queue corresponding to stream name. If it doesn't exist it throws

Parameter name: Queue/stream name, created by XLinkOut node

Returns Smart pointer to DataOutputQueue

2. `getOutputQueue(self: depthai.Device, name: str, maxSize: int, blocking: bool = True) -> dai::DataOutputQueue`

Gets a queue corresponding to stream name, if it exists, otherwise it throws. Also sets queue options

Parameter name: Queue/stream name, set in XLinkOut node

Parameter maxSize: Maximum number of messages in queue

Parameter blocking: Queue behavior once full. True specifies blocking and false overwriting of oldest messages. Default: true

Returns Smart pointer to DataOutputQueue

getOutputQueueNames (*self*: `depthai.Device`) \rightarrow `List[str]`

Get all available output queue names

Returns Vector of output queue names

getQueueEvent (**args*, ***kwargs*)

Overloaded function.

1. `getQueueEvent(self: depthai.Device, queueNames: List[str], timeout: datetime.timedelta = datetime.timedelta(days=-1, seconds=86399, microseconds=999999)) -> str`

Gets or waits until any of specified queues has received a message

Parameter queueNames: Names of queues for which to wait for

Parameter timeout: Timeout after which return regardless. If negative then wait is indefinite. Default is -1

Returns Queue name which received a message first

2. `getQueueEvent(self: depthai.Device, queueName: str, timeout: datetime.timedelta = datetime.timedelta(days=-1, seconds=86399, microseconds=999999)) -> str`

Gets or waits until specified queue has received a message

Parameter queueNames: Name of queues for which to wait for

Parameter timeout: Timeout after which return regardless. If negative then wait is indefinite. Default is -1

Returns Queue name which received a message

3. `getQueueEvent(self: depthai.Device, timeout: datetime.timedelta = datetime.timedelta(days=-1, seconds=86399, microseconds=999999)) -> str`

Gets or waits until any queue has received a message

Parameter timeout: Timeout after which return regardless. If negative then wait is indefinite. Default is -1

Returns Queue name which received a message

getQueueEvents (**args*, ***kwargs*)

Overloaded function.

1. `getQueueEvents(self: depthai.Device, queueNames: List[str], maxNumEvents: int = 18446744073709551615, timeout: datetime.timedelta = datetime.timedelta(days=-1, seconds=86399, microseconds=999999)) -> List[str]`

Gets or waits until any of specified queues has received a message

Parameter queueNames: Names of queues for which to block

Parameter maxNumEvents: Maximum number of events to remove from queue - Default is unlimited

Parameter timeout: Timeout after which return regardless. If negative then wait is indefinite - Default is -1

Returns Names of queues which received messages first

2. `getQueueEvents(self: depthai.Device, queueName: str, maxNumEvents: int = 18446744073709551615, timeout: datetime.timedelta = datetime.timedelta(days=-1, seconds=86399, microseconds=999999)) -> List[str]`

Gets or waits until specified queue has received a message

Parameter queueName: Name of queues for which to wait for

Parameter maxNumEvents: Maximum number of events to remove from queue. Default is unlimited

Parameter timeout: Timeout after which return regardless. If negative then wait is indefinite. Default is -1

Returns Names of queues which received messages first

3. `getQueueEvents(self: depthai.Device, maxNumEvents: int = 18446744073709551615, timeout: datetime.timedelta = datetime.timedelta(days=-1, seconds=86399, microseconds=999999)) -> List[str]`

Gets or waits until any any queue has received a message

Parameter maxNumEvents: Maximum number of events to remove from queue. Default is unlimited

Parameter timeout: Timeout after which return regardless. If negative then wait is indefinite. Default is -1

Returns Names of queues which received messages first

getSystemInformationLoggingRate (*self: depthai.Device*) \rightarrow float

Gets current rate of system information logging ("info" severity) in Hz.

Returns Logging rate in Hz

isPipelineRunning (*self: depthai.Device*) \rightarrow bool

Checks if devices pipeline is already running

Returns true if running, false otherwise

removeLogCallback (*self: depthai.Device, callbackId: int*) \rightarrow bool

Removes a callback

Parameter callbackId: Id of callback to be removed

Returns true if callback was removed, false otherwise

setLogLevel (*self: depthai.Device, level: dai::LogLevel*) \rightarrow None

Sets the devices logging severity level. This level affects which logs are transferred from device to host.

Parameter level: Logging severity

setLogOutputLevel (*self*: *depthai.Device*, *level*: *dai::LogLevel*) → *None*

Sets logging level which decides printing level to standard output. If lower than setLogLevel, no messages will be printed

Parameter level:

- Standard output printing severity

setSystemInformationLoggingRate (*self*: *depthai.Device*, *rateHz*: *float*) → *None*

Sets rate of system information logging (“info” severity). Default 1Hz If parameter is less or equal to zero, then system information logging will be disabled

Parameter rateHz: Logging rate in Hz

startPipeline (*self*: *depthai.Device*) → *bool*

Starts the execution of the devices pipeline

Returns true if pipeline started, false otherwise

class *depthai.DeviceBootloader*

Bases: *pybind11_builtins.pybind11_object*

Represents the DepthAI bootloader with the methods to interact with it.

Classes:

<i>Version</i>	Bootloader version structure
Methods:	
<i>__init__</i> (*args, **kwargs)	Overloaded function.
<i>close</i> (self)	Closes the connection to device.
<i>createDepthaiApplicationPackage</i> (pipeline, ...)	Creates application package which can be flashed to depthai device.
<i>flash</i> (self, progressCallback, None], pipeline)	Flashes a give pipeline to the board.
<i>flashBootloader</i> (self, progressCallback, ...)	Flashes bootloader to the current board
<i>flashDepthaiApplicationPackage</i> (self, ...)	Flashes a specific depthai application package that was generated using <i>createDepthaiApplicationPackage</i> or <i>saveDepthaiApplicationPackage</i>
<i>getAllAvailableDevices</i> ()	Searches for connected devices in either UN-BOOTED or BOOTLOADER states.
<i>getEmbeddedBootloaderBinary</i> ()	returns Embedded bootloader binary
<i>getEmbeddedBootloaderVersion</i> ()	returns Embedded bootloader version
<i>getFirstAvailableDevice</i> ()	Searches for connected devices in either UN-BOOTED or BOOTLOADER states and returns first available.
<i>getVersion</i> (self)	returns Version of current running bootloader

continues on next page

Table 51 – continued from previous page

<code>isEmbeddedVersion(self)</code>	returns True whether the bootloader running is flashed or booted by library
<code>saveDepthaiApplicationPackage(path, ...)</code>	Saves application package to a file which can be flashed to depthai device.

class VersionBases: `pybind11_builtins.pybind11_object`

Bootloader version structure

Methods:

<code>__init__(*args, **kwargs)</code>	Overloaded function.
<code>__init__(*args, **kwargs)</code> Overloaded function. 1. <code>__init__(self: depthai.DeviceBootloader.Version, v: str) -> None</code> Construct Version from string 2. <code>__init__(self: depthai.DeviceBootloader.Version, major: int, minor: int, patch: int) -> None</code> Construct Version major, minor and patch numbers	
<code>__init__(*args, **kwargs)</code>	Overloaded function. 1. <code>__init__(self: depthai.DeviceBootloader, deviceDesc: depthai.DeviceInfo) -> None</code> 2. <code>__init__(self: depthai.DeviceBootloader, deviceDesc: depthai.DeviceInfo, pathToCmd: str) -> None</code> Connects to or boots device in bootloader mode depending on devInfo state. Parameter devInfo: DeviceInfo of which to boot or connect to
<code>close(self: depthai.DeviceBootloader) -> None</code>	Closes the connection to device. Better alternative is the usage of context manager: <i>with depthai.DeviceBootloader(deviceInfo) as bootloader:</i>
<code>static createDepthaiApplicationPackage(pipeline: depthai.Pipeline, pathToCmd: str = "") -> List[int]</code>	Creates application package which can be flashed to depthai device. Parameter pipeline: Pipeline from which to create the application package Parameter pathToCmd: Optional path to custom device firmware Returns Depthai application package
<code>flash(self: depthai.DeviceBootloader, progressCallback: Callable[[float], None], pipeline: depthai.Pipeline) -> Tuple[bool, str]</code>	Flashes a give pipeline to the board. Parameter progressCallback: Callback that sends back a value between 0..1 which signifies current flashing progress Parameter pipeline: Pipeline to flash to the board

flashBootloader (*self*: [depthai.DeviceBootloader](#), *progressCallback*: [Callable\[\[float\], None\]](#), *path*: *str* = "") → [Tuple\[bool, str\]](#)

Flashes bootloader to the current board

Parameter progressCallback: Callback that sends back a value between 0..1 which signifies current flashing progress

Parameter path: Optional parameter to custom bootloader to flash

flashDepthaiApplicationPackage (*self*: [depthai.DeviceBootloader](#), *progressCallback*: [Callable\[\[float\], None\]](#), *package*: [List\[int\]](#)) → [Tuple\[bool, str\]](#)

Flashes a specific depthai application package that was generated using [createDepthaiApplicationPackage](#) or [saveDepthaiApplicationPackage](#)

Parameter progressCallback: Callback that sends back a value between 0..1 which signifies current flashing progress

Parameter package: Depthai application package to flash to the board

static getAllAvailableDevices () → [List\[depthai.DeviceInfo\]](#)

Searches for connected devices in either UNBOOTED or BOOTLOADER states.

Returns Vector of all found devices

static getEmbeddedBootloaderBinary () → [List\[int\]](#)

Returns Embedded bootloader binary

static getEmbeddedBootloaderVersion () → [depthai.DeviceBootloader.Version](#)

Returns Embedded bootloader version

static getFirstAvailableDevice () → [Tuple\[bool, depthai.DeviceInfo\]](#)

Searches for connected devices in either UNBOOTED or BOOTLOADER states and returns first available.

Returns Tuple of boolean and DeviceInfo. If found boolean is true and DeviceInfo describes the device. Otherwise false

getVersion (*self*: [depthai.DeviceBootloader](#)) → [depthai.DeviceBootloader.Version](#)

Returns Version of current running bootloader

isEmbeddedVersion (*self*: [depthai.DeviceBootloader](#)) → [bool](#)

Returns True whether the bootloader running is flashed or booted by library

static saveDepthaiApplicationPackage (*path*: *str*, *pipeline*: [depthai.Pipeline](#), *pathToCmd*: *str* = "") → [None](#)

Saves application package to a file which can be flashed to depthai device.

Parameter path: Path where to save the application package

Parameter pipeline: Pipeline from which to create the application package

Parameter pathToCmd: Optional path to custom device firmware

class [depthai.DeviceDesc](#)

Bases: [pybind11_builtins.pybind11_object](#)

Methods:

[__init__](#) (*self*)

Attributes:

name

platform

protocol

__init__ (*self*: depthai.DeviceDesc) → None**property** name**property** platform**property** protocol**class** depthai.DeviceInfo

Bases: pybind11_builtins.pybind11_object

Methods:

__init__ (*self*)

getMxId (*self*)

Attributes:

desc

state

__init__ (*self*: depthai.DeviceInfo) → None**property** desc**getMxId** (*self*: depthai.DeviceInfo) → str**property** state**class** depthai.GlobalProperties

Bases: pybind11_builtins.pybind11_object

Specify properties which apply for whole pipeline

Methods:

__init__ (*args, **kwargs) Initialize self.

Attributes:

leonOsFrequencyHz

leonRtFrequencyHz

pipelineName

pipelineVersion

__init__ (*args, **kwargs)

Initialize self. See help(type(self)) for accurate signature.

property leonOsFrequencyHz**property** leonRtFrequencyHz**property** pipelineName

property pipelineVersion

class depthai.ImageManip

Bases: *depthai.Node*

ImageManip node. Capability to crop, resize, warp, ... incoming image frames

Methods:

<code>__init__(*args, **kwargs)</code>	Initialize self.
<code>setCenterCrop(self, arg0, arg1)</code>	
<code>setCropRect(self, arg0, arg1, arg2, arg3)</code>	
<code>setFrameType(self, arg0)</code>	
<code>setHorizontalFlip(self, arg0)</code>	
<code>setKeepAspectRatio(self, arg0)</code>	
<code>setMaxOutputFrameSize(self, arg0)</code>	Specify maximum size of output image.
<code>setNumFramesPool(self, arg0)</code>	Specify number of frames in pool.
<code>setResize(self, arg0, arg1)</code>	
<code>setResizeThumbnail(self, arg0, arg1, arg2, ...)</code>	
<code>setWaitForConfigInput(self, arg0)</code>	Specify whether or not wait until configuration message arrives to inputConfig Input.

Attributes:

<code>initialConfig</code>	Initial config to use when manipulating frames
<code>inputConfig</code>	Input ImageManipConfig message with ability to modify parameters in runtime Default queue is blocking with size 8
<code>inputImage</code>	Input image to be modified Default queue is blocking with size 8
<code>out</code>	Outputs ImgFrame message that carries modified image.

`__init__(*args, **kwargs)`

Initialize self. See help(type(self)) for accurate signature.

property initialConfig

Initial config to use when manipulating frames

property inputConfig

Input ImageManipConfig message with ability to modify parameters in runtime Default queue is blocking with size 8

property inputImage

Input image to be modified Default queue is blocking with size 8

property out

Outputs ImgFrame message that carries modified image.

setCenterCrop (*self*: depthai.ImageManip, *arg0*: float, *arg1*: float) → None

setCropRect (*self*: depthai.ImageManip, *arg0*: float, *arg1*: float, *arg2*: float, *arg3*: float) → None

setFrameType (*self*: depthai.ImageManip, *arg0*: dai::RawImgFrame::Type) → None

setHorizontalFlip (*self*: depthai.ImageManip, *arg0*: bool) → None

setKeepAspectRatio (*self*: depthai.ImageManip, *arg0*: bool) → None

setMaxOutputFrameSize (*self*: depthai.ImageManip, *arg0*: int) → None
Specify maximum size of output image.

Parameter maxFrameSize: Maximum frame size in bytes

setNumFramesPool (*self*: depthai.ImageManip, *arg0*: int) → None
Specify number of frames in pool.

Parameter numFramesPool: How many frames should the pool have

setResize (*self*: depthai.ImageManip, *arg0*: int, *arg1*: int) → None

setResizeThumbnail (*self*: depthai.ImageManip, *arg0*: int, *arg1*: int, *arg2*: int, *arg3*: int, *arg4*: int) → None

setWaitForConfigInput (*self*: depthai.ImageManip, *arg0*: bool) → None
Specify whether or not wait until configuration message arrives to inputConfig Input.

Parameter wait: True to wait for configuration message, false otherwise

class depthai.ImageManipConfig

Bases: *depthai.Buffer*

ImageManipConfig message. Specifies image manipulation options like:

- Crop
- Resize
- Warp
- ...

Methods:

<code>__init__(self)</code>	
<code>getCropXMax(self)</code>	returns Bottom right X coordinate of crop region
<code>getCropXMin(self)</code>	returns Top left X coordinate of crop region
<code>getCropYMax(self)</code>	returns Bottom right Y coordinate of crop region
<code>getCropYMin(self)</code>	returns Top left Y coordinate of crop region
<code>getResizeHeight(self)</code>	returns Output image height
<code>getResizeWidth(self)</code>	returns Output image width

continues on next page

Table 61 – continued from previous page

<i>isResizeThumbnail</i> (self)	returns True if resize thumbnail mode is set, false otherwise
<i>setCenterCrop</i> (self, ratio, whRatio)	Specifies a centered crop.
<i>setCropRect</i> (self, xmin, ymin, xmax, ymax)	Specifies crop with rectangle with normalized values (0..1)
<i>setCropRotatedRect</i> (self, rr, normalizedCoords)	Specifies crop with rotated rectangle.
<i>setFrameType</i> (self, name)	Specify output frame type.
<i>setHorizontalFlip</i> (self, flip)	Specify horizontal flip
<i>setKeepAspectRatio</i> (self, keep)	Specifies to whether to keep aspect ratio or not
<i>setResize</i> (self, w, h)	Specifies output image size.
<i>setResizeThumbnail</i> (self, w, h, bgRed, ...)	Specifies output image size.
<i>setReusePreviousImage</i> (self, reuse)	Instruct ImageManip to not remove current image from its queue and use the same for next message.
<i>setRotationDegrees</i> (self, deg)	Specifies clockwise rotation in degrees
<i>setRotationRadians</i> (self, rad)	Specifies clockwise rotation in radians
<i>setSkipCurrentImage</i> (self, skip)	Instructs ImageManip to skip current image and wait for next in queue.
<i>setWarpBorderFillColor</i> (self, red, green, blue)	Specifies fill color for border pixels.
<i>setWarpBorderReplicatePixels</i> (self)	Specifies that warp replicates border pixels
<i>setWarpTransformFourPoints</i> (self, pt, ...)	Specifies warp by supplying 4 points in either absolute or normalized coordinates
<i>setWarpTransformMatrix3x3</i> (self, mat)	Specifies warp with a 3x3 matrix

__init__ (self: depthai.ImageManipConfig) → None

getCropXMax (self: depthai.ImageManipConfig) → float

Returns Bottom right X coordinate of crop region

getCropXMin (self: depthai.ImageManipConfig) → float

Returns Top left X coordinate of crop region

getCropYMax (self: depthai.ImageManipConfig) → float

Returns Bottom right Y coordinate of crop region

getCropYMin (self: depthai.ImageManipConfig) → float

Returns Top left Y coordinate of crop region

getResizeHeight (self: depthai.ImageManipConfig) → int

Returns Output image height

getResizeWidth (self: depthai.ImageManipConfig) → int

Returns Output image width

isResizeThumbnail (self: depthai.ImageManipConfig) → bool

Returns True if resize thumbnail mode is set, false otherwise

setCenterCrop (self: depthai.ImageManipConfig, ratio: float, whRatio: float = 1.0) → None
Specifies a centered crop.

Parameter ratio: Ratio between input image and crop region (0..1)

Parameter whRatio: Crop region aspect ratio - 1 equals to square, 1.7 equals to 16:9, ...

ImageManipConfig.setCropRect (*self*: `depthai.ImageManipConfig`, *xmin*: `float`, *ymin*: `float`,
Specifies crop with rectangle with normalized values (0..1)

Parameter xmin: Top left X coordinate of rectangle

Parameter ymin: Top left Y coordinate of rectangle

Parameter xmax: Bottom right X coordinate of rectangle

Parameter ymax: Bottom right Y coordinate of rectangle

setCropRotatedRect (*self*: `depthai.ImageManipConfig`, *rr*: `depthai.RotatedRect`, *normalizedCo-*
ords: `bool = True`) → `None`
Specifies crop with rotated rectangle. Optionally as non normalized coordinates

Parameter rr: Rotated rectangle which specifies crop

Parameter normalizedCoords: If true coordinates are in normalized range (0..1) otherwise absolute

setFrameType (*self*: `depthai.ImageManipConfig`, *name*: `depthai.RawImgFrame.Type`) → `None`
Specify output frame type.

Parameter name: Frame type

setHorizontalFlip (*self*: `depthai.ImageManipConfig`, *flip*: `bool`) → `None`
Specify horizontal flip

Parameter flip: True to enable flip, false otherwise

setKeepAspectRatio (*self*: `depthai.ImageManipConfig`, *keep*: `bool`) → `None`
Specifies to whether to keep aspect ratio or not

setResize (*self*: `depthai.ImageManipConfig`, *w*: `int`, *h*: `int`) → `None`
Specifies output image size. After crop stage the image will be stretched to fit.

Parameter w: Width in pixels

Parameter h: Height in pixels

setResizeThumbnail (*self*: `depthai.ImageManipConfig`, *w*: `int`, *h*: `int`, *bgRed*: `int = 0`, *bgGreen*: `int`
= 0, *bgBlue*: `int = 0`) → `None`
Specifies output image size. After crop stage the image will be resized by preserving aspect ration. Op-
tionally background can be specified.

Parameter w: Width in pixels

Parameter h: Height in pixels

Parameter bgRed: Red component

Parameter bgGreen: Green component

Parameter bgBlue: Blue component

setReusePreviousImage (*self*: `depthai.ImageManipConfig`, *reuse*: `bool`) → `None`
Instruct ImageManip to not remove current image from its queue and use the same for next message.

Parameter reuse: True to enable reuse, false otherwise

setRotationDegrees (*self*: `depthai.ImageManipConfig`, *deg*: `float`) → `None`
Specifies clockwise rotation in degrees

Parameter deg: Rotation in degrees

setRotationRadians (*self*: depthai.ImageManipConfig, *rad*: float) → None

Specifies clockwise rotation in radians

Parameter rad: Rotation in radians

setSkipCurrentImage (*self*: depthai.ImageManipConfig, *skip*: bool) → None

Instructs ImageManip to skip current image and wait for next in queue.

Parameter skip: True to skip current image, false otherwise

setWarpBorderFillColor (*self*: depthai.ImageManipConfig, *red*: int, *green*: int, *blue*: int) → None

Specifies fill color for border pixels. Example:

- setWarpBorderFillColor(255,255,255) -> white
- setWarpBorderFillColor(0,0,255) -> blue

Parameter red: Red component

Parameter green: Green component

Parameter blue: Blue component

setWarpBorderReplicatePixels (*self*: depthai.ImageManipConfig) → None

Specifies that warp replicates border pixels

setWarpTransformFourPoints (*self*: depthai.ImageManipConfig, *pt*: List[depthai.Point2f], *normalizedCoords*: bool) → None

Specifies warp by supplying 4 points in either absolute or normalized coordinates

Parameter pt: 4 points specifying warp

Parameter normalizedCoords: If true pt is interpreted as normalized, absolute otherwise

setWarpTransformMatrix3x3 (*self*: depthai.ImageManipConfig, *mat*: List[float]) → None

Specifies warp with a 3x3 matrix

Parameter mat: 3x3 matrix

class depthai.ImgDetection

Bases: pybind11_builtins.pybind11_object

Methods:

`__init__`(self)

Attributes:

`confidence`

`label`

`xmax`

`xmin`

`ymax`

`ymin`

`__init__` (*self*: depthai.ImgDetection) → None

property confidence

property label

property xmax**property** xmin**property** ymax**property** ymin**class** depthai.ImgDetectionsBases: *depthai.Buffer*

ImgDetections message. Carries normalized detection results

Methods:

<code>__init__(self)</code>	Construct ImgDetections message
-----------------------------	---------------------------------

Attributes:

<code>detections</code>	Detections
-------------------------	------------

`__init__(self: depthai.ImgDetections) → None`
Construct ImgDetections message

property detections
Detections

class depthai.ImgFrameBases: *depthai.Buffer*

ImgFrame message. Carries image data and metadata.

Classes:

<i>Specs</i>	
<i>Type</i>	Members:

Methods:

<code>__init__(self)</code>	
<code>getCategory(self)</code>	Retrieves image category
<code>getCvFrame(self)</code>	Returns BGR or grayscale frame compatible with use in other opencv functions
<code>getFrame(self, copy)</code>	Returns numpy array with shape as specified by width, height and type
<code>getHeight(self)</code>	Retrieves image height in pixels
<code>getInstanceNum(self)</code>	Retrieves instance number
<code>getSequenceNum(self)</code>	Retrieves image sequence number
<code>getTimestamp(self)</code>	Retrieves image timestamp related to steady_clock / time.monotonic
<code>getType(self)</code>	Retrieves image type
<code>getWidth(self)</code>	Retrieves image width in pixels
<code>setCategory(self, category)</code>	Parameter category:
<code>setFrame(self, array)</code>	Copies array bytes to ImgFrame buffer
<code>setHeight(self, height)</code>	Specifies frame height

continues on next page

Table 67 – continued from previous page

<i>setInstanceNum</i> (self, instance)	Instance number relates to the origin of the frame (which camera)
<i>setSequenceNum</i> (self, seq)	Specifies sequence number
<i>setTimestamp</i> (self, timestamp)	Specifies current timestamp, related to steady_clock / time.monotonic
<i>setType</i> (self, type)	Specifies frame type, RGB, BGR, ...
<i>setWidth</i> (self, width)	Specifies frame width

class SpecsBases: `pybind11_builtins.pybind11_object`**Methods:**

<code>__init__</code> (*args, **kwargs)	Initialize self.
---	------------------

Attributes:

<code>bytesPP</code>
<code>height</code>
<code>p1Offset</code>
<code>p2Offset</code>
<code>p3Offset</code>
<code>stride</code>
<code>type</code>
<code>width</code>

`__init__`(*args, **kwargs)
Initialize self. See `help(type(self))` for accurate signature.

property `bytesPP`**property** `height`**property** `p1Offset`**property** `p2Offset`**property** `p3Offset`**property** `stride`**property** `type`**property** `width`**class Type**Bases: `pybind11_builtins.pybind11_object`**Members:**`YUV422i``YUV444p``YUV420p``YUV422p``YUV400p`

RGBA8888
RGB161616
RGB888p
BGR888p
RGB888i
BGR888i
RGBF16F16F16p
BGRF16F16F16p
RGBF16F16F16i
BGRF16F16F16i
GRAY8
GRAYF16
LUT2
LUT4
LUT16
RAW16
RAW14
RAW12
RAW10
RAW8
PACK10
PACK12
YUV444i
NV12
NV21
BITSTREAM
HDR
NONE

Attributes:

<i>BGR888i</i>
<i>BGR888p</i>
<i>BGRF16F16F16i</i>
<i>BGRF16F16F16p</i>
<i>BITSTREAM</i>
<i>GRAY8</i>
<i>GRAYF16</i>
<i>HDR</i>

continues on next page

Table 70 – continued from previous page

<i>LUT16</i>
<i>LUT2</i>
<i>LUT4</i>
<i>NONE</i>
<i>NV12</i>
<i>NV21</i>
<i>PACK10</i>
<i>PACK12</i>
<i>RAW10</i>
<i>RAW12</i>
<i>RAW14</i>
<i>RAW16</i>
<i>RAW8</i>
<i>RGB161616</i>
<i>RGB888i</i>
<i>RGB888p</i>
<i>RGBA8888</i>
<i>RGBF16F16F16i</i>
<i>RGBF16F16F16p</i>
<i>YUV400p</i>
<i>YUV420p</i>
<i>YUV422i</i>
<i>YUV422p</i>
<i>YUV444i</i>
<i>YUV444p</i>
<i>name</i>
<i>value</i>

Methods:

<code>__init__(self, value)</code>
<code>BGR888i = <Type.BGR888i: 10></code>
<code>BGR888p = <Type.BGR888p: 8></code>
<code>BGRF16F16F16i = <Type.BGRF16F16F16i: 14></code>
<code>BGRF16F16F16p = <Type.BGRF16F16F16p: 12></code>
<code>BITSTREAM = <Type.BITSTREAM: 30></code>
<code>GRAY8 = <Type.GRAY8: 15></code>
<code>GRAYF16 = <Type.GRAYF16: 16></code>
<code>HDR = <Type.HDR: 31></code>
<code>LUT16 = <Type.LUT16: 19></code>
<code>LUT2 = <Type.LUT2: 17></code>
<code>LUT4 = <Type.LUT4: 18></code>
<code>NONE = <Type.NONE: 32></code>
<code>NV12 = <Type.NV12: 28></code>


```

NV21 = <Type.NV21: 29>
PACK10 = <Type.PACK10: 25>
PACK12 = <Type.PACK12: 26>
RAW10 = <Type.RAW10: 23>
RAW12 = <Type.RAW12: 22>
RAW14 = <Type.RAW14: 21>
RAW16 = <Type.RAW16: 20>
RAW8 = <Type.RAW8: 24>
RGB161616 = <Type.RGB161616: 6>
RGB888i = <Type.RGB888i: 9>
RGB888p = <Type.RGB888p: 7>
RGBA8888 = <Type.RGBA8888: 5>
RGBF16F16F16i = <Type.RGBF16F16F16i: 13>
RGBF16F16F16p = <Type.RGBF16F16F16p: 11>
YUV400p = <Type.YUV400p: 4>
YUV420p = <Type.YUV420p: 2>
YUV422i = <Type.YUV422i: 0>
YUV422p = <Type.YUV422p: 3>
YUV444i = <Type.YUV444i: 27>
YUV444p = <Type.YUV444p: 1>

__init__(self: depthai.RawImgFrame.Type, value: int) → None
property name
property value

__init__(self: depthai.ImgFrame) → None
getCategory(self: depthai.ImgFrame) → int
    Retrieves image category
getCvFrame(self: object) → object
    Returns BGR or grayscale frame compatible with use in other opencv functions
getFrame(self: object, copy: bool = False) → numpy.ndarray
    Returns numpy array with shape as specified by width, height and type
getHeight(self: depthai.ImgFrame) → int
    Retrieves image height in pixels
getInstanceNum(self: depthai.ImgFrame) → int
    Retrieves instance number
getSequenceNum(self: depthai.ImgFrame) → int
    Retrieves image sequence number
getTimestamp(self: depthai.ImgFrame) → datetime.timedelta
    Retrieves image timestamp related to steady_clock / time.monotonic

```

getType (*self*: `depthai.ImgFrame`) → `depthai.RawImgFrame.Type`
Retrieves image type

getWidth (*self*: `depthai.ImgFrame`) → `int`
Retrieves image width in pixels

setCategory (*self*: `depthai.ImgFrame`, *category*: `int`) → `None`

Parameter category: Image category

setFrame (*self*: `depthai.ImgFrame`, *array*: `numpy.ndarray`) → `None`
Copies array bytes to `ImgFrame` buffer

setHeight (*self*: `depthai.ImgFrame`, *height*: `int`) → `None`
Specifies frame height

Parameter width: frame height

setInstanceNum (*self*: `depthai.ImgFrame`, *instance*: `int`) → `None`
Instance number relates to the origin of the frame (which camera)

Parameter instance: Instance number

setSequenceNum (*self*: `depthai.ImgFrame`, *seq*: `int`) → `None`
Specifies sequence number

Parameter seq: Sequence number

setTimestamp (*self*: `depthai.ImgFrame`, *timestamp*: `datetime.timedelta`) → `None`
Specifies current timestamp, related to `steady_clock` / `time.monotonic`

setType (*self*: `depthai.ImgFrame`, *type*: `depthai.RawImgFrame.Type`) → `None`
Specifies frame type, RGB, BGR, ...

Parameter type: Type of image

setWidth (*self*: `depthai.ImgFrame`, *width*: `int`) → `None`
Specifies frame width

Parameter width: frame width

class `depthai.LogLevel`

Bases: `pybind11_builtins.pybind11_object`

Members:

TRACE

DEBUG

INFO

WARN

ERR

CRITICAL

OFF

Attributes:

CRITICAL

DEBUG

ERR

continues on next page

Table 72 – continued from previous page

<i>INFO</i>
<i>OFF</i>
<i>TRACE</i>
<i>WARN</i>
<i>name</i>
<i>value</i>
Methods:
<i>__init__</i> (self, value)
CRITICAL = <LogLevel.CRITICAL: 5>
DEBUG = <LogLevel.DEBUG: 1>
ERR = <LogLevel.ERR: 4>
INFO = <LogLevel.INFO: 2>
OFF = <LogLevel.OFF: 6>
TRACE = <LogLevel.TRACE: 0>
WARN = <LogLevel.WARN: 3>
<i>__init__</i> (self: depthai.LogLevel, value: int) → None
property name
property value
class depthai.MemoryInfo
Bases: pybind11_builtins.pybind11_object
MemoryInfo structure
Free, remaining and total memory stats
Methods:
<i>__init__</i> (self)
Attributes:
<i>remaining</i>
<i>total</i>
<i>used</i>
<i>__init__</i> (self: depthai.MemoryInfo) → None
property remaining
property total
property used
class depthai.MobileNetDetectionNetwork
Bases: <i>depthai.DetectionNetwork</i>
MobileNetDetectionNetwork node. Parses MobileNet results

Methods:

<code>__init__(*args, **kwargs)</code>	Initialize self.
--	------------------

`__init__(*args, **kwargs)`
Initialize self. See help(type(self)) for accurate signature.

class `depthai.MobileNetSpatialDetectionNetwork`

Bases: `depthai.SpatialDetectionNetwork`

MobileNetSpatialDetectionNetwork. Mobilenet-SSD based network with spatial location data.

Methods:

<code>__init__(*args, **kwargs)</code>	Initialize self.
--	------------------

`__init__(*args, **kwargs)`
Initialize self. See help(type(self)) for accurate signature.

class `depthai.MonoCamera`

Bases: `depthai.Node`

MonoCamera node. For use with grayscale sensors.

Classes:

<code>Properties</code>	alias of <code>depthai.MonoCameraProperties</code>
-------------------------	--

Methods:

<code>__init__(*args, **kwargs)</code>	Initialize self.
<code>getBoardSocket(self)</code>	Retrieves which board socket to use
<code>getCamId(self)</code>	
<code>getFps(self)</code>	Get rate at which camera should produce frames
<code>getImageOrientation(self)</code>	Get camera image orientation
<code>getResolution(self)</code>	Get sensor resolution
<code>getResolutionHeight(self)</code>	Get sensor resolution height
<code>getResolutionSize(self)</code>	Get sensor resolution as size
<code>getResolutionWidth(self)</code>	Get sensor resolution width
<code>setBoardSocket(self, boardSocket)</code>	Specify which board socket to use
<code>setCamId(self, arg0)</code>	
<code>setFps(self, fps)</code>	Set rate at which camera should produce frames
<code>setImageOrientation(self, imageOrientation)</code>	Set camera image orientation
<code>setResolution(self, resolution)</code>	Set sensor resolution

Attributes:

<code>initialControl</code>	Initial control options to apply to sensor
<code>inputControl</code>	Input for CameraControl message, which can modify camera parameters in runtime Default queue is blocking with size 8
<code>out</code>	Outputs ImgFrame message that carries RAW8 encoded (grayscale) frame data.

Propertiesalias of `depthai.MonoCameraProperties` **Classes:**

SensorResolution	Select the camera sensor resolution: 1280×720, 1280×800, 640×400
------------------	--

Methods:

<code>__init__(*args, **kwargs)</code>	Initialize self.
--	------------------

Attributes:

boardSocket
fps
initialControl
resolution

`__init__(*args, **kwargs)`
Initialize self. See `help(type(self))` for accurate signature.

getBoardSocket (*self*: `depthai.MonoCamera`) → `dai::CameraBoardSocket`
Retrieves which board socket to use

Returns Board socket to use

getCamId (*self*: `depthai.MonoCamera`) → `int`

getFps (*self*: `depthai.MonoCamera`) → `float`
Get rate at which camera should produce frames

Returns Rate in frames per second

getImageOrientation (*self*: `depthai.MonoCamera`) → `dai::CameraImageOrientation`
Get camera image orientation

getResolution (*self*: `depthai.MonoCamera`) → `dai::MonoCameraProperties::SensorResolution`
Get sensor resolution

getResolutionHeight (*self*: `depthai.MonoCamera`) → `int`
Get sensor resolution height

getResolutionSize (*self*: `depthai.MonoCamera`) → `Tuple[int, int]`
Get sensor resolution as size

getResolutionWidth (*self*: `depthai.MonoCamera`) → `int`
Get sensor resolution width

property initialControl
Initial control options to apply to sensor

property inputControl
Input for CameraControl message, which can modify camera parameters in runtime Default queue is blocking with size 8

property out
Outputs ImgFrame message that carries RAW8 encoded (grayscale) frame data.
Suitable for use StereoDepth node

setBoardSocket (*self*: *depthai.MonoCamera*, *boardSocket*: *dai::CameraBoardSocket*) → *None*
Specify which board socket to use

Parameter boardSocket: Board socket to use

setCamId (*self*: *depthai.MonoCamera*, *arg0*: *int*) → *None*

setFps (*self*: *depthai.MonoCamera*, *fps*: *float*) → *None*

Set rate at which camera should produce frames

Parameter fps: Rate in frames per second

setImageOrientation (*self*: *depthai.MonoCamera*, *imageOrientation*: *dai::CameraImageOrientation*) → *None*
Set camera image orientation

setResolution (*self*: *depthai.MonoCamera*, *resolution*: *dai::MonoCameraProperties::SensorResolution*) → *None*
Set sensor resolution

class *depthai.MonoCameraProperties*

Bases: *pybind11_builtins.pybind11_object*

Specify MonoCamera options such as camera ID, ...

Classes:

<i>SensorResolution</i>	Select the camera sensor resolution: 1280×720, 1280×800, 640×400
-------------------------	--

Methods:

<i>__init__</i> (*args, **kwargs)	Initialize self.
-----------------------------------	------------------

Attributes:

<i>boardSocket</i>
<i>fps</i>
<i>initialControl</i>
<i>resolution</i>

class *SensorResolution*

Bases: *pybind11_builtins.pybind11_object*

Select the camera sensor resolution: 1280×720, 1280×800, 640×400

Members:

THE_720_P

THE_800_P

THE_400_P

Attributes:

<i>THE_400_P</i>
<i>THE_720_P</i>
<i>THE_800_P</i>

continues on next page

Table 87 – continued from previous page

<i>name</i>	
<i>value</i>	
Methods:	
<code>__init__(self, value)</code>	
<code>THE_400_P = <SensorResolution.THE_400_P: 2></code>	
<code>THE_720_P = <SensorResolution.THE_720_P: 0></code>	
<code>THE_800_P = <SensorResolution.THE_800_P: 1></code>	
<code>__init__(self: depthai.MonoCameraProperties.SensorResolution, value: int) → None</code>	
property name	
property value	
<code>__init__(*args, **kwargs)</code>	Initialize self. See help(type(self)) for accurate signature.
property boardSocket	
property fps	
property initialControl	
property resolution	
class <code>depthai.NNData</code>	
Bases: <code>depthai.Buffer</code>	
NNData message. Carries tensors and their metadata	
Methods:	
<code>__init__(self)</code>	Construct NNData message.
<code>getAllLayerNames(self)</code>	returns Names of all layers added
<code>getAllLayers(self)</code>	returns All layers and their information
<code>getFirstLayerFp16(self)</code>	Convenience function to retrieve float values from first layers FP16 tensor
<code>getFirstLayerInt32(self)</code>	Convenience function to retrieve INT32 values from first layers tensor
<code>getFirstLayerUInt8(self)</code>	Convenience function to retrieve U8 data from first layer
<code>getLayer(self, name, tensor)</code>	Retrieve layers tensor information
<code>getLayerDatatype(self, name, datatype)</code>	Retrieve datatype of a layers tensor
<code>getLayerFp16(self, name)</code>	Convenience function to retrieve float values from layers FP16 tensor
<code>getLayerInt32(self, name)</code>	Convenience function to retrieve INT32 values from layers tensor
<code>getLayerUInt8(self, name)</code>	Convenience function to retrieve U8 data from layer

continues on next page

Table 89 – continued from previous page

<code>hasLayer(self, name)</code>	Checks if given layer exists
<code>setLayer(*args, **kwargs)</code>	Overloaded function.

`__init__` (*self*: `depthai.NNData`) → `None`
Construct NNData message.

`getAllLayerNames` (*self*: `depthai.NNData`) → `List[str]`

Returns Names of all layers added

`getAllLayers` (*self*: `depthai.NNData`) → `List[depthai.TensorInfo]`

Returns All layers and their information

`getFirstLayerFp16` (*self*: `depthai.NNData`) → `List[float]`

Convenience function to retrieve float values from first layers FP16 tensor

Returns Float data

`getFirstLayerInt32` (*self*: `depthai.NNData`) → `List[int]`

Convenience function to retrieve INT32 values from first layers tensor

Returns INT32 data

`getFirstLayerUInt8` (*self*: `depthai.NNData`) → `List[int]`

Convenience function to retrieve U8 data from first layer

Returns U8 binary data

`getLayer` (*self*: `depthai.NNData`, *name*: `str`, *tensor*: `depthai.TensorInfo`) → `bool`

Retrieve layers tensor information

Parameter name: Name of the layer

Parameter tensor: Outputs tensor information of that layer

Returns True if layer exists, false otherwise

`getLayerDatatype` (*self*: `depthai.NNData`, *name*: `str`, *datatype*: `depthai.TensorInfo.DataType`) → `bool`

Retrieve datatype of a layers tensor

Parameter name: Name of the layer

Parameter datatype: Datatype of layers tensor

Returns True if layer exists, false otherwise

`getLayerFp16` (*self*: `depthai.NNData`, *name*: `str`) → `List[float]`

Convenience function to retrieve float values from layers FP16 tensor

Parameter name: Name of the layer

Returns Float data

`getLayerInt32` (*self*: `depthai.NNData`, *name*: `str`) → `List[int]`

Convenience function to retrieve INT32 values from layers tensor

Parameter name: Name of the layer

Returns INT32 data

getLayerUInt8 (*self*: depthai.NNData, *name*: str) → List[int]

Convenience function to retrieve U8 data from layer

Parameter name: Name of the layer

Returns U8 binary data

hasLayer (*self*: depthai.NNData, *name*: str) → bool

Checks if given layer exists

Parameter name: Name of the layer

Returns True if layer exists, false otherwise

setLayer (*args, **kwargs)

Overloaded function.

1. setLayer(*self*: depthai.NNData, *name*: str, *data*: numpy.ndarray[numpy.uint8]) -> None

Set a layer with datatype U8.

Parameter name: Name of the layer

Parameter data: Data to store

2. setLayer(*self*: depthai.NNData, *name*: str, *data*: List[int]) -> None

Set a layer with datatype U8. Integers are casted to bytes.

Parameter name: Name of the layer

Parameter data: Data to store

3. setLayer(*self*: depthai.NNData, *name*: str, *data*: List[float]) -> None

Set a layer with datatype FP16. Float values are converted to FP16.

Parameter name: Name of the layer

Parameter data: Data to store

4. setLayer(*self*: depthai.NNData, *name*: str, *data*: List[float]) -> None

Set a layer with datatype FP16. Double values are converted to FP16.

Parameter name: Name of the layer

Parameter data: Data to store

class depthai.NeuralNetwork

Bases: *depthai.Node*

NeuralNetwork node. Runs a neural inference on input data.

Classes:

<i>Properties</i>	alias	of	<i>depthai.</i>
			<i>NeuralNetworkProperties</i>

Methods:

<code>__init__(*args, **kwargs)</code>	Initialize self.
<code>getNumInferenceThreads(self)</code>	How many inference threads will be used to run the network
<code>setBlobPath(self, path)</code>	Load network blob into assets and use once pipeline is started.
<code>setNumInferenceThreads(self, numThreads)</code>	How many threads should the node use to run the network.
<code>setNumNCEPerInferenceThread(self, ...)</code>	How many Neural Compute Engines should a single thread use for inference
<code>setNumPoolFrames(self, numFrames)</code>	Specifies how many frames will be available in the pool

Attributes:

<code>input</code>	Input message with data to be inferred upon Default queue is blocking with size 5
<code>out</code>	Outputs NNData message that carries inference results
<code>passthrough</code>	Passthrough message on which the inference was performed.

Propertiesalias of `depthai.NeuralNetworkProperties` **Methods:**

<code>__init__(*args, **kwargs)</code>	Initialize self.
--	------------------

Attributes:

<code>blobSize</code>
<code>blobUri</code>
<code>numFrames</code>
<code>numNCEPerThread</code>
<code>numThreads</code>

`__init__(*args, **kwargs)`Initialize self. See `help(type(self))` for accurate signature.**getNumInferenceThreads** (*self*: `depthai.NeuralNetwork`) → `int`

How many inference threads will be used to run the network

Returns Number of threads, 0, 1 or 2. Zero means AUTO**property input**

Input message with data to be inferred upon Default queue is blocking with size 5

property out

Outputs NNData message that carries inference results

property passthrough

Passthrough message on which the inference was performed.

Suitable for when input queue is set to non-blocking behavior.

setBlobPath (*self*: `depthai.NeuralNetwork`, *path*: `str`) → `None`

Load network blob into assets and use once pipeline is started.

Throws if file doesn't exist or isn't a valid network blob.

Parameter path: Path to network blob

setNumInferenceThreads (*self*: [depthai.NeuralNetwork](#), *numThreads*: *int*) → *None*

How many threads should the node use to run the network.

Parameter numThreads: Number of threads to dedicate to this node

setNumNCEPerInferenceThread (*self*: [depthai.NeuralNetwork](#), *numNCEPerThread*: *int*) → *None*

How many Neural Compute Engines should a single thread use for inference

Parameter numNCEPerThread: Number of NCE per thread

setNumPoolFrames (*self*: [depthai.NeuralNetwork](#), *numFrames*: *int*) → *None*

Specifies how many frames will be available in the pool

Parameter numFrames: How many frames will pool have

class [depthai.NeuralNetworkProperties](#)

Bases: [pybind11_builtins.pybind11_object](#)

Specify NeuralNetwork options such as blob path, ...

Methods:

[__init__](#) (*args, **kwargs)

Initialize self.

Attributes:

[blobSize](#)

[blobUri](#)

[numFrames](#)

[numNCEPerThread](#)

[numThreads](#)

[__init__](#) (*args, **kwargs)

Initialize self. See `help(type(self))` for accurate signature.

property [blobSize](#)

property [blobUri](#)

property [numFrames](#)

property [numNCEPerThread](#)

property [numThreads](#)

class [depthai.Node](#)

Bases: [pybind11_builtins.pybind11_object](#)

Abstract Node

Classes:

[Connection](#)

Connection between an Input and Output

[Id](#)

Node identifier.

continues on next page

Table 97 – continued from previous page

<i>Input</i>	
<i>Output</i>	

Methods:

<code>__init__(*args, **kwargs)</code>	Initialize self.
<code>getAssets(self)</code>	Retrieves all nodes assets
<code>getInputs(self)</code>	Retrieves all nodes inputs
<code>getName(self)</code>	Retrieves nodes name
<code>getOutputs(self)</code>	Retrieves all nodes outputs

Attributes:

<i>id</i>	Id of node
-----------	------------

class ConnectionBases: `pybind11_builtins.pybind11_object`

Connection between an Input and Output

Methods:

<code>__init__(*args, **kwargs)</code>	Initialize self.
--	------------------

Attributes:

<i>inputId</i>	
<i>inputName</i>	
<i>outputId</i>	
<i>outputName</i>	

`__init__(*args, **kwargs)`
Initialize self. See `help(type(self))` for accurate signature.

property `inputId`**property** `inputName`**property** `outputId`**property** `outputName`**class Id**Bases: `pybind11_builtins.pybind11_object`

Node identificator. Unique for every node on a single Pipeline

Methods:

<code>__init__(*args, **kwargs)</code>	Initialize self.
--	------------------

`__init__(*args, **kwargs)`
Initialize self. See `help(type(self))` for accurate signature.

class Input

Bases: `pybind11_builtins.pybind11_object`

Methods:

<code>__init__(*args, **kwargs)</code>	Initialize self.
<code>getBlocking(self)</code>	Get input queue behavior
<code>getQueueSize(self)</code>	Get input queue size.
<code>setBlocking(self, blocking)</code>	Overrides default input queue behavior.
<code>setQueueSize(self, size)</code>	Overrides default input queue size.

`__init__(*args, **kwargs)`

Initialize self. See `help(type(self))` for accurate signature.

getBlocking (*self*: `depthai.Node.Input`) → `bool`

Get input queue behavior

Returns True blocking, false overwriting

getQueueSize (*self*: `depthai.Node.Input`) → `int`

Get input queue size.

Returns Maximum input queue size

setBlocking (*self*: `depthai.Node.Input`, *blocking*: `bool`) → `None`

Overrides default input queue behavior.

Parameter blocking: True blocking, false overwriting

setQueueSize (*self*: `depthai.Node.Input`, *size*: `int`) → `None`

Overrides default input queue size. If queue size fills up, behavior depends on *blocking* attribute

Parameter size: Maximum input queue size

class Output

Bases: `pybind11_builtins.pybind11_object`

Methods:

<code>__init__(*args, **kwargs)</code>	Initialize self.
<code>canConnect(self, in)</code>	Check if connection is possible
<code>getConnections(self)</code>	Retrieve all connections from this output
<code>link(self, in)</code>	Link current output to input.
<code>unlink(self, in)</code>	Unlink a previously linked connection

`__init__(*args, **kwargs)`

Initialize self. See `help(type(self))` for accurate signature.

canConnect (*self*: `depthai.Node.Output`, *in*: `depthai.Node.Input`) → `bool`

Check if connection is possible

Parameter in: Input to connect to

Returns True if connection is possible, false otherwise

getConnections (*self*: `depthai.Node.Output`) → `List[dai::Node::Connection]`

Retrieve all connections from this output

Returns Vector of connections

link (*self*: `depthai.Node.Output`, *in*: `depthai.Node.Input`) → `None`

Link current output to input.

Throws an error if this output cannot be linked to given input, or if they are already linked

Parameter in: Input to link to

unlink (*self*: *depthai.Node.Output*, *in*: *depthai.Node.Input*) → *None*

Unlink a previously linked connection

Throws an error if not linked.

Parameter in: Input from which to unlink from

__init__ (**args*, ***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

getAssets (*self*: *depthai.Node*) → List[*depthai.Asset*]

Retrieves all nodes assets

getInputs (*self*: *depthai.Node*) → List[dai::Node::Input]

Retrieves all nodes inputs

getName (*self*: *depthai.Node*) → *str*

Retrieves nodes name

getOutputs (*self*: *depthai.Node*) → List[dai::Node::Output]

Retrieves all nodes outputs

property id

Id of node

class *depthai.ObjectTracker*

Bases: *depthai.Node*

ObjectTracker node. Performs object tracking using Kalman filter and hungarian algorithm.

Classes:

<i>Properties</i>	alias	of	<i>depthai.</i>
	<i>ObjectTrackerProperties</i>		

Methods:

<i>__init__</i> (<i>*args</i> , <i>**kwargs</i>)	Initialize self.
<i>setDetectionLabelsToTrack</i> (<i>self</i> , <i>labels</i>)	Specify detection labels to track.
<i>setMaxObjectsToTrack</i> (<i>self</i> , <i>maxObjectsToTrack</i>)	Specify maximum number of object to track.
<i>setTrackerIdAssignmentPolicy</i> (<i>self</i> , <i>type</i>)	Specify tracker ID assignment policy.
<i>setTrackerThreshold</i> (<i>self</i> , <i>threshold</i>)	Specify tracker threshold.
<i>setTrackerType</i> (<i>self</i> , <i>type</i>)	Specify tracker type algorithm.

Attributes:

<i>inputDetectionFrame</i>	Input ImgFrame message on which object detection was performed.
<i>inputDetections</i>	Input message with image detection from neural network.
<i>inputTrackerFrame</i>	Input ImgFrame message on which tracking will be performed.
<i>out</i>	Outputs Tracklets message that carries object tracking results.
<i>passthroughDetectionFrame</i>	Passthrough ImgFrame message on which object detection was performed.

continues on next page

Table 107 – continued from previous page

<i>passthroughDetections</i>	Passthrough image detections message from neural network output.
<i>passthroughTrackerFrame</i>	Passthrough ImgFrame message on which tracking was performed.

Propertiesalias of *depthai.ObjectTrackerProperties* **Methods:**

<code>__init__(*args, **kwargs)</code>	Initialize self.
--	------------------

Attributes:

<code>detectionLabelsToTrack</code>
<code>maxObjectsToTrack</code>
<code>trackerIdAssignmentPolicy</code>
<code>trackerThreshold</code>
<code>trackerType</code>

`__init__(*args, **kwargs)`
Initialize self. See help(type(self)) for accurate signature.

property inputDetectionFrame

Input ImgFrame message on which object detection was performed. Default queue is non-blocking with size 4.

property inputDetections

Input message with image detection from neural network. Default queue is non-blocking with size 4.

property inputTrackerFrame

Input ImgFrame message on which tracking will be performed. RGBp, BGRp, NV12, YUV420p types are supported. Default queue is non-blocking with size 4.

property out

Outputs Tracklets message that carries object tracking results.

property passthroughDetectionFrame

Passthrough ImgFrame message on which object detection was performed. Suitable for when input queue is set to non-blocking behavior.

property passthroughDetections

Passthrough image detections message from neural network output. Suitable for when input queue is set to non-blocking behavior.

property passthroughTrackerFrame

Passthrough ImgFrame message on which tracking was performed. Suitable for when input queue is set to non-blocking behavior.

setDetectionLabelsToTrack (*self*: *depthai.ObjectTracker*, *labels*: *List[int]*) → *None*

Specify detection labels to track.

Parameter labels: Detection labels to track. Default every label is tracked from image detection network output.

setMaxObjectsToTrack (*self*: *depthai.ObjectTracker*, *maxObjectsToTrack*: *int*) → *None*

Specify maximum number of object to track.

Parameter maxObjectsToTrack: Maximum number of object to track. Maximum 60.

setTrackerIdAssignmentPolicy (*self*: *depthai.ObjectTracker*, *type*: *dai::TrackerIdAssignmentPolicy*) → *None*
Specify tracker ID assignment policy.

Parameter type: Tracker ID assignment policy.

setTrackerThreshold (*self*: *depthai.ObjectTracker*, *threshold*: *float*) → *None*
Specify tracker threshold.

Parameter threshold: Above this threshold the detected objects will be tracked. Default 0, all image detections are tracked.

setTrackerType (*self*: *depthai.ObjectTracker*, *type*: *dai::TrackerType*) → *None*
Specify tracker type algorithm.

Parameter type: Tracker type.

class *depthai.ObjectTrackerProperties*

Bases: *pybind11_builtins.pybind11_object*

Properties for ObjectTracker

Methods:

<code>__init__(*args, **kwargs)</code>	Initialize self.
--	------------------

Attributes:

<code>detectionLabelsToTrack</code>
<code>maxObjectsToTrack</code>
<code>trackerIdAssignmentPolicy</code>
<code>trackerThreshold</code>
<code>trackerType</code>

`__init__(*args, **kwargs)`
Initialize self. See `help(type(self))` for accurate signature.

property `detectionLabelsToTrack`

property `maxObjectsToTrack`

property `trackerIdAssignmentPolicy`

property `trackerThreshold`

property `trackerType`

class *depthai.OpenVINO*

Bases: *pybind11_builtins.pybind11_object*

Support for basic OpenVINO related actions like version identification of neural network blobs,...

Attributes:

<code>VERSION_2020_1</code>
<code>VERSION_2020_2</code>
<code>VERSION_2020_3</code>
<code>VERSION_2020_4</code>
<code>VERSION_2021_1</code>

continues on next page

Table 112 – continued from previous page

<code>VERSION_2021_2</code>
<code>VERSION_2021_3</code>

Classes:

<code>Version</code>	OpenVINO Version supported version information
----------------------	--

Methods:

<code>__init__(*args, **kwargs)</code>	Initialize self.
<code>areVersionsBlobCompatible(v1, v2)</code>	Checks whether two blob versions are compatible
<code>getBlobLatestSupportedVersion(majorVersion, ...)</code>	Returns latest potentially supported version by a given blob version.
<code>getBlobSupportedVersions(majorVersion, ...)</code>	Returns a list of potentially supported versions for a specified blob major and minor versions.
<code>getVersionName(version)</code>	Returns string representation of a given version
<code>getVersions()</code>	returns Supported versions
<code>parseVersionName(versionString)</code>	Creates Version from string representation.

```
VERSION_2020_1 = <Version.VERSION_2020_1: 0>
```

```
VERSION_2020_2 = <Version.VERSION_2020_2: 1>
```

```
VERSION_2020_3 = <Version.VERSION_2020_3: 2>
```

```
VERSION_2020_4 = <Version.VERSION_2020_4: 3>
```

```
VERSION_2021_1 = <Version.VERSION_2021_1: 4>
```

```
VERSION_2021_2 = <Version.VERSION_2021_2: 5>
```

```
VERSION_2021_3 = <Version.VERSION_2021_3: 6>
```

```
class Version
```

```
    Bases: pybind11_builtins.pybind11_object
```

```
    OpenVINO Version supported version information
```

```
    Members:
```

```
        VERSION_2020_1
```

```
        VERSION_2020_2
```

```
        VERSION_2020_3
```

```
        VERSION_2020_4
```

```
        VERSION_2021_1
```

```
        VERSION_2021_2
```

```
        VERSION_2021_3
```

```
    Attributes:
```

<code>VERSION_2020_1</code>
<code>VERSION_2020_2</code>
<code>VERSION_2020_3</code>
<code>VERSION_2020_4</code>
<code>VERSION_2021_1</code>
<code>VERSION_2021_2</code>
<code>VERSION_2021_3</code>
<code>name</code>
<code>value</code>

Methods:

<code>__init__(self, value)</code>

```
VERSION_2020_1 = <Version.VERSION_2020_1: 0>
```

```
VERSION_2020_2 = <Version.VERSION_2020_2: 1>
```

```
VERSION_2020_3 = <Version.VERSION_2020_3: 2>
```

```
VERSION_2020_4 = <Version.VERSION_2020_4: 3>
```

```
VERSION_2021_1 = <Version.VERSION_2021_1: 4>
```

```
VERSION_2021_2 = <Version.VERSION_2021_2: 5>
```

```
VERSION_2021_3 = <Version.VERSION_2021_3: 6>
```

```
__init__(self: depthai.OpenVINO.Version, value: int) → None
```

property name

property value

```
__init__( *args, **kwargs)
```

Initialize self. See help(type(self)) for accurate signature.

```
static areVersionsBlobCompatible (v1: dai::OpenVINO::Version, v2:
dai::OpenVINO::Version) → bool
```

Checks whether two blob versions are compatible

```
OpenVINO.getBlobLatestSupportedVersion (majorVersion: int, minorVersion: int) -> dai::OpenVINO::Version
```

Returns latest potentially supported version by a given blob version.

Parameter majorVersion: Major version from OpenVINO blob

Parameter minorVersion: Minor version from OpenVINO blob

Returns Latest potentially supported version

```
OpenVINO.getBlobSupportedVersions (majorVersion: int, minorVersion: int) -> List[dai::OpenVINO::Version]
```

Returns a list of potentially supported versions for a specified blob major and minor versions.

Parameter majorVersion: Major version from OpenVINO blob

Parameter minorVersion: Minor version from OpenVINO blob

Returns Vector of potentially supported versions

```
static getVersionName (version: dai::OpenVINO::Version) → str
```

Returns string representation of a given version

Parameter version: OpenVINO version

Returns Name of a given version

static `getVersions ()` → List[dai::OpenVINO::Version]

Returns Supported versions

static `parseVersionName (versionString: str)` → dai::OpenVINO::Version
Creates Version from string representation. Throws if not possible.

Parameter versionString: Version as string

Returns Version object if successful

class `depthai.Pipeline`

Bases: `pybind11_builtins.pybind11_object`

Represents the pipeline, set of nodes and connections between them

Methods:

<code>__init__(self)</code>	Constructs a new pipeline
<code>createColorCamera(self)</code>	
<code>createImageManip(self)</code>	
<code>createMobileNetDetectionNetwork(self)</code>	
<code>createMobileNetSpatialDetectionNetwork(self)</code>	
<code>createMonoCamera(self)</code>	
<code>createNeuralNetwork(self)</code>	
<code>createObjectTracker(self)</code>	
<code>createSPIOut(self)</code>	
<code>createSpatialLocationCalculator(self)</code>	
<code>createStereoDepth(self)</code>	
<code>createSystemLogger(self)</code>	
<code>createVideoEncoder(self)</code>	
<code>createXLinkIn(self)</code>	
<code>createXLinkOut(self)</code>	
<code>createYoloDetectionNetwork(self)</code>	
<code>createYoloSpatialDetectionNetwork(self)</code>	
<code>getAllAssets(self)</code>	Get assets on the pipeline includes nodes assets
<code>getAllNodes(*args, **kwargs)</code>	Overloaded function.
<code>getAssetManager(*args, **kwargs)</code>	Overloaded function.
<code>getConnectionMap(self)</code>	Get a reference to internal connection representation
<code>getConnections(self)</code>	Get all connections
<code>getGlobalProperties(self)</code>	returns Global properties of current pipeline
<code>getNode(*args, **kwargs)</code>	Overloaded function.
<code>getNodeMap(self)</code>	Get a reference to internal node map
<code>link(self, arg0, arg1)</code>	Link output to an input.
<code>remove(self, node)</code>	Removes a node from pipeline
<code>setOpenVINOVersion(self, version)</code>	Set a specific OpenVINO version to use with this pipeline

continues on next page

Table 117 – continued from previous page

<i>unlink</i> (self, arg0, arg1)	Unlink output from an input.
<hr/>	
__init__ (self: <i>depthai.Pipeline</i>) → None	
Constructs a new pipeline	
createColorCamera (self: <i>depthai.Pipeline</i>) → <i>depthai.ColorCamera</i>	
createImageManip (self: <i>depthai.Pipeline</i>) → <i>depthai.ImageManip</i>	
createMobileNetDetectionNetwork (self: <i>depthai.Pipeline</i>) → <i>depthai.MobileNetDetectionNetwork</i>	
createMobileNetSpatialDetectionNetwork (self: <i>depthai.Pipeline</i>) → <i>depthai.MobileNetSpatialDetectionNetwork</i>	
createMonoCamera (self: <i>depthai.Pipeline</i>) → <i>depthai.MonoCamera</i>	
createNeuralNetwork (self: <i>depthai.Pipeline</i>) → <i>depthai.NeuralNetwork</i>	
createObjectTracker (self: <i>depthai.Pipeline</i>) → <i>depthai.ObjectTracker</i>	
createSPIOut (self: <i>depthai.Pipeline</i>) → <i>depthai.SPIOut</i>	
createSpatialLocationCalculator (self: <i>depthai.Pipeline</i>) → <i>depthai.SpatialLocationCalculator</i>	
createStereoDepth (self: <i>depthai.Pipeline</i>) → <i>depthai.StereoDepth</i>	
createSystemLogger (self: <i>depthai.Pipeline</i>) → <i>depthai.SystemLogger</i>	
createVideoEncoder (self: <i>depthai.Pipeline</i>) → <i>depthai.VideoEncoder</i>	
createXLinkIn (self: <i>depthai.Pipeline</i>) → <i>depthai.XLinkIn</i>	
createXLinkOut (self: <i>depthai.Pipeline</i>) → <i>depthai.XLinkOut</i>	
createYoloDetectionNetwork (self: <i>depthai.Pipeline</i>) → <i>depthai.YoloDetectionNetwork</i>	
createYoloSpatialDetectionNetwork (self: <i>depthai.Pipeline</i>) → <i>depthai.YoloSpatialDetectionNetwork</i>	
getAllAssets (self: <i>depthai.Pipeline</i>) → <i>depthai.AssetManager</i>	
Get assets on the pipeline includes nodes assets	
getAllNodes (*args, **kwargs)	
Overloaded function.	
1. <i>getAllNodes</i> (self: <i>depthai.Pipeline</i>) -> List[<i>depthai.Node</i>]	
Get a vector of all nodes	
2. <i>getAllNodes</i> (self: <i>depthai.Pipeline</i>) -> List[<i>depthai.Node</i>]	
Get a vector of all nodes	
getAssetManager (*args, **kwargs)	
Overloaded function.	
1. <i>getAssetManager</i> (self: <i>depthai.Pipeline</i>) -> <i>depthai.AssetManager</i>	
Get pipelines AssetManager as reference	
2. <i>getAssetManager</i> (self: <i>depthai.Pipeline</i>) -> <i>depthai.AssetManager</i>	
Get pipelines AssetManager as reference	

getConnectionMap (*self*: [depthai.Pipeline](#)) → Dict[int, Set[*depthai.Node.Connection*]]

Get a reference to internal connection representation

getConnections (*self*: [depthai.Pipeline](#)) → List[*depthai.Node.Connection*]

Get all connections

getGlobalProperties (*self*: [depthai.Pipeline](#)) → *depthai.GlobalProperties*

Returns Global properties of current pipeline

getNode (*args, **kwargs)

Overloaded function.

1. `getNode(self: depthai.Pipeline, arg0: int) -> depthai.Node`

Get node with id if it exists, nullptr otherwise

2. `getNode(self: depthai.Pipeline, arg0: int) -> depthai.Node`

Get node with id if it exists, nullptr otherwise

getNodeMap (*self*: [depthai.Pipeline](#)) → Dict[int, *depthai.Node*]

Get a reference to internal node map

link (*self*: [depthai.Pipeline](#), *arg0*: *depthai.Node.Output*, *arg1*: *depthai.Node.Input*) → None

Link output to an input. Both nodes must be on the same pipeline

Throws an error if they aren't or cannot be connected

Parameter out: Nodes output to connect from

Parameter in: Nodes input to connect to

remove (*self*: [depthai.Pipeline](#), *node*: *depthai.Node*) → None

Removes a node from pipeline

setOpenVINOVersion (*self*: *depthai.Pipeline*, *version*: *depthai.OpenVINO.Version* = *<Version.VERSION_2021_3: 6>*) → None

Set a specific OpenVINO version to use with this pipeline

unlink (*self*: [depthai.Pipeline](#), *arg0*: *depthai.Node.Output*, *arg1*: *depthai.Node.Input*) → None

Unlink output from an input.

Throws an error if link doesn't exists

Parameter out: Nodes output to unlink from

Parameter in: Nodes input to unlink to

class [depthai.Point2f](#)

Bases: [pybind11_builtins.pybind11_object](#)

Point2f structure

x and y coordinates that define a 2D point.

Methods:

`__init__`(*args, **kwargs)

Overloaded function.

Attributes:

x

y

```
__init__ (*args, **kwargs)
```

Overloaded function.

1. `__init__(self: depthai.Point2f) -> None`
2. `__init__(self: depthai.Point2f, arg0: float, arg1: float) -> None`

property x

property y

class `depthai.Point3f`

Bases: `pybind11_builtins.pybind11_object`

Point3f structure

x,y,z coordinates that define a 3D point.

Methods:

```
__init__ (*args, **kwargs)
```

Overloaded function.

Attributes:

`x`

`y`

`z`

```
__init__ (*args, **kwargs)
```

Overloaded function.

1. `__init__(self: depthai.Point3f) -> None`
2. `__init__(self: depthai.Point3f, arg0: float, arg1: float, arg2: float) -> None`

property x

property y

property z

class `depthai.RawBuffer`

Bases: `pybind11_builtins.pybind11_object`

Methods:

```
__init__ (self)
```

Attributes:

`data`

```
__init__ (self: depthai.RawBuffer) -> None
```

property data

class `depthai.RawCameraControl`

Bases: `depthai.RawBuffer`

Classes:

<i>AntiBandingMode</i>	Members:
<i>AutoFocusMode</i>	Members:
<i>AutoWhiteBalanceMode</i>	Members:
<i>EffectMode</i>	Members:
<i>SceneMode</i>	Members:

Methods:

<code>__init__(*args, **kwargs)</code>	Initialize self.
--	------------------

Attributes:

<i>autoFocusMode</i>
<i>cmdMask</i>
<i>lensPosition</i>

class AntiBandingModeBases: `pybind11_builtins.pybind11_object`

Members:

OFF

MAINS_50_HZ

MAINS_60_HZ

AUTO

Attributes:

<i>AUTO</i>
<i>MAINS_50_HZ</i>
<i>MAINS_60_HZ</i>
<i>OFF</i>
<i>name</i>
<i>value</i>

Methods:

<code>__init__(self, value)</code>

AUTO = <AntiBandingMode.AUTO: 3>**MAINS_50_HZ = <AntiBandingMode.MAINS_50_HZ: 1>****MAINS_60_HZ = <AntiBandingMode.MAINS_60_HZ: 2>****OFF = <AntiBandingMode.OFF: 0>**`__init__(self: depthai.RawCameraControl.AntiBandingMode, value: int) → None`**property name****property value**

class AutoFocusModeBases: `pybind11_builtins.pybind11_object`

Members:

OFF

AUTO

MACRO

CONTINUOUS_VIDEO

CONTINUOUS_PICTURE

EDOF

Attributes:

<i>AUTO</i>
<i>CONTINUOUS_PICTURE</i>
<i>CONTINUOUS_VIDEO</i>
<i>EDOF</i>
<i>MACRO</i>
<i>OFF</i>
<i>name</i>
<i>value</i>

Methods:

<i>__init__</i> (self, value)

AUTO = `<AutoFocusMode.AUTO: 1>`**CONTINUOUS_PICTURE** = `<AutoFocusMode.CONTINUOUS_PICTURE: 4>`**CONTINUOUS_VIDEO** = `<AutoFocusMode.CONTINUOUS_VIDEO: 3>`**EDOF** = `<AutoFocusMode.EDOF: 5>`**MACRO** = `<AutoFocusMode.MACRO: 2>`**OFF** = `<AutoFocusMode.OFF: 0>`*__init__*(self: `depthai.RawCameraControl.AutoFocusMode`, value: `int`) → `None`**property name****property value****class AutoWhiteBalanceMode**Bases: `pybind11_builtins.pybind11_object`

Members:

OFF

AUTO

INCANDESCENT

FLUORESCENT

WARM_FLUORESCENT

DAYLIGHT
 CLOUDY_DAYLIGHT
 TWILIGHT
 SHADE

Attributes:

<i>AUTO</i>
<i>CLOUDY_DAYLIGHT</i>
<i>DAYLIGHT</i>
<i>FLUORESCENT</i>
<i>INCANDESCENT</i>
<i>OFF</i>
<i>SHADE</i>
<i>TWILIGHT</i>
<i>WARM_FLUORESCENT</i>
<i>name</i>
<i>value</i>

Methods:

<i>__init__</i> (self, value)

```

AUTO = <AutoWhiteBalanceMode.AUTO: 1>
CLOUDY_DAYLIGHT = <AutoWhiteBalanceMode.CLOUDY_DAYLIGHT: 6>
DAYLIGHT = <AutoWhiteBalanceMode.DAYLIGHT: 5>
FLUORESCENT = <AutoWhiteBalanceMode.FLUORESCENT: 3>
INCANDESCENT = <AutoWhiteBalanceMode.INCANDESCENT: 2>
OFF = <AutoWhiteBalanceMode.OFF: 0>
SHADE = <AutoWhiteBalanceMode.SHADE: 8>
TWILIGHT = <AutoWhiteBalanceMode.TWILIGHT: 7>
WARM_FLUORESCENT = <AutoWhiteBalanceMode.WARM_FLUORESCENT: 4>
__init__(self: depthai.RawCameraControl.AutoWhiteBalanceMode, value: int) → None
property name
property value

```

class EffectMode

Bases: `pybind11_builtins.pybind11_object`

Members:

OFF
 MONO
 NEGATIVE
 SOLARIZE

SEPIA

POSTERIZE

WHITEBOARD

BLACKBOARD

AQUA

Attributes:

<i>AQUA</i>
<i>BLACKBOARD</i>
<i>MONO</i>
<i>NEGATIVE</i>
<i>OFF</i>
<i>POSTERIZE</i>
<i>SEPIA</i>
<i>SOLARIZE</i>
<i>WHITEBOARD</i>
<i>name</i>
<i>value</i>

Methods:

<code>__init__(self, value)</code>

AQUA = <EffectMode.AQUA: 8>

BLACKBOARD = <EffectMode.BLACKBOARD: 7>

MONO = <EffectMode.MONO: 1>

NEGATIVE = <EffectMode.NEGATIVE: 2>

OFF = <EffectMode.OFF: 0>

POSTERIZE = <EffectMode.POSTERIZE: 5>

SEPIA = <EffectMode.SEPIA: 4>

SOLARIZE = <EffectMode.SOLARIZE: 3>

WHITEBOARD = <EffectMode.WHITEBOARD: 6>

`__init__(self: depthai.RawCameraControl.EffectMode, value: int) → None`

property name

property value

class SceneMode

Bases: `pybind11_builtins.pybind11_object`

Members:

UNSUPPORTED

FACE_PRIORITY

ACTION

PORTRAIT
 LANDSCAPE
 NIGHT
 NIGHT_PORTRAIT
 THEATRE
 BEACH
 SNOW
 SUNSET
 STEADYPHOTO
 FIREWORKS
 SPORTS
 PARTY
 CANDLELIGHT
 BARCODE

Attributes:

<i>ACTION</i>
<i>BARCODE</i>
<i>BEACH</i>
<i>CANDLELIGHT</i>
<i>FACE_PRIORITY</i>
<i>FIREWORKS</i>
<i>LANDSCAPE</i>
<i>NIGHT</i>
<i>NIGHT_PORTRAIT</i>
<i>PARTY</i>
<i>PORTRAIT</i>
<i>SNOW</i>
<i>SPORTS</i>
<i>STEADYPHOTO</i>
<i>SUNSET</i>
<i>THEATRE</i>
<i>UNSUPPORTED</i>
<i>name</i>
<i>value</i>

Methods:

<code>__init__(self, value)</code>

ACTION = <SceneMode.ACTION: 2>

BARCODE = <SceneMode.BARCODE: 16>

BEACH = <SceneMode.BEACH: 8>

CANDLELIGHT = <SceneMode.CANDLELIGHT: 15>

```
FACE_PRIORITY = <SceneMode.FACE_PRIORITY: 1>
FIREWORKS = <SceneMode.FIREWORKS: 12>
LANDSCAPE = <SceneMode.LANDSCAPE: 4>
NIGHT = <SceneMode.NIGHT: 5>
NIGHT_PORTRAIT = <SceneMode.NIGHT_PORTRAIT: 6>
PARTY = <SceneMode.PARTY: 14>
PORTRAIT = <SceneMode.PORTRAIT: 3>
SNOW = <SceneMode.SNOW: 9>
SPORTS = <SceneMode.SPORTS: 13>
STEADYPHOTO = <SceneMode.STEADYPHOTO: 11>
SUNSET = <SceneMode.SUNSET: 10>
THEATRE = <SceneMode.THEATRE: 7>
UNSUPPORTED = <SceneMode.UNSUPPORTED: 0>

__init__(self: depthai.RawCameraControl.SceneMode, value: int) → None
property name
property value

__init__(*args, **kwargs)
    Initialize self. See help(type(self)) for accurate signature.

property autoFocusMode
property cmdMask
property lensPosition
```

```
class depthai.RawImageManipConfig
```

```
    Bases: depthai.RawBuffer
```

Classes:

CropConfig

CropRect

FormatConfig

ResizeConfig

Methods:

__init__(self)

Attributes:

cropConfig

enableCrop

enableFormat

enableResize

formatConfig

continues on next page

Table 139 – continued from previous page

*resizeConfig***class CropConfig**Bases: `pybind11_builtins.pybind11_object`**Methods:***`__init__`*(self)**Attributes:***`cropRatio`**`cropRect`**`cropRotatedRect`**`enableCenterCropRectangle`**`enableRotatedRect`**`normalizedCoords`**`widthHeightAspectRatio`**`__init__`* (self: `depthai.RawImageManipConfig.CropConfig`) → None**property** `cropRatio`**property** `cropRect`**property** `cropRotatedRect`**property** `enableCenterCropRectangle`**property** `enableRotatedRect`**property** `normalizedCoords`**property** `widthHeightAspectRatio`**class CropRect**Bases: `pybind11_builtins.pybind11_object`**Methods:***`__init__`*(self)**Attributes:***`xmax`**`xmin`**`ymax`**`ymin`**`__init__`* (self: `depthai.RawImageManipConfig.CropRect`) → None**property** `xmax`**property** `xmin`**property** `ymax`

property ymin

class FormatConfig

Bases: pybind11_builtins.pybind11_object

Methods:

`__init__(self)`

Attributes:

`flipHorizontal`

`type`

`__init__(self: depthai.RawImageManipConfig.FormatConfig) → None`

property flipHorizontal

property type

class ResizeConfig

Bases: pybind11_builtins.pybind11_object

Methods:

`__init__(self)`

Attributes:

`bgBlue`

`bgGreen`

`bgRed`

`enableRotation`

`enableWarp4pt`

`enableWarpMatrix`

`height`

`keepAspectRatio`

`lockAspectRatioFill`

`normalizedCoords`

`rotationAngleDeg`

`warpBorderReplicate`

`warpFourPoints`

`warpMatrix3x3`

`width`

`__init__(self: depthai.RawImageManipConfig.ResizeConfig) → None`

property bgBlue

property bgGreen

property bgRed

property enableRotation

property enableWarp4pt

```

property enableWarpMatrix
property height
property keepAspectRatio
property lockAspectRatioFill
property normalizedCoords
property rotationAngleDeg
property warpBorderReplicate
property warpFourPoints
property warpMatrix3x3
property width

```

```
__init__(self: depthai.RawImageManipConfig) → None
```

```

property cropConfig
property enableCrop
property enableFormat
property enableResize
property formatConfig
property resizeConfig

```

class `depthai.RawImgDetections`

Bases: `depthai.RawBuffer`

Methods:

```
__init__(self)
```

Attributes:

```
detections
```

```
__init__(self: depthai.RawImgDetections) → None
```

```
property detections
```

class `depthai.RawImgFrame`

Bases: `depthai.RawBuffer`

Classes:

```
Specs
```

```
Type
```

Members:

Methods:

```
__init__(self)
```

Attributes:

category

fb

instanceNum

sequenceNum

ts

class SpecsBases: `pybind11_builtins.pybind11_object`**Methods:**

`__init__(*args, **kwargs)` Initialize self.

Attributes:

bytesPP

height

p1Offset

p2Offset

p3Offset

stride

type

width

`__init__(*args, **kwargs)`
Initialize self. See `help(type(self))` for accurate signature.**property** `bytesPP`**property** `height`**property** `p1Offset`**property** `p2Offset`**property** `p3Offset`**property** `stride`**property** `type`**property** `width`**class Type**Bases: `pybind11_builtins.pybind11_object`**Members:**`YUV422i``YUV444p``YUV420p``YUV422p``YUV400p``RGBA8888`

RGB161616
RGB888p
BGR888p
RGB888i
BGR888i
RGBF16F16F16p
BGRF16F16F16p
RGBF16F16F16i
BGRF16F16F16i
GRAY8
GRAYF16
LUT2
LUT4
LUT16
RAW16
RAW14
RAW12
RAW10
RAW8
PACK10
PACK12
YUV444i
NV12
NV21
BITSTREAM
HDR
NONE

Attributes:

<i>BGR888i</i>
<i>BGR888p</i>
<i>BGRF16F16F16i</i>
<i>BGRF16F16F16p</i>
<i>BITSTREAM</i>
<i>GRAY8</i>
<i>GRAYF16</i>
<i>HDR</i>
<i>LUT16</i>
<i>LUT2</i>

continues on next page

Table 155 – continued from previous page

<i>LUT4</i>
<i>NONE</i>
<i>NV12</i>
<i>NV21</i>
<i>PACK10</i>
<i>PACK12</i>
<i>RAW10</i>
<i>RAW12</i>
<i>RAW14</i>
<i>RAW16</i>
<i>RAW8</i>
<i>RGB161616</i>
<i>RGB888i</i>
<i>RGB888p</i>
<i>RGBA8888</i>
<i>RGBF16F16F16i</i>
<i>RGBF16F16F16p</i>
<i>YUV400p</i>
<i>YUV420p</i>
<i>YUV422i</i>
<i>YUV422p</i>
<i>YUV444i</i>
<i>YUV444p</i>
<i>name</i>
<i>value</i>

Methods:

<code>__init__(self, value)</code>

```

BGR888i = <Type.BGR888i: 10>
BGR888p = <Type.BGR888p: 8>
BGRF16F16F16i = <Type.BGRF16F16F16i: 14>
BGRF16F16F16p = <Type.BGRF16F16F16p: 12>
BITSTREAM = <Type.BITSTREAM: 30>
GRAY8 = <Type.GRAY8: 15>
GRAYF16 = <Type.GRAYF16: 16>
HDR = <Type.HDR: 31>
LUT16 = <Type.LUT16: 19>
LUT2 = <Type.LUT2: 17>
LUT4 = <Type.LUT4: 18>
NONE = <Type.NONE: 32>
NV12 = <Type.NV12: 28>
NV21 = <Type.NV21: 29>

```

```

PACK10 = <Type.PACK10: 25>
PACK12 = <Type.PACK12: 26>
RAW10 = <Type.RAW10: 23>
RAW12 = <Type.RAW12: 22>
RAW14 = <Type.RAW14: 21>
RAW16 = <Type.RAW16: 20>
RAW8 = <Type.RAW8: 24>
RGB161616 = <Type.RGB161616: 6>
RGB888i = <Type.RGB888i: 9>
RGB888p = <Type.RGB888p: 7>
RGBA8888 = <Type.RGBA8888: 5>
RGBF16F16F16i = <Type.RGBF16F16F16i: 13>
RGBF16F16F16p = <Type.RGBF16F16F16p: 11>
YUV400p = <Type.YUV400p: 4>
YUV420p = <Type.YUV420p: 2>
YUV422i = <Type.YUV422i: 0>
YUV422p = <Type.YUV422p: 3>
YUV444i = <Type.YUV444i: 27>
YUV444p = <Type.YUV444p: 1>

__init__(self: depthai.RawImgFrame.Type, value: int) → None
property name
property value

__init__(self: depthai.RawImgFrame) → None
property category
property fb
property instanceNum
property sequenceNum
property ts

```

class `depthai.RawNNData`
 Bases: `depthai.RawBuffer`
 Methods:

```
__init__(self)
```

Attributes:

```
batchSize
tensors
```

```
__init__(self: depthai.RawNNData) → None
```

```
property batchSize
```

```
property tensors
```

```
class depthai.RawSpatialImgDetections
```

```
Bases: depthai.RawBuffer
```

```
Methods:
```

```
__init__(self)
```

```
Attributes:
```

```
detections
```

```
__init__(self: depthai.RawSpatialImgDetections) → None
```

```
property detections
```

```
class depthai.RawSystemInformation
```

```
Bases: depthai.RawBuffer
```

```
System information of device
```

```
Memory usage, cpu usage and chip temperature
```

```
Methods:
```

```
__init__(self)
```

```
Attributes:
```

```
chipTemperature
```

```
cmxMemoryUsage
```

```
ddrMemoryUsage
```

```
leonCssCpuUsage
```

```
leonCssMemoryUsage
```

```
leonMssCpuUsage
```

```
leonMssMemoryUsage
```

```
__init__(self: depthai.RawSystemInformation) → None
```

```
property chipTemperature
```

```
property cmxMemoryUsage
```

```
property ddrMemoryUsage
```

```
property leonCssCpuUsage
```

```
property leonCssMemoryUsage
```

```
property leonMssCpuUsage
```

```
property leonMssMemoryUsage
```

```
class depthai.RawTracklets
```

```
Bases: depthai.RawBuffer
```

Methods:

<code>__init__(*args, **kwargs)</code>	Initialize self.
--	------------------

Attributes:

<code>tracklets</code>

`__init__(*args, **kwargs)`
Initialize self. See help(type(self)) for accurate signature.

property tracklets**class depthai.Rect**

Bases: `pybind11_builtins.pybind11_object`

Rect structure

x,y coordinates together with width and height that define a rectangle. Can be either normalized [0,1] or absolute representation.

Methods:

<code>__init__(*args, **kwargs)</code>	Overloaded function.
<code>area(self)</code>	Area (width*height) of the rectangle
<code>bottomRight(self)</code>	The bottom-right corner
<code>contains(self, arg0)</code>	Checks whether the rectangle contains the point.
<code>denormalize(self, width, height)</code>	Denormalize rectangle.
<code>empty(self)</code>	True if rectangle is empty.
<code>isNormalized(self)</code>	Whether rectangle is normalized (coordinates in [0,1] range) or not.
<code>normalize(self, width, height)</code>	Normalize rectangle.
<code>size(self)</code>	Size (width, height) of the rectangle
<code>topLeft(self)</code>	The top-left corner.

Attributes:

<code>height</code>
<code>width</code>
<code>x</code>
<code>y</code>

`__init__(*args, **kwargs)`
Overloaded function.

1. `__init__(self: depthai.Rect) -> None`
2. `__init__(self: depthai.Rect, arg0: float, arg1: float, arg2: float, arg3: float) -> None`
3. `__init__(self: depthai.Rect, arg0: depthai.Point2f, arg1: depthai.Point2f) -> None`
4. `__init__(self: depthai.Rect, arg0: depthai.Point2f, arg1: depthai.Size2f) -> None`

area (*self*: `depthai.Rect`) \rightarrow `float`
Area (width*height) of the rectangle

bottomRight (*self*: [depthai.Rect](#)) → [depthai.Point2f](#)

The bottom-right corner

contains (*self*: [depthai.Rect](#), *arg0*: [depthai.Point2f](#)) → bool

Checks whether the rectangle contains the point.

denormalize (*self*: [depthai.Rect](#), *width*: int, *height*: int) → [depthai.Rect](#)

Denormalize rectangle.

Parameter width: Destination frame width.

Parameter height: Destination frame height.

empty (*self*: [depthai.Rect](#)) → bool

True if rectangle is empty.

property height

isNormalized (*self*: [depthai.Rect](#)) → bool

Whether rectangle is normalized (coordinates in [0,1] range) or not.

normalize (*self*: [depthai.Rect](#), *width*: int, *height*: int) → [depthai.Rect](#)

Normalize rectangle.

Parameter width: Source frame width.

Parameter height: Source frame height.

size (*self*: [depthai.Rect](#)) → [depthai.Size2f](#)

Size (width, height) of the rectangle

topLeft (*self*: [depthai.Rect](#)) → [depthai.Point2f](#)

The top-left corner.

property width

property x

property y

class [depthai.RotatedRect](#)

Bases: [pybind11_builtins.pybind11_object](#)

Methods:

[__init__](#) (*self*)

Attributes:

[angle](#)

[center](#)

[size](#)

[__init__](#) (*self*: [depthai.RotatedRect](#)) → None

property angle

property center

property size

class [depthai.SPIOut](#)

Bases: [depthai.Node](#)

SPIOut node. Sends messages over SPI.

Methods:

<code>__init__(*args, **kwargs)</code>	Initialize self.
<code>setBusId(self, id)</code>	Specifies SPI Bus number to use
<code>setStreamName(self, name)</code>	Specifies stream name over which the node will send data

Attributes:

<code>input</code>	Input for any type of messages to be transfered over SPI stream
--------------------	---

`__init__(*args, **kwargs)`
Initialize self. See help(type(self)) for accurate signature.

property input

Input for any type of messages to be transfered over SPI stream

Default queue is blocking with size 8

setBusId (*self*: `depthai.SPIOut`, *id*: `int`) → `None`
Specifies SPI Bus number to use

Parameter id: SPI Bus id

setStreamName (*self*: `depthai.SPIOut`, *name*: `str`) → `None`
Specifies stream name over which the node will send data

Parameter name: Stream name

class `depthai.Size2f`

Bases: `pybind11_builtins.pybind11_object`

Methods:

<code>__init__(*args, **kwargs)</code>	Overloaded function.
--	----------------------

Attributes:

<code>height</code>	
<code>width</code>	

`__init__(*args, **kwargs)`
Overloaded function.

1. `__init__(self: depthai.Size2f) -> None`

2. `__init__(self: depthai.Size2f, arg0: float, arg1: float) -> None`

property height

property width

class `depthai.SpatialDetectionNetwork`

Bases: `depthai.DetectionNetwork`

SpatialDetectionNetwork node. Runs a neural inference on input image and calculates spatial location data.

Classes:

<i>Properties</i>	alias of <i>depthai.SpatialDetectionNetworkProperties</i>
-------------------	---

Methods:

<code>__init__(*args, **kwargs)</code>	Initialize self.
<code>setBoundingBoxScaleFactor(self, scaleFactor)</code>	Specifies scale factor for detected bounding boxes.
<code>setDepthLowerThreshold(self, lowerThreshold)</code>	Specifies lower threshold in millimeters for depth values which will used to calculate spatial data
<code>setDepthUpperThreshold(self, upperThreshold)</code>	Specifies upper threshold in millimeters for depth values which will used to calculate spatial data

Attributes:

<i>boundingBoxMapping</i>	Outputs mapping of detected bounding boxes relative to depth map
<i>input</i>	Input message with data to be inferred upon Default queue is blocking with size 5
<i>inputDepth</i>	Input message with depth data used to retrieve spatial information about detected object Default queue is non-blocking with size 4
<i>out</i>	Outputs ImgDetections message that carries parsed detection results.
<i>passthrough</i>	Passthrough message on which the inference was performed.
<i>passthroughDepth</i>	Passthrough message for depth frame on which the spatial location calculation was performed.

Propertiesalias of *depthai.SpatialDetectionNetworkProperties* **Methods:**

<code>__init__(*args, **kwargs)</code>	Initialize self.
--	------------------

Attributes:

<code>depthThresholds</code>
<code>detectedBBScaleFactor</code>

`__init__(*args, **kwargs)`
Initialize self. See help(type(self)) for accurate signature.

property boundingBoxMapping

Outputs mapping of detected bounding boxes relative to depth map

Suitable for when displaying remapped bounding boxes on depth frame

property input

Input message with data to be inferred upon Default queue is blocking with size 5

property inputDepth

Input message with depth data used to retrieve spatial information about detected object Default queue is non-blocking with size 4

property out

Outputs ImgDetections message that carries parsed detection results.

property passthrough

Passthrough message on which the inference was performed.

Suitable for when input queue is set to non-blocking behavior.

property passthroughDepth

Passthrough message for depth frame on which the spatial location calculation was performed.

Suitable for when input queue is set to non-blocking behavior.

setBoundingBoxScaleFactor (*self*: [depthai.SpatialDetectionNetwork](#), *scaleFactor*: *float*) → *None*

Specifies scale factor for detected bounding boxes.

Parameter scaleFactor: Scale factor must be in the interval (0,1].

setDepthLowerThreshold (*self*: [depthai.SpatialDetectionNetwork](#), *lowerThreshold*: *int*) → *None*

Specifies lower threshold in millimeters for depth values which will used to calculate spatial data

Parameter lowerThreshold: LowerThreshold must be in the interval [0,upperThreshold] and less than upperThreshold.

setDepthUpperThreshold (*self*: [depthai.SpatialDetectionNetwork](#), *upperThreshold*: *int*) → *None*

Specifies upper threshold in millimeters for depth values which will used to calculate spatial data

Parameter upperThreshold: UpperThreshold must be in the interval [lowerThreshold,65535].

class [depthai.SpatialDetectionNetworkProperties](#)

Bases: [depthai.DetectionNetworkProperties](#)

Properties for SpatialDetectionNetwork

Methods:

<code>__init__</code> (*args, **kwargs)	Initialize self.
---	------------------

Attributes:

<code>depthThresholds</code>
<code>detectedBBScaleFactor</code>

`__init__` (*args, **kwargs)
Initialize self. See help(type(self)) for accurate signature.

property depthThresholds

property detectedBBScaleFactor

class [depthai.SpatialImgDetection](#)

Bases: [depthai.ImgDetection](#)

Spatial image detection structure

Contains image detection results together with spatial location data.

Methods:

`__init__(self)`

Attributes:

`spatialCoordinates`

`__init__(self: depthai.SpatialImgDetection) → None`
property `spatialCoordinates`
class `depthai.SpatialImgDetections`

 Bases: `depthai.Buffer`

SpatialImgDetections message. Carries detection results together with spatial location data

Methods:

`__init__(self)`

Attributes:

`detections`

`__init__(self: depthai.SpatialImgDetections) → None`
property `detections`
class `depthai.SpatialLocationCalculator`

 Bases: `depthai.Node`

SpatialLocationCalculator node. Calculates spatial location data on a set of ROIs on depth map.

Classes:

<code>Properties</code>	alias of <code>depthai.SpatialLocationCalculatorProperties</code>
-------------------------	---

Methods:

<code>__init__(*args, **kwargs)</code>	Initialize self.
<code>setWaitForConfigInput(self, wait)</code>	Specify whether or not wait until configuration message arrives to inputConfig Input.

Attributes:

<code>initialConfig</code>	Initial config to use when calculating spatial location data.
<code>inputConfig</code>	Input <code>SpatialLocationCalculatorConfig</code> message with ability to modify parameters in runtime.
<code>inputDepth</code>	Input message with depth data used to retrieve spatial information about detected object.
<code>out</code>	Outputs <code>SpatialLocationCalculatorData</code> message that carries spatial location results.

continues on next page

Table 186 – continued from previous page

<i>passthroughDepth</i>	Passthrough message on which the calculation was performed.
-------------------------	---

Propertiesalias of *depthai.SpatialLocationCalculatorProperties* **Methods:**

<code>__init__(*args, **kwargs)</code>	Initialize self.
--	------------------

Attributes:

<i>inputConfigSync</i>
<i>roiConfig</i>

`__init__(*args, **kwargs)`
Initialize self. See help(type(self)) for accurate signature.

property initialConfig

Initial config to use when calculating spatial location data.

property inputConfig

Input SpatialLocationCalculatorConfig message with ability to modify parameters in runtime. Default queue is non-blocking with size 4.

property inputDepth

Input message with depth data used to retrieve spatial information about detected object. Default queue is non-blocking with size 4.

property out

Outputs SpatialLocationCalculatorData message that carries spatial location results.

property passthroughDepth

Passthrough message on which the calculation was performed. Suitable for when input queue is set to non-blocking behavior.

setWaitForConfigInput (*self*: *depthai.SpatialLocationCalculator*, *wait*: *bool*) → *None*

Specify whether or not wait until configuration message arrives to inputConfig Input.

Parameter wait: True to wait for configuration message, false otherwise.**class depthai.SpatialLocationCalculatorConfig**Bases: *depthai.Buffer*

SpatialLocationCalculatorConfig message. Carries ROI (region of interest) and threshold for depth calculation

Methods:

<code>__init__(self)</code>	
<code>addROI(self, ROI)</code>	Add a new ROI to configuration data.
<code>getConfigData(self)</code>	Retrieve configuration data for SpatialLocationCalculator
<code>setROIs(self, ROIs)</code>	Set a vector of ROIs as configuration data.

`__init__(self: depthai.SpatialLocationCalculatorConfig) → None`

addROI (*self*: *depthai.SpatialLocationCalculatorConfig*, *ROI*: *depthai.SpatialLocationCalculatorConfigData*) → *None*
Add a new ROI to configuration data.

Parameter roi: Configuration parameters for ROI (region of interest)

getConfigData (*self*: `depthai.SpatialLocationCalculatorConfig`) →
List[`depthai.SpatialLocationCalculatorConfigData`]
Retrieve configuration data for SpatialLocationCalculator

Returns Vector of configuration parameters for ROIs (region of interests)

setROIs (*self*: `depthai.SpatialLocationCalculatorConfig`, *ROIs*: List[`depthai.SpatialLocationCalculatorConfigData`])
→ None
Set a vector of ROIs as configuration data.

Parameter ROIs: Vector of configuration parameters for ROIs (region of interests)

class `depthai.SpatialLocationCalculatorConfigData`

Bases: `pybind11_builtins.pybind11_object`

Methods:

`__init__`(*self*)

Attributes:

`depthThresholds`
`roi`

`__init__` (*self*: `depthai.SpatialLocationCalculatorConfigData`) → None

property `depthThresholds`

property `roi`

class `depthai.SpatialLocationCalculatorConfigThresholds`

Bases: `pybind11_builtins.pybind11_object`

Spatial location configuration thresholds structure

Contains configuration data for lower and upper threshold in millimeters for ROI. Values outside of threshold range will be ignored when calculating spatial coordinates from depth map.

Methods:

`__init__`(*self*)

Attributes:

`lowerThreshold`
`upperThreshold`

`__init__` (*self*: `depthai.SpatialLocationCalculatorConfigThresholds`) → None

property `lowerThreshold`

property `upperThreshold`

class `depthai.SpatialLocationCalculatorData`

Bases: `depthai.Buffer`

SpatialLocationCalculatorData message. Carries spatial information (X,Y,Z) and their configuration parameters

Methods:

<code>__init__(self)</code>	
<code>getSpatialLocations(self)</code>	Retrieve configuration data for SpatialLocationCalculatorData.

`__init__ (self: depthai.SpatialLocationCalculatorData) → None`

`getSpatialLocations (self: depthai.SpatialLocationCalculatorData) → List[depthai.SpatialLocations]`
 Retrieve configuration data for SpatialLocationCalculatorData.

Returns Vector of spatial location data, carrying spatial information (X,Y,Z)

class `depthai.SpatialLocationCalculatorProperties`

Bases: `pybind11_builtins.pybind11_object`

Specify SpatialLocationCalculator options

Methods:

<code>__init__(*args, **kwargs)</code>	Initialize self.
--	------------------

Attributes:

<code>inputConfigSync</code>
<code>roiConfig</code>

`__init__ (*args, **kwargs)`
 Initialize self. See `help(type(self))` for accurate signature.

property `inputConfigSync`

property `roiConfig`

class `depthai.SpatialLocations`

Bases: `pybind11_builtins.pybind11_object`

Spatial location information structure

Contains configuration data, average depth for the calculated ROI on depth map. Together with spatial coordinates: x,y,z relative to the center of depth map. Units are in millimeters.

Methods:

<code>__init__(self)</code>

Attributes:

<code>config</code>
<code>depthAverage</code>
<code>depthAveragePixelCount</code>
<code>spatialCoordinates</code>

`__init__ (self: depthai.SpatialLocations) → None`

property `config`

property `depthAverage`

property `depthAveragePixelCount`**property** `spatialCoordinates`**class** `depthai.StereoDepth`Bases: `depthai.Node`

StereoDepth node. Compute stereo disparity and depth from left-right image pair.

Classes:

<i>Properties</i>	alias of <code>depthai.StereoDepthProperties</code>
-------------------	---

Methods:

<code>__init__(*args, **kwargs)</code>	Initialize self.
<code>loadCalibrationData(self, data)</code>	Specify calibration data as a vector of bytes
<code>loadCalibrationFile(self, path)</code>	Specify local filesystem path to the calibration file
<code>setConfidenceThreshold(self, confThr)</code>	Confidence threshold for disparity calculation
<code>setEmptyCalibration(self)</code>	Specify that a passthrough/dummy calibration should be used, when input frames are already rectified (e.g.
<code>setExtendedDisparity(self, enable)</code>	Disparity range increased from 96 to 192, combined from full resolution and downsampled images.
<code>setInputResolution(self, width, height)</code>	Specify input resolution size
<code>setLeftRightCheck(self, enable)</code>	Computes and combines disparities in both L-R and R-L directions, and combine them.
<code>setMedianFilter(self, median)</code>	Parameter <code>median</code> :
<code>setOutputDepth(self, enable)</code>	Enable outputting 'depth' stream (converted from disparity).
<code>setOutputRectified(self, enable)</code>	Enable outputting rectified frames.
<code>setRectifyEdgeFillColor(self, color)</code>	Fill color for missing data at frame edges
<code>setRectifyMirrorFrame(self, enable)</code>	Mirror rectified frames
<code>setSubpixel(self, enable)</code>	Computes disparity with sub-pixel interpolation (5 fractional bits).

Attributes:

<code>depth</code>	Outputs <code>ImgFrame</code> message that carries RAW16 encoded (0..65535) depth data in millimeters.
<code>disparity</code>	Outputs <code>ImgFrame</code> message that carries RAW8 encoded (0..96 or 0..192 for Extended mode) disparity data.
<code>left</code>	Input for left <code>ImgFrame</code> of left-right pair
<code>rectifiedLeft</code>	Outputs <code>ImgFrame</code> message that carries RAW8 encoded (grayscale) rectified frame data.
<code>rectifiedRight</code>	Outputs <code>ImgFrame</code> message that carries RAW8 encoded (grayscale) rectified frame data.
<code>right</code>	Input for right <code>ImgFrame</code> of left-right pair
<code>syncedLeft</code>	Passthrough <code>ImgFrame</code> message from 'left' Input.
<code>syncedRight</code>	Passthrough <code>ImgFrame</code> message from 'right' Input.

Propertiesalias of `depthai.StereoDepthProperties` **Classes:**

MedianFilter	Median filter config for disparity post-processing
Methods:	
<code>__init__(*args, **kwargs)</code>	Initialize self.
Attributes:	
calibration	
confidenceThreshold	
enableExtendedDisparity	
enableLeftRightCheck	
enableOutputDepth	
enableOutputRectified	
enableSubpixel	
height	
median	
rectifyEdgeFillColor	
rectifyMirrorFrame	
width	

`__init__(*args, **kwargs)`
Initialize self. See help(type(self)) for accurate signature.

property depth

Outputs ImgFrame message that carries RAW16 encoded (0..65535) depth data in millimeters.

property disparity

Outputs ImgFrame message that carries RAW8 encoded (0..96 or 0..192 for Extended mode) disparity data.

property left

Input for left ImgFrame of left-right pair

Default queue is non-blocking with size 8

loadCalibrationData (*self*: [depthai.StereoDepth](#), *data*: *List[int]*) → *None*

Specify calibration data as a vector of bytes

Parameter path: Calibration data. If empty use EEPROM

loadCalibrationFile (*self*: [depthai.StereoDepth](#), *path*: *str*) → *None*

Specify local filesystem path to the calibration file

Parameter path: Path to calibration file. If empty use EEPROM

property rectifiedLeft

Outputs ImgFrame message that carries RAW8 encoded (grayscale) rectified frame data.

property rectifiedRight

Outputs ImgFrame message that carries RAW8 encoded (grayscale) rectified frame data.

property right

Input for right ImgFrame of left-right pair

Default queue is non-blocking with size 8

setConfidenceThreshold (*self*: [depthai.StereoDepth](#), *confThr*: *int*) → [None](#)

Confidence threshold for disparity calculation

Parameter confThr: Confidence threshold value 0..255

setEmptyCalibration (*self*: [depthai.StereoDepth](#)) → [None](#)

Specify that a passthrough/dummy calibration should be used, when input frames are already rectified (e.g. sourced from recordings on the host)

setExtendedDisparity (*self*: [depthai.StereoDepth](#), *enable*: *bool*) → [None](#)

Disparity range increased from 96 to 192, combined from full resolution and downsampled images.

Suitable for short range objects

setInputResolution (*self*: [depthai.StereoDepth](#), *width*: *int*, *height*: *int*) → [None](#)

Specify input resolution size

Optional if MonoCamera exists, otherwise necessary

setLeftRightCheck (*self*: [depthai.StereoDepth](#), *enable*: *bool*) → [None](#)

Computes and combines disparities in both L-R and R-L directions, and combine them.

For better occlusion handling

setMedianFilter (*self*: [depthai.StereoDepth](#), *median*: [dai::StereoDepthProperties::MedianFilter](#)) → [None](#)

Parameter median: Set kernel size for disparity/depth median filtering, or disable

setOutputDepth (*self*: [depthai.StereoDepth](#), *enable*: *bool*) → [None](#)

Enable outputting ‘depth’ stream (converted from disparity). In certain configurations, this will disable ‘disparity’ stream

setOutputRectified (*self*: [depthai.StereoDepth](#), *enable*: *bool*) → [None](#)

Enable outputting rectified frames. Optimizes computation on device side when disabled

setRectifyEdgeFillColor (*self*: [depthai.StereoDepth](#), *color*: *int*) → [None](#)

Fill color for missing data at frame edges

Parameter color: Grayscale 0..255, or -1 to replicate pixels

setRectifyMirrorFrame (*self*: [depthai.StereoDepth](#), *enable*: *bool*) → [None](#)

Mirror rectified frames

Parameter enable: True for normal disparity/depth, otherwise mirrored

setSubpixel (*self*: [depthai.StereoDepth](#), *enable*: *bool*) → [None](#)

Computes disparity with sub-pixel interpolation (5 fractional bits).

Suitable for long range

property syncedLeft

Passthrough ImgFrame message from ‘left’ Input.

property syncedRight

Passthrough ImgFrame message from ‘right’ Input.

class [depthai.StereoDepthProperties](#)

Bases: [pybind11_builtins.pybind11_object](#)

Specify StereoDepth options

Classes:

MedianFilter

Median filter config for disparity post-processing

Methods:*__init__*(*args, **kwargs)

Initialize self.

Attributes:*calibration**confidenceThreshold**enableExtendedDisparity**enableLeftRightCheck**enableOutputDepth**enableOutputRectified**enableSubpixel**height**median**rectifyEdgeFillColor**rectifyMirrorFrame**width***class MedianFilter**Bases: `pybind11_builtins.pybind11_object`

Median filter config for disparity post-processing

Members:

MEDIAN_OFF

KERNEL_3x3

KERNEL_5x5

KERNEL_7x7

Attributes:*KERNEL_3x3**KERNEL_5x5**KERNEL_7x7**MEDIAN_OFF**name**value***Methods:***__init__*(self, value)

KERNEL_3x3 = <MedianFilter.KERNEL_3x3: 3>

KERNEL_5x5 = <MedianFilter.KERNEL_5x5: 5>

KERNEL_7x7 = <MedianFilter.KERNEL_7x7: 7>

```
MEDIAN_OFF = <MedianFilter.MEDIAN_OFF: 0>

__init__(self: depthai.StereoDepthProperties.MedianFilter, value: int) → None

property name
property value

__init__(*args, **kwargs)
    Initialize self. See help(type(self)) for accurate signature.

property calibration
property confidenceThreshold
property enableExtendedDisparity
property enableLeftRightCheck
property enableOutputDepth
property enableOutputRectified
property enableSubpixel
property height
property median
property rectifyEdgeFillColor
property rectifyMirrorFrame
property width
```

```
class depthai.SystemInformation
```

```
    Bases: depthai.Buffer
```

```
    SystemInformation message. Carries memory usage, cpu usage and chip temperatures.
```

```
    Methods:
```

```
__init__(self)
```

```
    Attributes:
```

```
chipTemperature
```

```
cmxMemoryUsage
```

```
ddrMemoryUsage
```

```
leonCssCpuUsage
```

```
leonCssMemoryUsage
```

```
leonMssCpuUsage
```

```
leonMssMemoryUsage
```

```
__init__(self: depthai.SystemInformation) → None
```

```
property chipTemperature
```

```
property cmxMemoryUsage
```

```
property ddrMemoryUsage
```

```
property leonCssCpuUsage
```

```
property leonCssMemoryUsage
```

property `leonMssCpuUsage`

property `leonMssMemoryUsage`

class `depthai.SystemLogger`

Bases: `depthai.Node`

SystemLogger node. Send system information periodically.

Methods:

<code>__init__</code> (*args, **kwargs)	Initialize self.
<code>setRate</code> (self, hz)	Specify logging rate, at which messages will be sent to out output

Attributes:

<code>out</code>	Outputs SystemInformation message that carries various system information like memory and CPU usage, temperatures, ...
------------------	--

`__init__` (*args, **kwargs)

Initialize self. See help(type(self)) for accurate signature.

property `out`

Outputs SystemInformation message that carries various system information like memory and CPU usage, temperatures, ...

setRate (self: `depthai.SystemLogger`, hz: *float*) → *None*

Specify logging rate, at which messages will be sent to out output

Parameter hz: Sending rate in hertz (messages per second)

class `depthai.SystemLoggerProperties`

Bases: `pybind11_builtins.pybind11_object`

Methods:

<code>__init__</code> (*args, **kwargs)	Initialize self.
---	------------------

Attributes:

<code>rateHz</code>	
---------------------	--

`__init__` (*args, **kwargs)

Initialize self. See help(type(self)) for accurate signature.

property `rateHz`

class `depthai.TensorInfo`

Bases: `pybind11_builtins.pybind11_object`

Classes:

<code>DataType</code>	Members:
<code>StorageOrder</code>	Members:

Methods:

`__init__(self)`

Attributes:

`dataType`

`dims`

`name`

`numDimensions`

`offset`

`order`

`strides`

class DataTypeBases: `pybind11_builtins.pybind11_object`

Members:

FP16

U8F

INT

FP32

I8

Attributes:

`FP16`

`FP32`

`I8`

`INT`

`U8F`

`name`

`value`

Methods:

`__init__(self, value)`

`FP16 = <DataType.FP16: 0>``FP32 = <DataType.FP32: 3>``I8 = <DataType.I8: 4>``INT = <DataType.INT: 2>``U8F = <DataType.U8F: 1>``__init__(self: depthai.TensorInfo.DataType, value: int) → None``property name``property value`

```
class StorageOrder
    Bases: pybind11_builtins.pybind11_object
    Members:
        NHWC
        NHCW
        NCHW
        HWC
        CHW
        WHC
        HCW
        WCH
        CWH
        NC
        CN
        C
        H
        W
    Attributes:
```

<i>C</i>
<i>CHW</i>
<i>CN</i>
<i>CWH</i>
<i>H</i>
<i>HCW</i>
<i>HWC</i>
<i>NC</i>
<i>NCHW</i>
<i>NHCW</i>
<i>NHWC</i>
<i>W</i>
<i>WCH</i>
<i>WHC</i>
<i>name</i>
<i>value</i>

Methods:

```
__init__(self, value)
```

```
C = <StorageOrder.C: 3>
CHW = <StorageOrder.CHW: 801>
CN = <StorageOrder.CN: 52>
```

```
CWH = <StorageOrder.CWH: 786>
H = <StorageOrder.H: 2>
HCW = <StorageOrder.HCW: 561>
HWC = <StorageOrder.HWC: 531>
NC = <StorageOrder.NC: 67>
NCHW = <StorageOrder.NCHW: 17185>
NHCW = <StorageOrder.NHCW: 16945>
NHWC = <StorageOrder.NHWC: 16915>
W = <StorageOrder.W: 1>
WCH = <StorageOrder.WCH: 306>
WHC = <StorageOrder.WHC: 291>

__init__(self: depthai.TensorInfo.StorageOrder, value: int) → None
property name
property value
__init__(self: depthai.TensorInfo) → None
property dataType
property dims
property name
property numDimensions
property offset
property order
property strides

class depthai.Timestamp
    Bases: pybind11_builtins.pybind11_object
    Methods:

    __init__(self)

    Attributes:

    nsec
    sec

    __init__(self: depthai.Timestamp) → None
    property nsec
    property sec

class depthai.TrackerIdAssignmentPolicy
    Bases: pybind11_builtins.pybind11_object
    Members:
```

UNIQUE_ID

SMALLEST_ID

Attributes:

SMALLEST_ID

UNIQUE_ID

name

value

Methods:

__init__(self, value)

SMALLEST_ID = <TrackerIdAssignmentPolicy.SMALLEST_ID: 1>

UNIQUE_ID = <TrackerIdAssignmentPolicy.UNIQUE_ID: 0>

__init__(self: depthai.TrackerIdAssignmentPolicy, value: int) → None

property name

property value

class depthai.TrackerType

Bases: pybind11_builtins.pybind11_object

Members:

ZERO_TERM_IMAGELESS

ZERO_TERM_COLOR_HISTOGRAM

Attributes:

ZERO_TERM_COLOR_HISTOGRAM

ZERO_TERM_IMAGELESS

name

value

Methods:

__init__(self, value)

ZERO_TERM_COLOR_HISTOGRAM = <TrackerType.ZERO_TERM_COLOR_HISTOGRAM: 6>

ZERO_TERM_IMAGELESS = <TrackerType.ZERO_TERM_IMAGELESS: 5>

__init__(self: depthai.TrackerType, value: int) → None

property name

property value

class depthai.Tracklet

Bases: pybind11_builtins.pybind11_object

Tracklet structure

Contains tracklets from object tracker output.

Classes:

TrackingStatus

Members:

Methods:

__init__(self)

Attributes:

id

label

roi

spatialCoordinates

srcImgDetection

status

class TrackingStatus

Bases: `pybind11_builtins.pybind11_object`

Members:

NEW

TRACKED

LOST

Attributes:

LOST

NEW

TRACKED

name

value

Methods:

__init__(self, value)

LOST = <TrackingStatus.LOST: 2>

NEW = <TrackingStatus.NEW: 0>

TRACKED = <TrackingStatus.TRACKED: 1>

__init__(self: `depthai.Tracklet.TrackingStatus`, value: *int*) → None

property name

property value

__init__(self: `depthai.Tracklet`) → None

property id

property label

property roi

property spatialCoordinates

property srcImgDetection

property status

class depthai.Tracklets

Bases: *depthai.Buffer*

Tracklets message. Carries object tracking information.

Methods:

__init__(self)

Attributes:

<i>tracklets</i>	Retrieve data for Tracklets.
------------------	------------------------------

__init__(self: depthai.Tracklets) → None

property tracklets

Retrieve data for Tracklets.

Returns Vector of object tracker data, carrying tracking information.

class depthai.VideoEncoder

Bases: *depthai.Node*

VideoEncoder node. Encodes frames into MJPEG, H264 or H265.

Classes:

<i>Properties</i>	alias of <i>depthai.VideoEncoderProperties</i>
-------------------	--

Methods:

<i>__init__</i> (*args, **kwargs)	Initialize self.
<i>getBitrate</i> (self)	Get bitrate in bps
<i>getBitrateKbps</i> (self)	Get bitrate in kbps
<i>getFrameRate</i> (self)	Get frame rate
<i>getHeight</i> (self)	Get input height
<i>getKeyframeFrequency</i> (self)	Get keyframe frequency
<i>getLossless</i> (self)	Get lossless mode.
<i>getNumBFrames</i> (self)	Get number of B frames
<i>getNumFramesPool</i> (self)	Get number of frames in pool
<i>getProfile</i> (self)	Get profile
<i>getQuality</i> (self)	Get quality
<i>getRateControlMode</i> (self)	Get rate control mode
<i>getSize</i> (self)	Get input size
<i>getWidth</i> (self)	Get input width
<i>setBitrate</i> (self, bitrateKbps)	Set output bitrate in bps.

continues on next page

Table 237 – continued from previous page

<code>setBitrateKbps(self, bitrateKbps)</code>	Set output bitrate in kbps.
<code>setDefaultProfilePreset(*args, **kwargs)</code>	Overloaded function.
<code>setFrameRate(self, frameRate)</code>	Sets expected frame rate
<code>setKeyframeFrequency(self, freq)</code>	Set keyframe frequency.
<code>setLossless(self, arg0)</code>	Set lossless mode.
<code>setNumBFrames(self, numBFrames)</code>	Set number of B frames to be inserted
<code>setNumFramesPool(self, frames)</code>	Set number of frames in pool
<code>setProfile(*args, **kwargs)</code>	Overloaded function.
<code>setQuality(self, quality)</code>	Set quality
<code>setRateControlMode(self, mode)</code>	Set rate control mode

Attributes:

<code>bitstream</code>	Outputs <code>ImgFrame</code> message that carries BITSTREAM encoded (MJPEG, H264 or H265) frame data.
<code>input</code>	Input for NV12 <code>ImgFrame</code> to be encoded Default queue is blocking with size set by 'setNumFramesPool' (4).

Propertiesalias of `depthai.VideoEncoderProperties` **Classes:**

Profile	Encoding profile, H264, H265 or MJPEG
RateControlMode	Rate control mode specifies if constant or variable bitrate should be used (H264 / H265)

Methods:

<code>__init__(*args, **kwargs)</code>	Initialize self.
--	------------------

Attributes:

<code>bitrate</code>
<code>height</code>
<code>keyframeFrequency</code>
<code>maxBitrate</code>
<code>numBFrames</code>
<code>numFramesPool</code>
<code>profile</code>
<code>quality</code>
<code>rateCtrlMode</code>
<code>width</code>

`__init__(*args, **kwargs)`
Initialize self. See `help(type(self))` for accurate signature.

property bitstreamOutputs `ImgFrame` message that carries BITSTREAM encoded (MJPEG, H264 or H265) frame data.**getBitrate** (*self*: `depthai.VideoEncoder`) → `int`

Get bitrate in bps

getBitrateKbps (*self*: depthai.VideoEncoder) → int
Get bitrate in kbps

getFrameRate (*self*: depthai.VideoEncoder) → int
Get frame rate

getHeight (*self*: depthai.VideoEncoder) → int
Get input height

getKeyframeFrequency (*self*: depthai.VideoEncoder) → int
Get keyframe frequency

getLossless (*self*: depthai.VideoEncoder) → bool
Get lossless mode. Applies only when using [M]JPEG profile.

getNumBFrames (*self*: depthai.VideoEncoder) → int
Get number of B frames

getNumFramesPool (*self*: depthai.VideoEncoder) → int
Get number of frames in pool

Returns Number of pool frames

getProfile (*self*: depthai.VideoEncoder) → dai::VideoEncoderProperties::Profile
Get profile

getQuality (*self*: depthai.VideoEncoder) → int
Get quality

getRateControlMode (*self*: depthai.VideoEncoder) → dai::VideoEncoderProperties::RateControlMode
Get rate control mode

getSize (*self*: depthai.VideoEncoder) → Tuple[int, int]
Get input size

getWidth (*self*: depthai.VideoEncoder) → int
Get input width

property input
Input for NV12 ImgFrame to be encoded Default queue is blocking with size set by 'setNumFramesPool' (4).

setBitrate (*self*: depthai.VideoEncoder, *bitrateKbps*: int) → None
Set output bitrate in bps. Final bitrate depends on rate control mode

setBitrateKbps (*self*: depthai.VideoEncoder, *bitrateKbps*: int) → None
Set output bitrate in kbps. Final bitrate depends on rate control mode

setDefaultProfilePreset (*args, **kwargs)
Overloaded function.

1. setDefaultProfilePreset(*self*: depthai.VideoEncoder, *width*: int, *height*: int, *fps*: float, *profile*: dai::VideoEncoderProperties::Profile) -> None

Sets a default preset based on specified input size, frame rate and profile

Parameter width: Input frame width

Parameter height: Input frame height

Parameter fps: Frame rate in frames per second

Parameter profile: Encoding profile

2. `setDefaultProfilePreset(self: depthai.VideoEncoder, size: Tuple[int, int], fps: float, profile: dai::VideoEncoderProperties::Profile) -> None`

Sets a default preset based on specified input size, frame rate and profile

Parameter size: Input frame size

Parameter fps: Frame rate in frames per second

Parameter profile: Encoding profile

setFrameRate (*self: depthai.VideoEncoder, frameRate: int*) \rightarrow None

Sets expected frame rate

Parameter frameRate: Frame rate in frames per second

setKeyframeFrequency (*self: depthai.VideoEncoder, freq: int*) \rightarrow None

Set keyframe frequency. Every Nth frame a keyframe is inserted.

Applicable only to H264 and H265 profiles

Examples:

- 30 FPS video, keyframe frequency: 30. Every 1s a keyframe will be inserted
- 60 FPS video, keyframe frequency: 180. Every 3s a keyframe will be inserted

setLossless (*self: depthai.VideoEncoder, arg0: bool*) \rightarrow None

Set lossless mode. Applies only to [M]JPEG profile

Parameter lossless: True to enable lossless jpeg encoding, false otherwise

setNumBFrames (*self: depthai.VideoEncoder, numBFrames: int*) \rightarrow None

Set number of B frames to be inserted

setNumFramesPool (*self: depthai.VideoEncoder, frames: int*) \rightarrow None

Set number of frames in pool

Parameter frames: Number of pool frames

setProfile (**args, **kwargs*)

Overloaded function.

1. `setProfile(self: depthai.VideoEncoder, size: Tuple[int, int], profile: dai::VideoEncoderProperties::Profile) -> None`

Set encoding profile

2. `setProfile(self: depthai.VideoEncoder, width: int, height: int, profile: dai::VideoEncoderProperties::Profile) -> None`

Set encoding profile

setQuality (*self: depthai.VideoEncoder, quality: int*) \rightarrow None

Set quality

Parameter quality: Value between 0-100%. Approximates quality

setRateControlMode (*self: depthai.VideoEncoder, mode: dai::VideoEncoderProperties::RateControlMode*) \rightarrow None

Set rate control mode

class `depthai.VideoEncoderProperties`

Bases: `pybind11_builtins.pybind11_object`

Specify VideoEncoder options such as profile, bitrate, ...

Classes:

<i>Profile</i>	Encoding profile, H264, H265 or MJPEG
<i>RateControlMode</i>	Rate control mode specifies if constant or variable bitrate should be used (H264 / H265)

Methods:

<code>__init__(*args, **kwargs)</code>	Initialize self.
--	------------------

Attributes:

<i>bitrate</i>
<i>height</i>
<i>keyframeFrequency</i>
<i>maxBitrate</i>
<i>numBFrames</i>
<i>numFramesPool</i>
<i>profile</i>
<i>quality</i>
<i>rateCtrlMode</i>
<i>width</i>

class ProfileBases: `pybind11_builtins.pybind11_object`

Encoding profile, H264, H265 or MJPEG

Members:

H264_BASELINE

H264_HIGH

H264_MAIN

H265_MAIN

MJPEG

Attributes:

<i>H264_BASELINE</i>
<i>H264_HIGH</i>
<i>H264_MAIN</i>
<i>H265_MAIN</i>
<i>MJPEG</i>
<i>name</i>
<i>value</i>

Methods:

<code>__init__(self, value)</code>

H264_BASELINE = <Profile.H264_BASELINE: 0>

H264_HIGH = <Profile.H264_HIGH: 1>

H264_MAIN = <Profile.H264_MAIN: 2>

H265_MAIN = <Profile.H265_MAIN: 3>

MJPEG = <Profile.MJPEG: 4>

`__init__(self: depthai.VideoEncoderProperties.Profile, value: int) → None`

property name

property value

class RateControlMode

Bases: `pybind11_builtins.pybind11_object`

Rate control mode specifies if constant or variable bitrate should be used (H264 / H265)

Members:

CBR

VBR

Attributes:

CBR

VBR

name

value

Methods:

`__init__(self, value)`

CBR = <RateControlMode.CBR: 0>

VBR = <RateControlMode.VBR: 1>

`__init__(self: depthai.VideoEncoderProperties.RateControlMode, value: int) → None`

property name

property value

`__init__(*args, **kwargs)`

Initialize self. See `help(type(self))` for accurate signature.

property bitrate

property height

property keyframeFrequency

property maxBitrate

property numBFrames

property numFramesPool

property profile

property quality

property rateCtrlMode

property width

class depthai.XLinkConnection

Bases: pybind11_builtins.pybind11_object

Methods:

<code>__init__(*args, **kwargs)</code>	Overloaded function.
<code>getAllConnectedDevices(state)</code>	
<code>getDeviceByMxId(mxId, state)</code>	
<code>getFirstDevice(state)</code>	

`__init__(*args, **kwargs)`

Overloaded function.

1. `__init__(self: depthai.XLinkConnection, arg0: depthai.DeviceInfo, arg1: List[int]) -> None`
2. `__init__(self: depthai.XLinkConnection, arg0: depthai.DeviceInfo, arg1: str) -> None`
3. `__init__(self: depthai.XLinkConnection, arg0: depthai.DeviceInfo) -> None`

static `getAllConnectedDevices(state: depthai.XLinkDeviceState = <XLinkDeviceState.X_LINK_ANY_STATE: 0>) → List[depthai.DeviceInfo]`

static `getDeviceByMxId(mxId: str, state: depthai.XLinkDeviceState = <XLinkDeviceState.X_LINK_ANY_STATE: 0>) → Tuple[bool, depthai.DeviceInfo]`

static `getFirstDevice(state: depthai.XLinkDeviceState = <XLinkDeviceState.X_LINK_ANY_STATE: 0>) → Tuple[bool, depthai.DeviceInfo]`

class depthai.XLinkDeviceState

Bases: pybind11_builtins.pybind11_object

Members:

X_LINK_ANY_STATE

X_LINK_BOOTED

X_LINK_UNBOOTED

X_LINK_BOOTLOADER

Attributes:

<code>X_LINK_ANY_STATE</code>
<code>X_LINK_BOOTED</code>
<code>X_LINK_BOOTLOADER</code>
<code>X_LINK_UNBOOTED</code>
<code>name</code>
<code>value</code>

Methods:

<code>__init__(self, value)</code>

`X_LINK_ANY_STATE = <XLinkDeviceState.X_LINK_ANY_STATE: 0>`

`X_LINK_BOOTED = <XLinkDeviceState.X_LINK_BOOTED: 1>`

`X_LINK_BOOTLOADER = <XLinkDeviceState.X_LINK_BOOTLOADER: 3>`

```
X_LINK_UNBOOTED = <XLinkDeviceState.X_LINK_UNBOOTED: 2>
```

```
__init__(self: depthai.XLinkDeviceState, value: int) → None
```

property name

property value

```
class depthai.XLinkIn
```

Bases: *depthai.Node*

XLinkIn node. Receives messages over XLink.

Methods:

<code>__init__(*args, **kwargs)</code>	Initialize self.
<code>getMaxDataSize(self)</code>	Get maximum messages size in bytes
<code>getNumFrames(self)</code>	Get number of frames in pool
<code>getStreamName(self)</code>	Get stream name
<code>setMaxDataSize(self, maxDataSize)</code>	Set maximum message size it can receive
<code>setNumFrames(self, numFrames)</code>	Set number of frames in pool for sending messages forward
<code>setStreamName(self, streamName)</code>	Specifies XLink stream name to use.

Attributes:

<code>out</code>	Outputs message of same type as send from host.
------------------	---

```
__init__(*args, **kwargs)
    Initialize self. See help(type(self)) for accurate signature.
```

```
getMaxDataSize(self: depthai.XLinkIn) → int
    Get maximum messages size in bytes
```

```
getNumFrames(self: depthai.XLinkIn) → int
    Get number of frames in pool
```

```
getStreamName(self: depthai.XLinkIn) → str
    Get stream name
```

```
property out
    Outputs message of same type as send from host.
```

```
setMaxDataSize(self: depthai.XLinkIn, maxDataSize: int) → None
    Set maximum message size it can receive
```

Parameter maxDataSize: Maximum size in bytes

```
setNumFrames(self: depthai.XLinkIn, numFrames: int) → None
    Set number of frames in pool for sending messages forward
```

Parameter numFrames: Maximum number of frames in pool

```
setStreamName(self: depthai.XLinkIn, streamName: str) → None
    Specifies XLink stream name to use.
```

The name should not start with double underscores ‘__’, as those are reserved for internal use.

Parameter name: Stream name

```
class depthai.XLinkOut
```

Bases: *depthai.Node*

XLinkOut node. Sends messages over XLink.

Methods:

<code>__init__(*args, **kwargs)</code>	Initialize self.
<code>getFpsLimit(self)</code>	Get rate limit in messages per second
<code>getMetadataOnly(self)</code>	Get whether to transfer only messages attributes and not buffer data
<code>getStreamName(self)</code>	Get stream name
<code>setFpsLimit(self, fpsLimit)</code>	Specifies a message sending limit.
<code>setMetadataOnly(self, arg0)</code>	Specify whether to transfer only messages attributes and not buffer data
<code>setStreamName(self, streamName)</code>	Specifies XLink stream name to use.

Attributes:

<code>input</code>	Input for any type of messages to be transfered over XLink stream
--------------------	---

`__init__(*args, **kwargs)`

Initialize self. See `help(type(self))` for accurate signature.

getFpsLimit (*self*: `depthai.XLinkOut`) → `float`

Get rate limit in messages per second

getMetadataOnly (*self*: `depthai.XLinkOut`) → `bool`

Get whether to transfer only messages attributes and not buffer data

getStreamName (*self*: `depthai.XLinkOut`) → `str`

Get stream name

property input

Input for any type of messages to be transfered over XLink stream

Default queue is blocking with size 8

setFpsLimit (*self*: `depthai.XLinkOut`, *fpsLimit*: `float`) → `None`

Specifies a message sending limit. It's approximated from specified rate.

Parameter fps: Approximate rate limit in messages per second

setMetadataOnly (*self*: `depthai.XLinkOut`, *arg0*: `bool`) → `None`

Specify whether to transfer only messages attributes and not buffer data

setStreamName (*self*: `depthai.XLinkOut`, *streamName*: `str`) → `None`

Specifies XLink stream name to use.

The name should not start with double underscores `'__'`, as those are reserved for internal use.

Parameter name: Stream name

class `depthai.XLinkPlatform`

Bases: `pybind11_builtins.pybind11_object`

Members:

`X_LINK_ANY_PLATFORM`

`X_LINK_MYRIAD_2`

`X_LINK_MYRIAD_X`

Attributes:

X_LINK_ANY_PLATFORM

X_LINK_MYRIAD_2

X_LINK_MYRIAD_X

name

value

Methods:

__init__(self, value)

X_LINK_ANY_PLATFORM = <XLinkPlatform.X_LINK_ANY_PLATFORM: 0>

X_LINK_MYRIAD_2 = <XLinkPlatform.X_LINK_MYRIAD_2: 2450>

X_LINK_MYRIAD_X = <XLinkPlatform.X_LINK_MYRIAD_X: 2480>

__init__ (self: depthai.XLinkPlatform, value: int) → None

property name

property value

class depthai.XLinkProtocol

Bases: pybind11_builtins.pybind11_object

Members:

X_LINK_USB_VSC

X_LINK_USB_CDC

X_LINK_PCIE

X_LINK_IPC

X_LINK_NMB_OF_PROTOCOLS

X_LINK_ANY_PROTOCOL

Attributes:

X_LINK_ANY_PROTOCOL

X_LINK_IPC

X_LINK_NMB_OF_PROTOCOLS

X_LINK_PCIE

X_LINK_USB_CDC

X_LINK_USB_VSC

name

value

Methods:

__init__(self, value)

X_LINK_ANY_PROTOCOL = <XLinkProtocol.X_LINK_ANY_PROTOCOL: 5>

X_LINK_IPC = <XLinkProtocol.X_LINK_IPC: 3>

`X_LINK_NMB_OF_PROTOCOLS = <XLinkProtocol.X_LINK_NMB_OF_PROTOCOLS: 4>`

`X_LINK_PCIE = <XLinkProtocol.X_LINK_PCIE: 2>`

`X_LINK_USB_CDC = <XLinkProtocol.X_LINK_USB_CDC: 1>`

`X_LINK_USB_VSC = <XLinkProtocol.X_LINK_USB_VSC: 0>`

`__init__(self: depthai.XLinkProtocol, value: int) → None`

property name

property value

class `depthai.YoloDetectionNetwork`

Bases: `depthai.DetectionNetwork`

YoloDetectionNetwork node. Parses Yolo results

Methods:

<code>__init__(*args, **kwargs)</code>	Initialize self.
<code>setAnchorMasks(self, anchorMasks, List[int])</code>	Set anchor masks
<code>setAnchors(self, anchors)</code>	Set anchors
<code>setCoordinateSize(self, coordinates)</code>	Set coordianate size
<code>setIouThreshold(self, thresh)</code>	Set Iou threshold
<code>setNumClasses(self, numClasses)</code>	Set num classes

`__init__(*args, **kwargs)`

Initialize self. See help(type(self)) for accurate signature.

setAnchorMasks (*self*: `depthai.YoloDetectionNetwork`, *anchorMasks*: `Dict[str, List[int]]`) → `None`

Set anchor masks

setAnchors (*self*: `depthai.YoloDetectionNetwork`, *anchors*: `List[float]`) → `None`

Set anchors

setCoordinateSize (*self*: `depthai.YoloDetectionNetwork`, *coordinates*: `int`) → `None`

Set coordianate size

setIouThreshold (*self*: `depthai.YoloDetectionNetwork`, *thresh*: `float`) → `None`

Set Iou threshold

setNumClasses (*self*: `depthai.YoloDetectionNetwork`, *numClasses*: `int`) → `None`

Set num classes

class `depthai.YoloSpatialDetectionNetwork`

Bases: `depthai.SpatialDetectionNetwork`

YoloSpatialDetectionNetwork. (tiny)Yolov3/v4 based network with spatial location data.

Methods:

<code>__init__(*args, **kwargs)</code>	Initialize self.
<code>setAnchorMasks(self, anchorMasks, List[int])</code>	Set anchor masks
<code>setAnchors(self, anchors)</code>	Set anchors
<code>setCoordinateSize(self, coordinates)</code>	Set coordianate size
<code>setIouThreshold(self, thresh)</code>	Set Iou threshold
<code>setNumClasses(self, numClasses)</code>	Set num classes

`__init__(*args, **kwargs)`

Initialize self. See `help(type(self))` for accurate signature.

setAnchorMasks (*self*: `depthai.YoloSpatialDetectionNetwork`, *anchorMasks*: `Dict[str, List[int]]`) → `None`
Set anchor masks

setAnchors (*self*: `depthai.YoloSpatialDetectionNetwork`, *anchors*: `List[float]`) → `None`
Set anchors

setCoordinateSize (*self*: `depthai.YoloSpatialDetectionNetwork`, *coordinates*: `int`) → `None`
Set coordinate size

setIouThreshold (*self*: `depthai.YoloSpatialDetectionNetwork`, *thresh*: `float`) → `None`
Set Iou threshold

setNumClasses (*self*: `depthai.YoloSpatialDetectionNetwork`, *numClasses*: `int`) → `None`
Set num classes

3.13 C++ API Reference

namespace dai

Enums

enum CameraBoardSocket

Which Camera socket to use.

AUTO denotes that the decision will be made by device

Values:

enumerator AUTO

enumerator RGB

enumerator LEFT

enumerator RIGHT

enum CameraImageOrientation

Camera sensor image orientation / pixel readout. This exposes direct sensor settings. 90 or 270 degrees rotation is not available.

AUTO denotes that the decision will be made by device (e.g. on OAK-1/megaAI: ROTATE_180_DEG).

Values:

enumerator AUTO

enumerator NORMAL

enumerator HORIZONTAL_MIRROR

enumerator VERTICAL_FLIP

enumerator ROTATE_180_DEG

enum ProcessorType

On which processor the node will be placed

Enum specifying processor

Values:

```
    enumerator LOS
    enumerator LRT
enum DatatypeEnum
    Values:
    enumerator Buffer
    enumerator ImgFrame
    enumerator NNData
    enumerator ImageManipConfig
    enumerator CameraControl
    enumerator ImgDetections
    enumerator SpatialImgDetections
    enumerator SystemInformation
    enumerator SpatialLocationCalculatorConfig
    enumerator SpatialLocationCalculatorData
    enumerator Tracklets
enum LogLevel
    Values:
    enumerator TRACE
    enumerator DEBUG
    enumerator INFO
    enumerator WARN
    enumerator ERR
    enumerator CRITICAL
    enumerator OFF
enum TrackerType
    Values:
    enumerator ZERO_TERM_IMAGELESS
    enumerator ZERO_TERM_COLOR_HISTOGRAM
enum TrackerIdAssignmentPolicy
    Values:
    enumerator UNIQUE_ID
    enumerator SMALLEST_ID
```

Functions

```
std::ostream &operator<< (std::ostream &out, const Tracklet::TrackingStatus &status)

bool initialize ()

bool isDatatypeSubclassOf (DatatypeEnum parent, DatatypeEnum children)
```

Variables

```
constexpr const char *LOG_DEFAULT_PATTERN = "[%E.%e] [%n] [%^%l%$] %v"

constexpr const char *XLINK_CHANNEL_PIPELINE_CONFIG = "__pipeline_config"

constexpr const char *XLINK_CHANNEL_MAIN_RPC = "__rpc_main"

constexpr const char *XLINK_CHANNEL_TIMESYNC = "__timesync"

constexpr const char *XLINK_CHANNEL_LOG = "__log"

constexpr std::uint32_t XLINK_USB_BUFFER_MAX_SIZE = 5 * 1024 * 1024

constexpr const std::chrono::milliseconds XLINK_WATCHDOG_TIMEOUT = {1500}

class ADatatype
    #include <ADatatype.hpp> Abstract message.

    Subclassed by dai::Buffer

struct Asset
    #include <AssetManager.hpp> Asset is identified with string key and can store arbitrary binary data.

class AssetManager
    #include <AssetManager.hpp> AssetManager can store assets and serialize.
```

Public Functions

```
void addExisting (std::vector<std::shared_ptr<Asset>> assets)
    Adds all assets in an array to the AssetManager

    Parameters
    • assets: Vector of assets to add

void add (Asset asset)
    Adds an asset object to AssetManager.

    Parameters
    • asset: Asset to add

void add (const std::string &key, Asset asset)
    Adds an asset object to AssetManager with a specified key. Key value will be assigned to an Asset as well

    If asset with key already exists, the function throws an error

    Parameters
    • key: Key under which the asset should be stored
    • asset: Asset to store

void set (const std::string &key, Asset asset)
    Adds or overwrites existing asset with a specified key.

    Parameters
```

- key: Key under which the asset should be stored
- asset: *Asset* to store

`std::shared_ptr<const Asset> get (const std::string &key) const`
Return *Asset* assigned to the specified key or throws an error otherwise

`std::shared_ptr<Asset> get (const std::string &key)`
Return *Asset* assigned to the specified key or throws an error otherwise

`std::vector<std::shared_ptr<const Asset>> getAll () const`
Return All asset stored in the *AssetManager*

`std::vector<std::shared_ptr<Asset>> getAll ()`
Return All asset stored in the *AssetManager*

`std::size_t size () const`
Return Number of asset stored in the *AssetManager*

`void remove (const std::string &key)`
Removes asset with key
Parameters

- key: Key of asset to remove

`void serialize (Assets &serAssets, std::vector<std::uint8_t> &assetStorage) const`
Serializes.

class Assets
Subclassed by *dai::AssetsMutable*

class AssetsMutable : public *dai::Assets*

struct AssetView

class Buffer : public *dai::ADatatype*
#include <Buffer.hpp> Base message - buffer of binary data.
Subclassed by *dai::CameraControl*, *dai::ImageManipConfig*, *dai::ImgDetections*,
dai::ImgFrame, *dai::NNData*, *dai::SpatialImgDetections*, *dai::SpatialLocationCalculatorConfig*,
dai::SpatialLocationCalculatorData, *dai::SystemInformation*, *dai::Tracklets*

Public Functions

Buffer ()
Creates *Buffer* message.

`std::vector<std::uint8_t> &getData ()`
Return Reference to internal buffer

`void setData (std::vector<std::uint8_t> data)`
Parameters

- data: Copies data to internal buffer

class CallbackHandler

class CameraControl : public *dai::Buffer*
#include <CameraControl.hpp> *CameraControl* message Specifies various camera control commands like:

- Still capture
- Auto focus

- Anti banding
- Auto white balance
- Scene
- Effect
- ...

Public Functions

CameraControl ()

Construct *CameraControl* message.

void **setCaptureStill** (bool *capture*)
Set a command to capture a still image

void **setStartStreaming** ()
Set a command to start streaming

void **setStopStreaming** ()
Set a command to stop streaming

void **setAutoFocusMode** (*AutoFocusMode mode*)
Set a command to specify autofocus mode

void **setAutoFocusTrigger** ()
Set a command to trigger autofocus

void **setAutoFocusRegion** (uint16_t *startX*, uint16_t *startY*, uint16_t *width*, uint16_t *height*)
Set a command to specify focus region in pixels

Parameters

- *startX*: X coordinate of top left corner of region
- *startY*: Y coordinate of top left corner of region
- *width*: Region width
- *height*: Region height

void **setManualFocus** (uint8_t *lensPosition*)
Set a command to specify manual focus position

Parameters

- *lensPosition*: specify lens position 0..255

void **setAutoExposureEnable** ()
Set a command to enable auto exposure

void **setAutoExposureLock** (bool *lock*)
Set a command to specify lock auto exposure

Parameters

- *lock*: Auto exposure lock mode enabled or disabled

void **setAutoExposureRegion** (uint16_t *startX*, uint16_t *startY*, uint16_t *width*, uint16_t *height*)
Set a command to specify auto exposure region in pixels

Parameters

- *startX*: X coordinate of top left corner of region
- *startY*: Y coordinate of top left corner of region
- *width*: Region width
- *height*: Region height

void **setAutoExposureCompensation** (int8_t *compensation*)
Set a command to specify auto exposure compenstaion
Parameters

- *compensation*: Compensation value between -128..127

void **setAntiBandingMode** (*AntiBandingMode mode*)
Set a command to specify auto banding mode
Parameters

- *mode*: Auto banding mode to use

void **setManualExposure** (uint32_t *exposureTimeUs*, uint32_t *sensitivityIso*)
Set a command to manually specify exposure
Parameters

- *exposureTimeUs*: Exposure time in microseconds
- *sensitivityIso*: Sensitivity as ISO value

void **setAutoWhiteBalanceMode** (*AutoWhiteBalanceMode mode*)
Set a command to specify auto white balance mode
Parameters

- *mode*: Auto white balance mode to use

void **setAutoWhiteBalanceLock** (bool *lock*)
Set a command to specify auto white balance lock
Parameters

- *lock*: Auto white balance lock mode enabled or disabled

void **setBrightness** (uint16_t *value*)
Set a command to specify auto white balance lock
Parameters

- *lock*: Auto white balance lock mode enabled or disabled

void **setContrast** (uint16_t *value*)
Set a command to specify auto white balance lock
Parameters

- *lock*: Auto white balance lock mode enabled or disabled

void **setSaturation** (uint16_t *value*)
Set a command to specify saturation value
Parameters

- *value*: Saturation

void **setSharpness** (uint16_t *value*)
Set a command to specify sharpness value
Parameters

- *value*: Sharpness

void **setNoiseReductionStrength** (uint16_t *value*)
Set a command to specify noise reduction strength
Parameters

- *value*: Noise reduction strength

void **setLumaDenoise** (uint16_t *value*)
Set a command to specify luma denoise value
Parameters

- *value*: Luma denoise

void **setChromaDenoise** (uint16_t *value*)
Set a command to specify chroma denoise value
Parameters

- `value`: Chroma denoise

void **setSceneMode** (*SceneMode mode*)
Set a command to specify scene mode
Parameters

- `mode`: Scene mode

void **setEffectMode** (*EffectMode mode*)
Set a command to specify effect mode
Parameters

- `mode`: Effect mode

bool **getCaptureStill** () **const**
Check whether command to capture a still is set
Return True if capture still command is set

struct ChipTemperature
#include <ChipTemperature.hpp> Chip temperature information.
Multiple temperature measurement points and their average

Public Members

float **css**
CPU Subsystem.

float **mss**
Media Subsystem.

float **upa**
Shave Array.

float **dss**
DRAM Subsystem.

float **average**
Average of measurements.

struct ColorCameraProperties
#include <ColorCameraProperties.hpp> Specify ColorCamera options such as camera ID, ...

Public Types

enum SensorResolution
Select the camera sensor resolution
Values:

- enumerator THE_1080_P**
- enumerator THE_4_K**
- enumerator THE_12_MP**

enum ColorOrder
For 24 bit color these can be either RGB or BGR
Values:

- enumerator BGR**
- enumerator RGB**

Public Members

CameraBoardSocket **boardSocket** = *CameraBoardSocket::AUTO*

Which socket will color camera use

CameraImageOrientation **imageOrientation** = *CameraImageOrientation::AUTO*

Camera sensor image orientation / pixel readout

ColorOrder **colorOrder** = *ColorOrder::BGR*

For 24 bit color these can be either RGB or BGR

bool **interleaved** = true

Are colors interleaved (R1G1B1, R2G2B2, ...) or planar (R1R2..., G1G2..., B1B2)

bool **fp16** = false

Are values FP16 type (0.0 - 255.0)

uint32_t **previewHeight** = 300

Preview frame output height

uint32_t **previewWidth** = 300

Preview frame output width

int32_t **videoWidth** = AUTO

Preview frame output width

int32_t **videoHeight** = AUTO

Preview frame output height

int32_t **stillWidth** = AUTO

Preview frame output width

int32_t **stillHeight** = AUTO

Preview frame output height

SensorResolution **resolution** = *SensorResolution::THE_1080_P*

Select the camera sensor resolution

float **fps** = 30.0

Camera sensor FPS

float **sensorCropX** = AUTO

Initial sensor crop, -1 signifies center crop

bool **inputConfigSync** = false

Whether to wait for config at 'inputConfig' io

bool **previewKeepAspectRatio** = true

Whether to keep aspect ratio of input (video size) or not

struct CpuUsage

#include <CpuUsage.hpp> *CpuUsage* structure

Average usage in percent and time span of the average (since last query)

Public Members

float **average**
Average CPU usage, expressed with a normalized value (0-1)

int32_t **msTime**
Time span in which the average was calculated in milliseconds.

class DataInputQueue
#include <DataQueue.hpp> Access to send messages through XLink stream

Public Functions

void **setMaxDataSize** (std::size_t *maxSize*)
Sets maximum message size. If message is larger than specified, then an exception is issued.

Parameters

- *maxSize*: Maximum message size to add to queue

std::size_t **getMaxDataSize** ()
Gets maximum queue size.

Return Maximum message size

void **setBlocking** (bool *blocking*)
Sets queue behavior when full (*maxSize*)

Parameters

- *blocking*: Specifies if block or overwrite the oldest message in the queue

bool **getBlocking** () **const**
Gets current queue behavior when full (*maxSize*)

Return true if blocking, false otherwise

void **setMaxSize** (unsigned int *maxSize*)
Sets queue maximum size

Parameters

- *maxSize*: Specifies maximum number of messages in the queue

unsigned int **getMaxSize** (unsigned int *maxSize*) **const**
Gets queue maximum size

Return Maximum queue size

std::string **getName** () **const**
Gets queues name

Return Queue name

void **send** (const std::shared_ptr<RawBuffer> &*rawMsg*)
Adds a raw message to the queue, which will be picked up and sent to the device. Can either block if 'blocking' behavior is true or overwrite oldest

Parameters

- *rawMsg*: Message to add to the queue

void **send** (const std::shared_ptr<ADatatype> &*msg*)
Adds a message to the queue, which will be picked up and sent to the device. Can either block if 'blocking' behavior is true or overwrite oldest

Parameters

- *msg*: Message to add to the queue

void **send** (const *ADatatype* &msg)

Adds a message to the queue, which will be picked up and sent to the device. Can either block if 'blocking' behavior is true or overwrite oldest

Parameters

- msg: Message to add to the queue

bool **send** (const std::shared_ptr<*RawBuffer*> &rawMsg, std::chrono::milliseconds *timeout*)

Adds message to the queue, which will be picked up and sent to the device. Can either block until timeout if 'blocking' behavior is true or overwrite oldest

Parameters

- rawMsg: Message to add to the queue
- timeout: Maximum duration to block in milliseconds

bool **send** (const std::shared_ptr<*ADatatype*> &msg, std::chrono::milliseconds *timeout*)

Adds message to the queue, which will be picked up and sent to the device. Can either block until timeout if 'blocking' behavior is true or overwrite oldest

Parameters

- msg: Message to add to the queue
- timeout: Maximum duration to block in milliseconds

bool **send** (const *ADatatype* &msg, std::chrono::milliseconds *timeout*)

Adds message to the queue, which will be picked up and sent to the device. Can either block until timeout if 'blocking' behavior is true or overwrite oldest

Parameters

- msg: Message to add to the queue
- timeout: Maximum duration to block in milliseconds

class DataOutputQueue

#include <DataQueue.hpp> Access to receive messages coming from XLink stream

Public Types

using CallbackId = int

Alias for callback id.

Public Functions

void **setBlocking** (bool *blocking*)

Sets queue behavior when full (maxSize)

Parameters

- blocking: Specifies if block or overwrite the oldest message in the queue

bool **getBlocking** () const

Gets current queue behavior when full (maxSize)

Return true if blocking, false otherwise

void **setMaxSize** (unsigned int *maxSize*)

Sets queue maximum size

Parameters

- maxSize: Specifies maximum number of messages in the queue

unsigned int **getMaxSize** (unsigned int *maxSize*) const

Gets queue maximum size

Return Maximum queue size

std::string **getName** () **const**

Gets queues name

Return Queue name

CallbackId **addCallback** (std::function<void> std::string, std::shared_ptr<*ADatatype*>

>Adds a callback on message received

Return Callback id

Parameters

- *callback*: Callback function with queue name and message pointer

CallbackId **addCallback** (std::function<void> std::shared_ptr<*ADatatype*>

>Adds a callback on message received

Return Callback id

Parameters

- *callback*: Callback function with message pointer

CallbackId **addCallback** (std::function<void>

> *callback* Adds a callback on message received

Return Callback id

Parameters

- *callback*: Callback function without any parameters

bool **removeCallback** (*CallbackId* *callbackId*)

Removes a callback

Return true if callback was removed, false otherwise

Parameters

- *callbackId*: Id of callback to be removed

template<class **T**>

bool **has** ()

Check whether front of the queue has message of type T

Return true if queue isn't empty and the first element is of type T, false otherwise

bool **has** ()

Check whether front of the queue has a message (isn't empty)

Return true if queue isn't empty, false otherwise

template<class **T**>

std::shared_ptr<*T*> **tryGet** ()

Try to retrieve message T from queue. If message isn't of type T it returns nullptr

Return Message of type T or nullptr if no message available

std::shared_ptr<*ADatatype*> **tryGet** ()

Try to retrieve message from queue. If no message available, return immediately with nullptr

Return Message or nullptr if no message available

template<class **T**>

std::shared_ptr<*T*> **get** ()

Block until a message is available.

Return Message of type T or nullptr if no message available

std::shared_ptr<*ADatatype*> **get** ()

Block until a message is available.

Return Message or nullptr if no message available

```
template<class T>
std::shared_ptr<T> front ()
    Gets first message in the queue.
```

Return Message of type T or nullptr if no message available

```
std::shared_ptr<ADatatype> front ()
    Gets first message in the queue.
```

Return Message or nullptr if no message available

```
template<class T, typename Rep, typename Period>
std::shared_ptr<T> get (std::chrono::duration<Rep, Period> timeout, bool &hasTimedout)
    Block until a message is available with a timeout.
```

Return Message of type T otherwise nullptr if message isn't type T or timeout occurred

Parameters

- timeout: Duration for which the function should block
- [out] hasTimedout: Outputs true if timeout occurred, false otherwise

```
template<typename Rep, typename Period>
std::shared_ptr<ADatatype> get (std::chrono::duration<Rep, Period> timeout, bool &hasTimedout)
    Block until a message is available with a timeout.
```

Return Message of type T otherwise nullptr if message isn't type T or timeout occurred

Parameters

- timeout: Duration for which the function should block
- [out] hasTimedout: Outputs true if timeout occurred, false otherwise

```
template<class T>
std::vector<std::shared_ptr<T>> tryGetAll ()
    Try to retrieve all messages in the queue.
```

Return Vector of messages which can either be of type T or nullptr

```
std::vector<std::shared_ptr<ADatatype>> tryGetAll ()
    Try to retrieve all messages in the queue.
```

Return Vector of messages

```
template<class T>
std::vector<std::shared_ptr<T>> getAll ()
    Block until at least one message in the queue. Then return all messages from the queue.
```

Return Vector of messages which can either be of type T or nullptr

```
std::vector<std::shared_ptr<ADatatype>> getAll ()
    Block until at least one message in the queue. Then return all messages from the queue.
```

Return Vector of messages

```
template<class T, typename Rep, typename Period>
std::vector<std::shared_ptr<T>> getAll (std::chrono::duration<Rep, Period> timeout, bool &has-
                                     Timedout)
    Block for maximum timeout duration. Then return all messages from the queue.
```

Return Vector of messages which can either be of type T or nullptr

Parameters

- timeout: Maximum duration to block
- [out] hasTimedout: Outputs true if timeout occurred, false otherwise

```
template<typename Rep, typename Period>
```

`std::vector<std::shared_ptr<ADatatype>> getAll (std::chrono::duration<Rep, Period> timeout, bool &hasTimedout)`

Block for maximum timeout duration. Then return all messages from the queue.

Return Vector of messages

Parameters

- `timeout`: Maximum duration to block
- `[out] hasTimedout`: Outputs true if timeout occurred, false otherwise

struct DetectionNetworkProperties : public *dai::NeuralNetworkProperties*

#include <DetectionNetworkProperties.hpp> Properties for DetectionNetwork

Subclassed by *dai::SpatialDetectionNetworkProperties*

Public Members

DetectionNetworkType **nnFamily**

Generic Neural Network properties.

int **classes**

YOLO specific network properties.

class Device

#include <Device.hpp> Represents the DepthAI device with the methods to interact with it.

Public Functions

Device (const *Pipeline* &pipeline)

Connects to any available device with a DEFAULT_SEARCH_TIME timeout.

Parameters

- `pipeline`: - *Pipeline* to be executed on the device

Device (const *Pipeline* &pipeline, bool *usb2Mode*)

Connects to any available device with a DEFAULT_SEARCH_TIME timeout.

Parameters

- `pipeline`: - *Pipeline* to be executed on the device
- `usb2Mode`: - Boot device using USB2 mode firmware

Device (const *Pipeline* &pipeline, const char **pathToCmd*)

Connects to any available device with a DEFAULT_SEARCH_TIME timeout.

Parameters

- `pipeline`: - *Pipeline* to be executed on the device
- `pathToCmd`: - Path to custom device firmware

Device (const *Pipeline* &pipeline, const std::string &*pathToCmd*)

Connects to any available device with a DEFAULT_SEARCH_TIME timeout.

Parameters

- `pipeline`: - *Pipeline* to be executed on the device
- `pathToCmd`: - Path to custom device firmware

Device (const *Pipeline* &pipeline, const *DeviceInfo* &*devInfo*, bool *usb2Mode* = false)

Connects to device specified by *devInfo*.

Parameters

- `pipeline`: - *Pipeline* to be executed on the device
- `devInfo`: - *DeviceInfo* which specifies which device to connect to
- `usb2Mode`: - Boot device using USB2 mode firmware

Device (const *Pipeline* &pipeline, const *DeviceInfo* &devInfo, const char *pathToCmd)

Connects to device specified by devInfo.

Parameters

- pipeline: - *Pipeline* to be executed on the device
- devInfo: - *DeviceInfo* which specifies which device to connect to
- pathToCmd: - Path to custom device firmware

Device (const *Pipeline* &pipeline, const *DeviceInfo* &devInfo, const std::string &pathToCmd)

Connects to device specified by devInfo.

Parameters

- pipeline: - *Pipeline* to be executed on the device
- devInfo: - *DeviceInfo* which specifies which device to connect to
- usb2Mode: - Path to custom device firmware

~Device ()

Device destructor. Closes the connection and data queues.

bool **isPipelineRunning** ()

Checks if devices pipeline is already running

Return true if running, false otherwise

bool **startPipeline** ()

Starts the execution of the devices pipeline

Return true if pipeline started, false otherwise

void **setLogLevel** (*LogLevel* level)

Sets the devices logging severity level. This level affects which logs are transfered from device to host.

Parameters

- level: Logging severity

LogLevel **getLogLevel** ()

Gets current logging severity level of the device.

Return Logging severity level

void **setLogOutputLevel** (*LogLevel* level)

Sets logging level which decides printing level to standard output. If lower than setLogLevel, no messages will be printed

Parameters

- level: - Standard output printing severity

LogLevel **getLogOutputLevel** ()

Gets logging level which decides printing level to standard output.

Return Standard output printing severity

int **addLogCallback** (std::function<void> *LogMessage*

> callback) Add a callback for device logging. The callback will be called from a separate thread with the *LogMessage* being passed.

Return Id which can be used to later remove the callback

Parameters

- callback: - Callback to call whenever a log message arrives

bool **removeLogCallback** (int callbackId)

Removes a callback

Return true if callback was removed, false otherwise

Parameters

- `callbackId`: Id of callback to be removed

void **setSystemInformationLoggingRate** (float *rateHz*)

Sets rate of system information logging (“info” severity). Default 1Hz If parameter is less or equal to zero, then system information logging will be disabled

Parameters

- `rateHz`: Logging rate in Hz

float **getSystemInformationLoggingRate** ()

Gets current rate of system information logging (“info” severity) in Hz.

Return Logging rate in Hz

std::shared_ptr<*DataOutputQueue*> **getOutputQueue** (const std::string &*name*)

Gets an output queue corresponding to stream name. If it doesn’t exist it throws

Return Smart pointer to *DataOutputQueue*

Parameters

- `name`: Queue/stream name, created by XLinkOut node

std::shared_ptr<*DataOutputQueue*> **getOutputQueue** (const std::string &*name*, unsigned int *maxSize*, bool *blocking* = true)

Gets a queue corresponding to stream name, if it exists, otherwise it throws. Also sets queue options

Return Smart pointer to *DataOutputQueue*

Parameters

- `name`: Queue/stream name, set in XLinkOut node
- `maxSize`: Maximum number of messages in queue
- `blocking`: Queue behavior once full. True specifies blocking and false overwriting of oldest messages. Default: true

std::vector<std::string> **getOutputQueueNames** () const

Get all available output queue names

Return Vector of output queue names

std::shared_ptr<*DataInputQueue*> **getInputQueue** (const std::string &*name*)

Gets an input queue corresponding to stream name. If it doesn’t exist it throws

Return Smart pointer to *DataInputQueue*

Parameters

- `name`: Queue/stream name, set in XLinkIn node

std::shared_ptr<*DataInputQueue*> **getInputQueue** (const std::string &*name*, unsigned int *maxSize*, bool *blocking* = true)

Gets an input queue corresponding to stream name. If it doesn’t exist it throws. Also sets queue options

Return Smart pointer to *DataInputQueue*

Parameters

- `name`: Queue/stream name, set in XLinkOut node
- `maxSize`: Maximum number of messages in queue
- `blocking`: Queue behavior once full. True: blocking, false: overwriting of oldest messages. Default: true

std::vector<std::string> **getInputQueueNames** () const

Get all available input queue names

Return Vector of input queue names

```
std::vector<std::string> getQueueEvents (const      std::vector<std::string>      &queue-
                                         Names,      std::size_t      maxNumEvents      =
                                         std::numeric_limits<std::size_t>::max(),
                                         std::chrono::microseconds      timeout      =
                                         std::chrono::microseconds(-1))
```

Gets or waits until any of specified queues has received a message

Return Names of queues which received messages first

Parameters

- `queueNames`: Names of queues for which to block
- `maxNumEvents`: Maximum number of events to remove from queue - Default is unlimited
- `timeout`: Timeout after which return regardless. If negative then wait is indefinite - Default is -1

```
std::vector<std::string> getQueueEvents (std::string      queueName,      std::size_t      maxNu-
                                         mEvents      =      std::numeric_limits<std::size_t>::max(),
                                         std::chrono::microseconds      timeout      =
                                         std::chrono::microseconds(-1))
```

Gets or waits until specified queue has received a message

Return Names of queues which received messages first

Parameters

- `queueName`: Name of queues for which to wait for
- `maxNumEvents`: Maximum number of events to remove from queue. Default is unlimited
- `timeout`: Timeout after which return regardless. If negative then wait is indefinite. Default is -1

```
std::vector<std::string> getQueueEvents (std::size_t      maxNumEvents      =
                                         std::numeric_limits<std::size_t>::max(),
                                         std::chrono::microseconds      timeout      =
                                         std::chrono::microseconds(-1))
```

Gets or waits until any any queue has received a message

Return Names of queues which received messages first

Parameters

- `maxNumEvents`: Maximum number of events to remove from queue. Default is unlimited
- `timeout`: Timeout after which return regardless. If negative then wait is indefinite. Default is -1

```
std::string getQueueEvent (const      std::vector<std::string>      &queueNames,
                           std::chrono::microseconds      timeout = std::chrono::microseconds(-1))
```

Gets or waits until any of specified queues has received a message

Return Queue name which received a message first

Parameters

- `queueNames`: Names of queues for which to wait for
- `timeout`: Timeout after which return regardless. If negative then wait is indefinite. Default is -1

```
std::string getQueueEvent (std::string      queueName,      std::chrono::microseconds      timeout      =
                           std::chrono::microseconds(-1))
```

Gets or waits until specified queue has received a message

Return Queue name which received a message

Parameters

- `queueNames`: Name of queues for which to wait for
- `timeout`: Timeout after which return regardless. If negative then wait is indefinite. Default is -1

std::string **getQueueEvent** (std::chrono::microseconds *timeout* = std::chrono::microseconds(-1))

Gets or waits until any queue has received a message

Return Queue name which received a message

Parameters

- **timeout**: Timeout after which return regardless. If negative then wait is indefinite. Default is -1

MemoryInfo **getDdrMemoryUsage** ()

Retrieves current DDR memory information from device

Return Used, remaining and total ddr memory

MemoryInfo **getCmxMemoryUsage** ()

Retrieves current CMX memory information from device

Return Used, remaining and total cmx memory

MemoryInfo **getLeonCssHeapUsage** ()

Retrieves current CSS Leon CPU heap information from device

Return Used, remaining and total heap memory

MemoryInfo **getLeonMssHeapUsage** ()

Retrieves current MSS Leon CPU heap information from device

Return Used, remaining and total heap memory

ChipTemperature **getChipTemperature** ()

Retrieves current chip temperature as measured by device

Return Temperature of various onboard sensors

CpuUsage **getLeonCssCpuUsage** ()

Retrieves average CSS Leon CPU usage

Return Average CPU usage and sampling duration

CpuUsage **getLeonMssCpuUsage** ()

Retrieves average MSS Leon CPU usage

Return Average CPU usage and sampling duration

void **close** ()

Explicitly closes connection to device.

Note This function does not need to be explicitly called as destructor closes the device automatically

bool **isClosed** () **const**

Is the device already closed (or disconnected)

Public Static Functions

template<typename **Rep**, typename **Period**>

std::tuple<bool, *DeviceInfo*> **getAnyAvailableDevice** (std::chrono::duration<*Rep*, *Period*> *timeout*)

Waits for any available device with a timeout

Return a tuple of bool and *DeviceInfo*. Bool specifies if device was found. *DeviceInfo* specifies the found device

Parameters

- **timeout**: - duration of time to wait for the any device

`std::tuple<bool, DeviceInfo> getAnyAvailableDevice ()`

Gets any available device

Return a tuple of bool and *DeviceInfo*. Bool specifies if device was found. *DeviceInfo* specifies the found device

`std::tuple<bool, DeviceInfo> getFirstAvailableDevice ()`

Gets first available device. *Device* can be either in XLINK_UNBOOTED or XLINK_BOOTLOADER state

Return a tuple of bool and *DeviceInfo*. Bool specifies if device was found. *DeviceInfo* specifies the found device

`std::tuple<bool, DeviceInfo> getDeviceByMxId (std::string mxId)`

Finds a device by MX ID. Example: 14442C10D13EABCE00

Return a tuple of bool and *DeviceInfo*. Bool specifies if device was found. *DeviceInfo* specifies the found device

Parameters

- *mxId*: - MyraidX ID which uniquely specifies a device

`std::vector<DeviceInfo> getAllAvailableDevices ()`

Returns all connected devices

Return vector of connected devices

`std::vector<std::uint8_t> getEmbeddedDeviceBinary (bool usb2Mode, Open-
VINO::Version version =
Pipeline::DEFAULT_OPENVINO_VERSION)`

Gets device firmware binary for a specific *OpenVINO* version

Return firmware binary

Parameters

- *usb2Mode*: - USB2 mode firmware
- *version*: - Version of *OpenVINO* which firmware will support

Public Static Attributes

`constexpr std::chrono::seconds DEFAULT_SEARCH_TIME = {3}`

Default search time for constructors which discover devices.

`constexpr std::size_t EVENT_QUEUE_MAXIMUM_SIZE = {2048}`

Maximum number of elements in event queue.

`constexpr float DEFAULT_SYSTEM_INFORMATION_LOGGING_RATE_HZ = {1.0f}`

Default rate at which system information is logged.

`class DeviceBootloader`

`#include <DeviceBootloader.hpp>` Represents the DepthAI bootloader with the methods to interact with it.

Public Functions

DeviceBootloader (const *DeviceInfo* &devInfo)

Connects to or boots device in bootloader mode depending on devInfo state.

Parameters

- devInfo: *DeviceInfo* of which to boot or connect to

DeviceBootloader (const *DeviceInfo* &devInfo, const std::string &pathToBootloader)

Connects to or boots device in bootloader mode depending on devInfo state with a custom bootloader firmware.

Parameters

- devInfo: *DeviceInfo* of which to boot or connect to
- pathToBootloader: Custom bootloader firmware to boot

DeviceBootloader (const *DeviceInfo* &devInfo, const char *pathToBootloader)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

std::tuple<bool, std::string> **flash** (std::function<void> float
> progressCallback, *Pipeline* &pipeline) Flashes a give pipeline to the board.

Parameters

- progressCallback: Callback that sends back a value between 0..1 which signifies current flashing progress
- pipeline: *Pipeline* to flash to the board

std::tuple<bool, std::string> **flashDepthaiApplicationPackage** (std::function<void> float
> progressCallback, std::vector<uint8_t> package) Flashes a specific depthai application package that was generated using createDepthaiApplicationPackage or saveDepthaiApplicationPackage

Parameters

- progressCallback: Callback that sends back a value between 0..1 which signifies current flashing progress
- package: Depthai application package to flash to the board

std::tuple<bool, std::string> **flashBootloader** (std::function<void> float
> progressCallback, std::string path = "") Flashes bootloader to the current board

Parameters

- progressCallback: Callback that sends back a value between 0..1 which signifies current flashing progress
- path: Optional parameter to custom bootloader to flash

Version **getVersion** ()

Return *Version* of current running bootloader

bool **isEmbeddedVersion** ()

Return True whether the bootloader running is flashed or booted by library

void **close** ()

Explicitly closes connection to device.

Note This function does not need to be explicitly called as destructor closes the device automatically

bool **isClosed** () const

Is the device already closed (or disconnected)

Public Static Functions

`std::tuple<bool, DeviceInfo> getFirstAvailableDevice ()`
 Searches for connected devices in either UNBOOTED or BOOTLOADER states and returns first available.
Return Tuple of boolean and *DeviceInfo*. If found boolean is true and *DeviceInfo* describes the device. Otherwise false

`std::vector<DeviceInfo> getAllAvailableDevices ()`
 Searches for connected devices in either UNBOOTED or BOOTLOADER states.
Return Vector of all found devices

`std::vector<uint8_t> createDepthaiApplicationPackage (Pipeline &pipeline, std::string pathToCmd = "")`
 Creates application package which can be flashed to depthai device.
Return Depthai application package
Parameters

- pipeline: *Pipeline* from which to create the application package
- pathToCmd: Optional path to custom device firmware

`void saveDepthaiApplicationPackage (std::string path, Pipeline &pipeline, std::string pathToCmd = "")`
 Saves application package to a file which can be flashed to depthai device.
Parameters

- path: Path where to save the application package
- pipeline: *Pipeline* from which to create the application package
- pathToCmd: Optional path to custom device firmware

Version `getEmbeddedBootloaderVersion ()`
Return Embedded bootloader version

`std::vector<std::uint8_t> getEmbeddedBootloaderBinary ()`
Return Embedded bootloader binary

struct Version
#include <DeviceBootloader.hpp> Bootloader version structure.

Public Functions

Version (`const` std::string &v)
 Construct *Version* from string.

Version (unsigned *major*, unsigned *minor*, unsigned *patch*)
 Construct *Version* major, minor and patch numbers.

std::string **toString** () `const`
 Convert *Version* to string.

struct DeviceInfo
#include <XLinkConnection.hpp> Describes a connected device

struct GlobalProperties
#include <GlobalProperties.hpp> Specify properties which apply for whole pipeline

Public Members

double **leonCssFrequencyHz** = 700 * 1000 * 1000

Set frequency of Leon OS - Increasing can improve performance, at the cost of higher power draw

double **leonMssFrequencyHz** = 700 * 1000 * 1000

Set frequency of Leon RT - Increasing can improve performance, at the cost of higher power draw

class ImageManipConfig: public *dai::Buffer*

#include <ImageManipConfig.hpp> *ImageManipConfig* message. Specifies image manipulation options like:

- Crop
- Resize
- Warp
- ...

Public Functions

ImageManipConfig()

Construct *ImageManipConfig* message.

void **setCropRect** (float *xmin*, float *ymin*, float *xmax*, float *ymax*)

Specifies crop with rectangle with normalized values (0..1)

Parameters

- *xmin*: Top left X coordinate of rectangle
- *ymin*: Top left Y coordinate of rectangle
- *xmax*: Bottom right X coordinate of rectangle
- *ymax*: Bottom right Y coordinate of rectangle

void **setCropRotatedRect** (*RotatedRect* *rr*, bool *normalizedCoords* = true)

Specifies crop with rotated rectangle. Optionally as non normalized coordinates

Parameters

- *rr*: Rotated rectangle which specifies crop
- *normalizedCoords*: If true coordinates are in normalized range (0..1) otherwise absolute

void **setCenterCrop** (float *ratio*, float *whRatio* = 1.0f)

Specifies a centered crop.

Parameters

- *ratio*: Ratio between input image and crop region (0..1)
- *whRatio*: Crop region aspect ratio - 1 equals to square, 1.7 equals to 16:9, ...

void **setWarpTransformFourPoints** (std::vector<*Point2f*> *pt*, bool *normalizedCoords*)

Specifies warp by supplying 4 points in either absolute or normalized coordinates

Parameters

- *pt*: 4 points specifying warp
- *normalizedCoords*: If true *pt* is interpreted as normalized, absolute otherwise

void **setWarpTransformMatrix3x3** (std::vector<float> *mat*)

Specifies warp with a 3x3 matrix

Parameters

- *mat*: 3x3 matrix

void **setWarpBorderReplicatePixels** ()

Specifies that warp replicates border pixels

void **setWarpBorderFillColor** (int *red*, int *green*, int *blue*)

Specifies fill color for border pixels. Example:

- `setWarpBorderFillColor(255,255,255)` -> white
- `setWarpBorderFillColor(0,0,255)` -> blue

Parameters

- *red*: Red component
- *green*: Green component
- *blue*: Blue component

void **setRotationDegrees** (float *deg*)

Specifies clockwise rotation in degrees

Parameters

- *deg*: Rotation in degrees

void **setRotationRadians** (float *rad*)

Specifies clockwise rotation in radians

Parameters

- *rad*: Rotation in radians

void **setResize** (int *w*, int *h*)

Specifies output image size. After crop stage the image will be stretched to fit.

Parameters

- *w*: Width in pixels
- *h*: Height in pixels

void **setResizeThumbnail** (int *w*, int *h*, int *bgRed* = 0, int *bgGreen* = 0, int *bgBlue* = 0)

Specifies output image size. After crop stage the image will be resized by preserving aspect ration.

Optionally background can be specified.

Parameters

- *w*: Width in pixels
- *h*: Height in pixels
- *bgRed*: Red component
- *bgGreen*: Green component
- *bgBlue*: Blue component

void **setFrameType** (*ImgFrame::Type* *name*)

Specify output frame type.

Parameters

- *name*: Frame type

void **setHorizontalFlip** (bool *flip*)

Specify horizontal flip

Parameters

- *flip*: True to enable flip, false otherwise

void **setReusePreviousImage** (bool *reuse*)

Instruct ImageManip to not remove current image from its queue and use the same for next message.

Parameters

- *reuse*: True to enable reuse, false otherwise

void **setSkipCurrentImage** (bool *skip*)

Instructs ImageManip to skip current image and wait for next in queue.

Parameters

- *skip*: True to skip current image, false otherwise

void **setKeepAspectRatio** (bool *keep*)

Specifies to whether to keep aspect ratio or not

```
float getCropXMin () const  
    Return Top left X coordinate of crop region  
  
float getCropYMin () const  
    Return Top left Y coordinate of crop region  
  
float getCropXMax () const  
    Return Bottom right X coordinate of crop region  
  
float getCropYMax () const  
    Return Bottom right Y coordinate of crop region  
  
int getResizeWidth () const  
    Return Output image width  
  
int getResizeHeight () const  
    Return Output image height  
  
bool isResizeThumbnail () const  
    Return True if resize thumbnail mode is set, false otherwise  
  
struct ImageManipProperties  
    #include <ImageManipProperties.hpp> Specify ImageManip options
```

Public Members

```
RawImageManipConfig initialConfig  
    Initial configuration for ImageManip node.  
  
bool inputConfigSync = false  
    Whether to wait for config at 'inputConfig' IO.  
  
int outputFrameSize = 1 * 1024 * 1024  
    Maximum output frame size in bytes (eg: 300x300 BGR image -> 300*300*3 bytes)  
  
int numFramesPool = 4  
    Num frames in output pool.  
  
struct ImgDetection  
    Subclassed by dai::SpatialImgDetection  
  
class ImgDetections : public dai::Buffer  
    #include <ImgDetections.hpp> ImgDetections message. Carries normalized detection results
```

Public Functions

```
ImgDetections ()  
    Construct ImgDetections message.
```

Public Members

`std::vector<ImgDetection> &detections`
 Detections.

class `ImgFrame` : **public** *dai::Buffer*
#include <ImgFrame.hpp> *ImgFrame* message. Carries image data and metadata.

Public Functions

ImgFrame ()
 Construct *ImgFrame* message. *Timestamp* is set to now

`std::chrono::time_point<std::chrono::steady_clock, std::chrono::steady_clock::duration> getTimestamp ()` **const**
 Retrieves image timestamp related to steady_clock / time.monotonic

`unsigned int getInstanceNum ()` **const**
 Retrieves instance number

`unsigned int getCategory ()` **const**
 Retrieves image category

`unsigned int getSequenceNum ()` **const**
 Retrieves image sequence number

`unsigned int getWidth ()` **const**
 Retrieves image width in pixels

`unsigned int getHeight ()` **const**
 Retrieves image height in pixels

Type `getType ()` **const**
 Retrieves image type

`void setTimestamp (std::chrono::time_point<std::chrono::steady_clock, std::chrono::steady_clock::duration> timestamp)`
 Specifies current timestamp, related to steady_clock / time.monotonic

`void setInstanceNum (unsigned int instance)`
 Instance number relates to the origin of the frame (which camera)

Parameters

- *instance*: Instance number

`void setCategory (unsigned int category)`

Parameters

- *category*: Image category

`void setSequenceNum (unsigned int seq)`
 Specifies sequence number

Parameters

- *seq*: Sequence number

`void setWidth (unsigned int width)`
 Specifies frame width

Parameters

- *width*: frame width

void **setHeight** (unsigned int)
Specifies frame height

Parameters

- width: frame height

void **setType** (*Type* type)
Specifies frame type, RGB, BGR, ...

Parameters

- type: Type of image

void **setFrame** (cv::Mat frame)

Copies cv::Mat data to *ImgFrame* buffer

Note This API only available if OpenCV support enabled

Parameters

- frame: Input cv::Mat frame from which to copy the data

cv::Mat **getFrame** (bool copy = false)

Retrieves data as cv::Mat with specified width, height and type

Note This API only available if OpenCV support enabled

Return cv::Mat with corresponding to *ImgFrame* parameters

Parameters

- copy: If false only a reference to data is made, otherwise a copy

cv::Mat **getCvFrame** ()

Retrieves cv::Mat suitable for use in common opencv functions. *ImgFrame* is converted to color BGR interleaved or grayscale depending on type.

Note This API only available if OpenCV support enabled

A copy is always made

Return cv::Mat for use in opencv functions

template<typename T>

class **LockingQueue**

struct **LogMessage**

struct **MemoryInfo**

#include <MemoryInfo.hpp> *MemoryInfo* structure

Free, remaining and total memory stats

struct **MonoCameraProperties**

#include <MonoCameraProperties.hpp> Specify MonoCamera options such as camera ID, ...

Public Types

enum **SensorResolution**

Select the camera sensor resolution: 1280×720, 1280×800, 640×400

Values:

enumerator **THE_720_P**

enumerator **THE_800_P**

enumerator **THE_400_P**

Public Members

CameraBoardSocket **boardSocket** = *CameraBoardSocket::AUTO*

Which socket will mono camera use

CameraImageOrientation **imageOrientation** = *CameraImageOrientation::AUTO*

Camera sensor image orientation / pixel readout

SensorResolution **resolution** = *SensorResolution::THE_720_P*

Select the camera sensor resolution

float **fps** = 30.0

Camera sensor FPS

struct MyConsumerProperties

#include <MyConsumerProperties.hpp> Specify message and processor placement of MyConsumer node

Public Members

ProcessorType **processorPlacement**

On which processor the node will be placed

struct MyProducerProperties

#include <MyProducerProperties.hpp> Specify message and processor placement of MyProducer node

Public Members

tl::optional<std::string> **message**

Message to be sent forward

ProcessorType **processorPlacement** = *ProcessorType::LOS*

On which processor the node will be placed

struct NeuralNetworkProperties

#include <NeuralNetworkProperties.hpp> Specify NeuralNetwork options such as blob path, ...

Subclassed by *dai::DetectionNetworkProperties*

Public Members

tl::optional<std::uint32_t> **blobSize**

Blob binary size in bytes

std::string **blobUri**

Uri which points to blob

std::uint32_t **numFrames** = 8

Number of available output tensors in pool

std::uint32_t **numThreads** = 0

Number of threads to create for running inference. 0 = auto

std::uint32_t **numNCEPerThread** = 0

Number of NCE (Neural Compute Engine) per inference thread. 0 = auto

class NNData : public dai::Buffer

#include <NNData.hpp> *NNData* message. Carries tensors and their metadata

Public Functions

NNData ()

Construct *NNData* message.

void **setLayer** (**const** std::string &*name*, std::vector<std::uint8_t> *data*)

Set a layer with datatype U8.

Parameters

- *name*: Name of the layer
- *data*: Data to store

void **setLayer** (**const** std::string &*name*, **const** std::vector<int> &*data*)

Set a layer with datatype U8. Integers are casted to bytes.

Parameters

- *name*: Name of the layer
- *data*: Data to store

void **setLayer** (**const** std::string &*name*, std::vector<float> *data*)

Set a layer with datatype FP16. Float values are converted to FP16.

Parameters

- *name*: Name of the layer
- *data*: Data to store

void **setLayer** (**const** std::string &*name*, std::vector<double> *data*)

Set a layer with datatype FP16. Double values are converted to FP16.

Parameters

- *name*: Name of the layer
- *data*: Data to store

std::vector<std::string> **getAllLayerNames** () **const**

Return Names of all layers added

std::vector<*TensorInfo*> **getAllLayers** () **const**

Return All layers and their information

bool **getLayer** (**const** std::string &*name*, *TensorInfo* &*tensor*) **const**

Retrieve layers tensor information

Return True if layer exists, false otherwise

Parameters

- *name*: Name of the layer
- [out] *tensor*: Outputs tensor information of that layer

bool **hasLayer** (**const** std::string &*name*) **const**

Checks if given layer exists

Return True if layer exists, false otherwise

Parameters

- *name*: Name of the layer

bool **getLayerDatatype** (**const** std::string &*name*, *TensorInfo::DataType* &*datatype*) **const**

Retrieve datatype of a layers tensor

Return True if layer exists, false otherwise

Parameters

- *name*: Name of the layer
- [out] *datatype*: Datatype of layers tensor

std::vector<std::uint8_t> **getLayerUInt8** (**const** std::string &*name*) **const**

Convenience function to retrieve U8 data from layer

Return U8 binary data

Parameters

- name: Name of the layer

std::vector<float> **getLayerFp16** (const std::string &name) const
 Convenience function to retrieve float values from layers FP16 tensor

Return Float data

Parameters

- name: Name of the layer

std::vector<std::int32_t> **getLayerInt32** (const std::string &name) const
 Convenience function to retrieve INT32 values from layers tensor

Return INT32 data

Parameters

- name: Name of the layer

std::vector<std::uint8_t> **getFirstLayerUInt8** () const
 Convenience function to retrieve U8 data from first layer

Return U8 binary data

std::vector<float> **getFirstLayerFp16** () const
 Convenience function to retrieve float values from first layers FP16 tensor

Return Float data

std::vector<std::int32_t> **getFirstLayerInt32** () const
 Convenience function to retrieve INT32 values from first layers tensor

Return INT32 data

class Node

#include <Node.hpp> Abstract Node.

Subclassed by *dai::node::ColorCamera*, *dai::node::ImageManip*, *dai::node::MonoCamera*,
dai::node::MyProducer, *dai::node::NeuralNetwork*, *dai::node::ObjectTracker*,
dai::node::SpatialLocationCalculator, *dai::node::SPIOut*, *dai::node::StereoDepth*,
dai::node::SystemLogger, *dai::node::VideoEncoder*, *dai::node::XLinkIn*, *dai::node::XLinkOut*

Public Types

using Id = std::int64_t

Node identifier. Unique for every node on a single *Pipeline*.

Public Functions

std::string **getName** () const = 0
 Retrieves nodes name.

std::vector<Output> **getOutputs** () = 0
 Retrieves all nodes outputs.

std::vector<Input> **getInputs** () = 0
 Retrieves all nodes inputs.

std::vector<std::shared_ptr<Asset>> **getAssets** ()
 Retrieves all nodes assets.

Public Members

const *Id* id
Id of node.

struct Connection
#include <Node.hpp> *Connection* between an Input and Output.

struct NodeConnectionSchema
#include <NodeConnectionSchema.hpp> Specifies a connection between nodes IOs

struct NodeIoInfo

struct NodeObjInfo

struct ObjectTrackerProperties
#include <ObjectTrackerProperties.hpp> Properties for ObjectTracker

class OpenVINO
#include <OpenVINO.hpp> Support for basic *OpenVINO* related actions like version identification of neural network blobs,...

Public Types

enum Version
OpenVINO Version supported version information.

Values:

enumerator VERSION_2020_1

enumerator VERSION_2020_2

enumerator VERSION_2020_3

enumerator VERSION_2020_4

enumerator VERSION_2021_1

enumerator VERSION_2021_2

enumerator VERSION_2021_3

Public Static Functions

std::vector<*Version*> getVersions ()
Return Supported versions

std::string getVersionName (*Version* version)
Returns string representation of a given version
Return Name of a given version
Parameters

- *version*: *OpenVINO* version

***Version* parseVersionName (const std::string &versionString)**
Creates Version from string representation. Throws if not possible.
Return Version object if successful
Parameters

- *versionString*: Version as string

`std::vector<Version> getBlobSupportedVersions (std::uint32_t majorVersion, std::uint32_t minorVersion)`

Returns a list of potentially supported versions for a specified blob major and minor versions.

Return Vector of potentially supported versions

Parameters

- majorVersion: Major version from [OpenVINO](#) blob
- minorVersion: Minor version from [OpenVINO](#) blob

[Version](#) `getBlobLatestSupportedVersion (std::uint32_t majorVersion, std::uint32_t minorVersion)`

Returns latest potentially supported version by a given blob version.

Return Latest potentially supported version

Parameters

- majorVersion: Major version from [OpenVINO](#) blob
- minorVersion: Minor version from [OpenVINO](#) blob

`bool areVersionsBlobCompatible (Version v1, Version v2)`

Checks whether two blob versions are compatible

`template<typename T>`

`class Pimpl`

`class Pipeline`

#include <Pipeline.hpp> Represents the pipeline, set of nodes and connections between them.

Public Functions

`Pipeline ()`

Constructs a new pipeline

[GlobalProperties](#) `getGlobalProperties () const`

Return Global properties of current pipeline

[PipelineSchema](#) `getPipelineSchema ()`

Return [Pipeline](#) schema

`template<class N>`

`std::shared_ptr<N> create ()`

Adds a node to pipeline.

[Node](#) is specified by template argument N

`void remove (std::shared_ptr<Node> node)`

Removes a node from pipeline.

`std::vector<std::shared_ptr<const Node>> getAllNodes () const`

Get a vector of all nodes.

`std::vector<std::shared_ptr<Node>> getAllNodes ()`

Get a vector of all nodes.

`std::shared_ptr<const Node> getNode (Node::Id id) const`

Get node with id if it exists, nullptr otherwise.

`std::shared_ptr<Node> getNode (Node::Id id)`

Get node with id if it exists, nullptr otherwise.

`std::vector<Node::Connection> getConnections () const`

Get all connections.

const *NodeConnectionMap* &**getConnectionMap** () **const**

Get a reference to internal connection representation.

const *NodeMap* &**getNodeMap** () **const**

Get a reference to internal node map.

void link (**const** *Node*::Output &*out*, **const** *Node*::Input &*in*)

Link output to an input. Both nodes must be on the same pipeline

Throws an error if they aren't or cannot be connected

Parameters

- *out*: Nodes output to connect from
- *in*: Nodes input to connect to

void unlink (**const** *Node*::Output &*out*, **const** *Node*::Input &*in*)

Unlink output from an input.

Throws an error if link doesn't exists

Parameters

- *out*: Nodes output to unlink from
- *in*: Nodes input to unlink to

AssetManager **getAllAssets** () **const**

Get assets on the pipeline includes nodes assets.

const *AssetManager* &**getAssetManager** () **const**

Get pipelines *AssetManager* as reference.

AssetManager &**getAssetManager** ()

Get pipelines *AssetManager* as reference.

void setOpenVINOVersion (*OpenVINO*::Version *version*)

Set a specific *OpenVINO* version to use with this pipeline.

Public Static Attributes

constexpr auto **DEFAULT_OPENVINO_VERSION** = *PipelineImpl*::DEFAULT_OPENVINO_VERSION

Default *Pipeline* opencvino version.

class PipelineImpl

struct PipelineSchema

#include <PipelineSchema.hpp> Specifies whole pipeline, nodes, properties and connections between nodes IOs

struct Point2f

#include <Point2f.hpp> *Point2f* structure

x and y coordinates that define a 2D point.

struct Point3f

#include <Point3f.hpp> *Point3f* structure

x,y,z coordinates that define a 3D point.

struct RawBuffer

Subclassed by *dai::RawCameraControl*, *dai::RawImageManipConfig*,
dai::RawImgDetections, *dai::RawImgFrame*, *dai::RawNNData*, *dai::RawSpatialImgDetections*,
dai::RawSpatialLocationCalculatorConfig, *dai::RawSpatialLocations*, *dai::RawSystemInformation*,
dai::RawTracklets

```
struct RawCameraControl : public dai::RawBuffer
```

Public Members

```
uint8_t lensPosition = 0
```

Lens/VCM position, range: 0..255. Used with `autoFocusMode = OFF`. With current IMX378 modules:

- max 255: macro focus, at 8cm distance
- infinite focus at about 120..130 (may vary from module to module)
- lower values lead to out-of-focus (lens too close to the sensor array)

```
struct ManualExposureParams
```

```
struct RegionParams
```

```
struct RawImageManipConfig : public dai::RawBuffer
```

```
struct CropConfig
```

```
struct CropRect
```

```
struct FormatConfig
```

```
struct ResizeConfig
```

Public Members

```
bool keepAspectRatio = true
```

Whether to keep aspect ratio of input or not

```
struct RawImgDetections : public dai::RawBuffer
```

```
struct RawImgFrame : public dai::RawBuffer
```

```
struct Specs
```

```
struct RawNNData : public dai::RawBuffer
```

```
struct RawSpatialImgDetections : public dai::RawBuffer
```

```
struct RawSpatialLocationCalculatorConfig : public dai::RawBuffer
```

```
struct RawSpatialLocations : public dai::RawBuffer
```

```
struct RawSystemInformation : public dai::RawBuffer
```

`#include <RawSystemInformation.hpp>` System information of device

Memory usage, cpu usage and chip temperature

Public Members

MemoryInfo **ddrMemoryUsage**
DDR memory usage.

MemoryInfo **cmxMemoryUsage**
CMX memory usage.

MemoryInfo **leonCssMemoryUsage**
LeonCss heap usage.

MemoryInfo **leonMssMemoryUsage**
LeonMss heap usage.

CpuUsage **leonCssCpuUsage**
LeonCss cpu usage.

CpuUsage **leonMssCpuUsage**
LeonMss cpu usage.

ChipTemperature **chipTemperature**
Chip temperatures.

struct RawTracklets : public *dai::RawBuffer*

struct Rect

#include <Rect.hpp> *Rect* structure

x,y coordinates together with width and height that define a rectangle. Can be either normalized [0,1] or absolute representation.

Public Functions

Point2f **topLeft () const**
The top-left corner.

Point2f **bottomRight () const**
The bottom-right corner

Size2f **size () const**
Size (width, height) of the rectangle

float **area () const**
Area (width*height) of the rectangle

bool **empty () const**
True if rectangle is empty.

bool **contains (const *Point2f* &pt) const**
Checks whether the rectangle contains the point.

bool **isNormalized () const**
Whether rectangle is normalized (coordinates in [0,1] range) or not.

Rect **denormalize** (int *width*, int *height*)
Denormalize rectangle.

Parameters

- *width*: Destination frame width.
- *height*: Destination frame height.

Rect **normalize** (int *width*, int *height*)
Normalize rectangle.

Parameters

- **width**: Source frame width.
- **height**: Source frame height.

```
struct RotatedRect
```

Public Members

float **angle**
degrees, increasing clockwise

```
struct Size2f
```

```
struct SpatialDetectionNetworkProperties : public dai::DetectionNetworkProperties
#include <SpatialDetectionNetworkProperties.hpp> Properties for SpatialDetectionNetwork
```

```
struct SpatialImgDetection : public dai::ImgDetection
#include <RawSpatialImgDetections.hpp> Spatial image detection structure
```

Contains image detection results together with spatial location data.

```
class SpatialImgDetections : public dai::Buffer
#include <SpatialImgDetections.hpp> SpatialImgDetections message. Carries detection results together
with spatial location data
```

Public Functions

```
SpatialImgDetections ()
Construct SpatialImgDetections message.
```

Public Members

std::vector<SpatialImgDetection> &**detections**
Detection results.

```
class SpatialLocationCalculatorConfig : public dai::Buffer
#include <SpatialLocationCalculatorConfig.hpp> SpatialLocationCalculatorConfig message. Carries
ROI (region of interest) and threshold for depth calculation
```

Public Functions

```
SpatialLocationCalculatorConfig ()
Construct SpatialLocationCalculatorConfig message.
```

```
void setROIs (std::vector<SpatialLocationCalculatorConfigData> ROIs)
Set a vector of ROIs as configuration data.
```

Parameters

- **ROIs**: Vector of configuration parameters for ROIs (region of interests)

```
void addROI (SpatialLocationCalculatorConfigData &ROI)
Add a new ROI to configuration data.
```

Parameters

- **roi**: Configuration parameters for ROI (region of interest)

```
std::vector<SpatialLocationCalculatorConfigData> getConfigData () const
Retrieve configuration data for SpatialLocationCalculator
```

Return Vector of configuration parameters for ROIs (region of interests)

struct SpatialLocationCalculatorConfigData

struct SpatialLocationCalculatorConfigThresholds

#include <RawSpatialLocationCalculatorConfig.hpp> Spatial location configuration thresholds structure

Contains configuration data for lower and upper threshold in millimeters for ROI. Values outside of threshold range will be ignored when calculating spatial coordinates from depth map.

class SpatialLocationCalculatorData : public dai::Buffer

#include <SpatialLocationCalculatorData.hpp> *SpatialLocationCalculatorData* message. Carries spatial information (X,Y,Z) and their configuration parameters

Public Functions

SpatialLocationCalculatorData ()

Construct *SpatialLocationCalculatorData* message.

std::vector<SpatialLocations> &getSpatialLocations () const

Retrieve configuration data for *SpatialLocationCalculatorData*.

Return Vector of spatial location data, carrying spatial information (X,Y,Z)

struct SpatialLocationCalculatorProperties

#include <SpatialLocationCalculatorProperties.hpp> Specify SpatialLocationCalculator options

Public Members

bool **inputConfigSync** = false

Whether to wait for config at 'inputConfig' IO.

struct SpatialLocations

#include <RawSpatialLocations.hpp> Spatial location information structure

Contains configuration data, average depth for the calculated ROI on depth map. Together with spatial coordinates: x,y,z relative to the center of depth map. Units are in millimeters.

struct SPIOutProperties

#include <SPIOutProperties.hpp> Properties for SPIOut node

Public Members

std::string **streamName**

Output stream name.

int **busId**

SPI bus to use.

struct StereoDepthProperties

#include <StereoDepthProperties.hpp> Specify StereoDepth options

Public Types

enum MedianFilter

Median filter config for disparity post-processing

Values:

enumerator **MEDIAN_OFF**

enumerator **KERNEL_3x3**

enumerator **KERNEL_5x5**

enumerator **KERNEL_7x7**

Public Members

std::vector<std::uint8_t> **calibration**

Calibration data byte array

MedianFilter **median** = *MedianFilter::KERNEL_5x5*

Set kernel size for disparity/depth median filtering, or disable

std::int32_t **confidenceThreshold** = 200

Confidence threshold for disparity calculation, 0..255

bool **enableLeftRightCheck** = false

Computes and combines disparities in both L-R and R-L directions, and combine them. For better occlusion handling

bool **enableSubpixel** = false

Computes disparity with sub-pixel interpolation (5 fractional bits), suitable for long range

bool **enableExtendedDisparity** = false

Disparity range increased from 96 to 192, combined from full resolution and downsampled images. Suitable for short range objects

bool **rectifyMirrorFrame** = true

Mirror rectified frames: true to have disparity/depth normal (non-mirrored)

std::int32_t **rectifyEdgeFillColor** = -1

Fill color for missing data at frame edges: grayscale 0..255, or -1 to replicate pixels

bool **enableOutputRectified** = false

Enable outputting rectified frames. Optimizes computation on device side when disabled

bool **enableOutputDepth** = false

Enable outputting 'depth' stream (converted from disparity). In certain configurations, this will disable 'disparity' stream

tl::optional<std::int32_t> **width**

Input frame width. Optional (taken from MonoCamera nodes if they exist)

tl::optional<std::int32_t> **height**

Input frame height. Optional (taken from MonoCamera nodes if they exist)

class SystemInformation: public *dai::Buffer*

#include <SystemInformation.hpp> *SystemInformation* message. Carries memory usage, cpu usage and chip temperatures.

Public Functions

SystemInformation()

Construct *SystemInformation* message.

struct SystemLoggerProperties

#include <SystemLoggerProperties.hpp> SystemLoggerProperties

Public Members

float **rateHz** = 1.0f

Rate at which the messages are going to be sent in hertz

struct TensorInfo

struct Timestamp

struct Tracklet

#include <RawTracklets.hpp> Tracklet structure

Contains tracklets from object tracker output.

Public Members

Rect **roi**

Tracked region of interest.

std::int32_t **id**

Tracklet's ID.

std::int32_t **label**

Tracklet's label ID.

std::int32_t **age**

Number of frames it is being tracked for.

TrackingStatus **status**

Status of tracklet.

ImgDetection **srcImgDetection**

Image detection that is tracked.

Point3f **spatialCoordinates**

Spatial coordinates of tracklet.

class Tracklets : public dai::Buffer

#include <Tracklets.hpp> Tracklets message. Carries object tracking information.

Public Functions

Tracklets()
Construct *Tracklets* message.

Public Members

`std::vector<Tracklet> &tracklets`
Retrieve data for *Tracklets*.
Return Vector of object tracker data, carrying tracking information.

struct VideoEncoderProperties
#include <VideoEncoderProperties.hpp> Specify VideoEncoder options such as profile, bitrate, ...

Public Types

enum RateControlMode
Rate control mode specifies if constant or variable bitrate should be used (H264 / H265)

Values:

enumerator CBR

enumerator VBR

enum Profile
Encoding profile, H264, H265 or MJPEG

Values:

enumerator H264_BASELINE

enumerator H264_HIGH

enumerator H264_MAIN

enumerator H265_MAIN

enumerator MJPEG

Public Members

`std::int32_t bitrate = 8000`
Specifies preferred bitrate (kb) of compressed output bitstream

`std::int32_t keyframeFrequency = 30`
Every x number of frames a keyframe will be inserted

`std::int32_t maxBitrate = 8000`
Specifies maximum bitrate (kb) of compressed output bitstream

`std::int32_t numBframes = 0`
Specifies number of B frames to be inserted

`std::uint32_t numFramesPool = 4`
This options specifies how many frames are available in this nodes pool (can help if receiver node is slow at consuming)

Profile **profile = Profile::H264_BASELINE**
Encoding profile, H264, H265 or MJPEG

std::int32_t **quality** = 80
Value between 0-100% (approximates quality)

bool **lossless** = false
Lossless mode ([M]JPEG only)

RateControlMode **rateCtrlMode** = *RateControlMode::CBR*
Rate control mode specifies if constant or variable bitrate should be used (H264 / H265)

std::int32_t **width** = 1920
Input and compressed output frame width

std::int32_t **height** = 1080
Input and compressed output frame height

float **frameRate** = 30.0f
Frame rate

class XLinkConnection
#include <XLinkConnection.hpp> Represents connection between host and device over XLink protocol

Public Functions

void **close** ()
Explicitly closes xlink connection.
Note This function does not need to be explicitly called as destructor closes the connection automatically

bool **isClosed** () **const**
Is the connection already closed (or disconnected)

struct XLinkInProperties
#include <XLinkInProperties.hpp> Properties for XLinkIn which define stream name

Public Members

std::string **streamName**
Name of stream

std::uint32_t **maxDataSize** = *dai::XLINK_USB_BUFFER_MAX_SIZE*
Maximum input data size

std::uint32_t **numFrames** = 8
Number of frames in pool

struct XLinkOutProperties
#include <XLinkOutProperties.hpp> Properties for XLinkOut which define stream name

Public Members

float **maxFpsLimit** = -1
Set a limit to how many packets will be sent further to host

std::string **streamName**
Name of stream

bool **metadataOnly** = false
Whether to transfer data or only object attributes

class XLinkStream

namespace bootloader

Variables

constexpr const char ***XLINK_CHANNEL_BOOTLOADER** = "__bootloader"

constexpr const char ***XLINK_CHANNEL_WATCHDOG** = "__watchdog"

constexpr std::uint32_t **XLINK_STREAM_MAX_SIZE** = 5 * 1024 * 1024

constexpr const std::chrono::milliseconds **XLINK_WATCHDOG_TIMEOUT** = {1500}

namespace request

Enums

enum Command
Values:

- enumerator** USB_ROM_BOOT
- enumerator** BOOT_APPLICATION
- enumerator** UPDATE_FLASH
- enumerator** GET_BOOTLOADER_VERSION

struct BootApplication

struct GetBootloaderVersion

struct UpdateFlash

struct UsbRomBoot

namespace response

Enums

enum Command
Values:

- enumerator** FLASH_COMPLETE
- enumerator** FLASH_STATUS_UPDATE
- enumerator** BOOTLOADER_VERSION

struct BootloaderVersion

```
    struct FlashComplete
    struct FlashStatusUpdate
namespace build
```

Variables

```
constexpr const char *VERSION = "2.2.1"
constexpr const int VERSION_MAJOR = 2
constexpr const int VERSION_MINOR = 2
constexpr const int VERSION_PATCH = 1
namespace node
```

```
class ColorCamera : public dai::Node
    #include <ColorCamera.hpp> ColorCamera node. For use with color sensors.
```

Public Functions

ColorCamera (**const** std::shared_ptr<*PipelineImpl*> &par, int64_t nodeId)
Constructs *ColorCamera* node.

void **setBoardSocket** (*CameraBoardSocket* boardSocket)
Specify which board socket to use

Parameters

- boardSocket: Board socket to use

CameraBoardSocket **getBoardSocket** () **const**
Retrieves which board socket to use
Return Board socket to use

void **setCamId** (int64_t id)
Set which color camera to use.

int64_t **getCamId** () **const**
Get which color camera to use.

void **setImageOrientation** (*CameraImageOrientation* imageOrientation)
Set camera image orientation.

CameraImageOrientation **getImageOrientation** () **const**
Get camera image orientation.

void **setColorOrder** (*ColorCameraProperties::ColorOrder* colorOrder)
Set color order of preview output images. RGB or BGR.

ColorCameraProperties::ColorOrder **getColorOrder** () **const**
Get color order of preview output frames. RGB or BGR.

void **setInterleaved** (bool interleaved)
Set planar or interleaved data of preview output frames.

bool **getInterleaved** () **const**
Get planar or interleaved data of preview output frames.

void **setFp16** (bool fp16)
Set fp16 (0..255) data type of preview output frames.

```

bool getFp16 () const
    Get fp16 (0..255) data of preview output frames.

void setPreviewSize (int width, int height)
    Set preview output size.

void setVideoSize (int width, int height)
    Set video output size.

void setStillSize (int width, int height)
    Set still output size.

void setResolution (Properties::SensorResolution resolution)
    Set sensor resolution.

Properties::SensorResolution getResolution () const
    Get sensor resolution.

void setFps (float fps)
    Set rate at which camera should produce frames
Parameters
    • fps: Rate in frames per second

float getFps () const
    Get rate at which camera should produce frames
Return Rate in frames per second

std::tuple<int, int> getPreviewSize () const
    Get preview size as tuple.

int getPreviewWidth () const
    Get preview width.

int getPreviewHeight () const
    Get preview height.

std::tuple<int, int> getVideoSize () const
    Get video size as tuple.

int getVideoWidth () const
    Get video width.

int getVideoHeight () const
    Get video height.

std::tuple<int, int> getStillSize () const
    Get still size as tuple.

int getStillWidth () const
    Get still width.

int getStillHeight () const
    Get still height.

std::tuple<int, int> getResolutionSize () const
    Get sensor resolution as size.

int getResolutionWidth () const
    Get sensor resolution width.

int getResolutionHeight () const
    Get sensor resolution height.

```

void **sensorCenterCrop** ()
Specify sensor center crop. Resolution size / video size

void **setSensorCrop** (float x, float y)
Specifies sensor crop rectangle
Parameters

- x: Top left X coordinate
- y: Top left Y coordinate

std::tuple<float, float> **getSensorCrop** () **const**
Return Sensor top left crop coordinates

float **getSensorCropX** () **const**
Get sensor top left x crop coordinate.

float **getSensorCropY** () **const**
Get sensor top left y crop coordinate.

void **setWaitForConfigInput** (bool wait)
Specify to wait until inputConfig receives a configuration message, before sending out a frame.
Parameters

- wait: True to wait for inputConfig message, false otherwise

bool **getWaitForConfigInput** ()
See [setWaitForConfigInput](#)
Return True if wait for inputConfig message, false otherwise

void **setPreviewKeepAspectRatio** (bool keep)
Specifies whether preview output should preserve aspect ratio, after downscaling from video size or not.
Parameters

- keep: If true, a larger crop region will be considered to still be able to create the final image in the specified aspect ratio. Otherwise video size is resized to fit preview size

bool **getPreviewKeepAspectRatio** ()
See [setPreviewKeepAspectRatio](#)
Return Preview keep aspect ratio option

Public Members

CameraControl **initialControl**

Initial control options to apply to sensor

Input **inputConfig** = { *this, "inputConfig", Input::Type::SReceiver, false, 8, { { *DatatypeEnum::ImageManipConfig*, false } } }
Input for *ImageManipConfig* message, which can modify crop paremeters in runtime

Default queue is non-blocking with size 8

Input **inputControl** = { *this, "inputControl", Input::Type::SReceiver, true, 8, { { *DatatypeEnum::CameraControl*, false } } }
Input for *CameraControl* message, which can modify camera parameters in runtime

Default queue is blocking with size 8

Output **video** = { *this, "video", Output::Type::MSender, { { *DatatypeEnum::ImgFrame*, false } } }
Outputs *ImgFrame* message that carries NV12 encoded (YUV420, UV plane interleaved) frame data.

Suitable for use with *VideoEncoder* node

Output **preview** = { *this, "preview", Output::Type::MSender, {{ *DatatypeEnum::ImgFrame*, false }} }
Outputs *ImgFrame* message that carries BGR/RGB planar/interleaved encoded frame data.

Suitable for use with *NeuralNetwork* node

Output **still** = { *this, "still", Output::Type::MSender, {{ *DatatypeEnum::ImgFrame*, false }} }
Outputs *ImgFrame* message that carries NV12 encoded (YUV420, UV plane interleaved) frame data.

The message is sent only when a *CameraControl* message arrives to inputControl with captureStill command set.

class DetectionNetwork : public *dai::node::NeuralNetwork*

#include <DetectionNetwork.hpp> *DetectionNetwork*. Base for different network specializations.

Subclassed by *dai::node::MobileNetDetectionNetwork*, *dai::node::SpatialDetectionNetwork*, *dai::node::YoloDetectionNetwork*

Public Functions

void **setConfidenceThreshold** (float *thresh*)

Specifies confidence threshold at which to filter the rest of the detections.

Parameters

- *thresh*: Detection confidence must be greater than specified threshold to be added to the list

Public Members

Input **input** = { *this, "in", Input::Type::SReceiver, true, 5, {{ *DatatypeEnum::Buffer*, true }} }

Input message with data to be inferred upon Default queue is blocking with size 5

Output **out** = { *this, "out", Output::Type::MSender, {{ *DatatypeEnum::ImgDetections*, false }} }

Outputs *ImgDetections* message that carries parsed detection results.

Output **passthrough** = { *this, "passthrough", Output::Type::MSender, {{ *DatatypeEnum::Buffer*, true }} }

Passthrough message on which the inference was performed.

Suitable for when input queue is set to non-blocking behavior.

class ImageManip : public *dai::Node*

#include <ImageManip.hpp> *ImageManip* node. Capability to crop, resize, warp, ... incoming image frames.

Public Functions

void **setWaitForConfigInput** (bool *wait*)

Specify whether or not wait until configuration message arrives to inputConfig Input.

Parameters

- *wait*: True to wait for configuration message, false otherwise

void **setNumFramesPool** (int *numFramesPool*)

Specify number of frames in pool.

Parameters

- *numFramesPool*: How many frames should the pool have

void **setMaxOutputFrameSize** (int *maxFrameSize*)

Specify maximum size of output image.

Parameters

- `maxFrameSize`: Maximum frame size in bytes

Public Members

ImageManipConfig **initialConfig**

Initial config to use when manipulating frames

Input **inputConfig** = { *this, "inputConfig", Input::Type::SReceiver, true, 8, {{*DatatypeEnum::ImageManipConfig*, Input *ImageManipConfig* message with ability to modify parameters in runtime Default queue is blocking with size 8

Input **inputImage** = { *this, "inputImage", Input::Type::SReceiver, true, 8, {{*DatatypeEnum::ImgFrame*, true}}} Input image to be modified Default queue is blocking with size 8

Output **out** = { *this, "out", Output::Type::MSender, {{*DatatypeEnum::ImgFrame*, true}}} Outputs *ImgFrame* message that carries modified image.

class MobileNetDetectionNetwork : public *dai::node::DetectionNetwork*
#include <DetectionNetwork.hpp> *MobileNetDetectionNetwork* node. Parses MobileNet results.

class MobileNetSpatialDetectionNetwork : public *dai::node::SpatialDetectionNetwork*
#include <SpatialDetectionNetwork.hpp> *MobileNetSpatialDetectionNetwork*. Mobilenet-SSD based network with spatial location data.

class MonoCamera : public *dai::Node*
#include <MonoCamera.hpp> *MonoCamera* node. For use with grayscale sensors.

Public Functions

void **setBoardSocket** (*CameraBoardSocket* boardSocket)
Specify which board socket to use

Parameters

- `boardSocket`: Board socket to use

CameraBoardSocket **getBoardSocket** () const

Retrieves which board socket to use

Return Board socket to use

void **setImageOrientation** (*CameraImageOrientation* imageOrientation)
Set camera image orientation.

CameraImageOrientation **getImageOrientation** () const
Get camera image orientation.

void **setResolution** (*Properties::SensorResolution* resolution)
Set sensor resolution.

Properties::SensorResolution **getResolution** () const
Get sensor resolution.

void **setFps** (float fps)
Set rate at which camera should produce frames

Parameters

- `fps`: Rate in frames per second

float **getFps** () const
Get rate at which camera should produce frames
Return Rate in frames per second

std::tuple<int, int> **getResolutionSize** () const

Get sensor resolution as size.

int **getResolutionWidth** () const

Get sensor resolution width.

int **getResolutionHeight** () const

Get sensor resolution height.

Public Members

CameraControl **initialControl**

Initial control options to apply to sensor

Input **inputControl** = { *this, "inputControl", Input::Type::SReceiver, true, 8, {{ *DatatypeEnum::CameraControl*, false }}

Input for *CameraControl* message, which can modify camera parameters in runtime Default queue is blocking with size 8

Output **out** = { *this, "out", Output::Type::MSender, {{ *DatatypeEnum::ImgFrame*, false }}

Outputs *ImgFrame* message that carries RAW8 encoded (grayscale) frame data.

Suitable for use *StereoDepth* node

class MyProducer : public *dai::Node*

class NeuralNetwork : public *dai::Node*

#include <NeuralNetwork.hpp> *NeuralNetwork* node. Runs a neural inference on input data.

Subclassed by *dai::node::DetectionNetwork*

Public Functions

void **setBlobPath** (const std::string &path)

Load network blob into assets and use once pipeline is started.

Throws if file doesn't exist or isn't a valid network blob.

Parameters

- path: Path to network blob

void **setNumPoolFrames** (int numFrames)

Specifies how many frames will be available in the pool

Parameters

- numFrames: How many frames will pool have

void **setNumInferenceThreads** (int numThreads)

How many threads should the node use to run the network.

Parameters

- numThreads: Number of threads to dedicate to this node

void **setNumNCEPerInferenceThread** (int numNCEPerThread)

How many Neural Compute Engines should a single thread use for inference

Parameters

- numNCEPerThread: Number of NCE per thread

int **getNumInferenceThreads** ()

How many inference threads will be used to run the network

Return Number of threads, 0, 1 or 2. Zero means AUTO

Public Members

Input **input** = { *this, "in", Input::Type::SReceiver, true, 5, { { *DatatypeEnum::Buffer*, true } } }
Input message with data to be inferred upon Default queue is blocking with size 5

Output **out** = { *this, "out", Output::Type::MSender, { { *DatatypeEnum::NNData*, false } } }
Outputs *NNData* message that carries inference results

Output **passthrough** = { *this, "passthrough", Output::Type::MSender, { { *DatatypeEnum::Buffer*, true } } }
Passthrough message on which the inference was performed.

Suitable for when input queue is set to non-blocking behavior.

class ObjectTracker : public *dai::Node*
#include <ObjectTracker.hpp> *ObjectTracker* node. Performs object tracking using Kalman filter and hungarian algorithm.

Public Functions

void **setTrackerThreshold** (float *threshold*)

Specify tracker threshold.

Parameters

- *threshold*: Above this threshold the detected objects will be tracked. Default 0, all image detections are tracked.

void **setMaxObjectsToTrack** (std::int32_t *maxObjectsToTrack*)

Specify maximum number of object to track.

Parameters

- *maxObjectsToTrack*: Maximum number of object to track. Maximum 60.

void **setDetectionLabelsToTrack** (std::vector<std::uint32_t> *labels*)

Specify detection labels to track.

Parameters

- *labels*: Detection labels to track. Default every label is tracked from image detection network output.

void **setTrackerType** (*TrackerType* *type*)

Specify tracker type algorithm.

Parameters

- *type*: Tracker type.

void **setTrackerIdAssignmentPolicy** (*TrackerIdAssignmentPolicy* *type*)

Specify tracker ID assignment policy.

Parameters

- *type*: Tracker ID assignment policy.

Public Members

Input **inputTrackerFrame** = { *this, "inputTrackerFrame", Input::Type::SReceiver, false, 4, { { *DatatypeEnum::ImgFrame*, true } } }
Input *ImgFrame* message on which tracking will be performed. RGBp, BGRp, NV12, YUV420p types are supported. Default queue is non-blocking with size 4.

Input **inputDetectionFrame** = { *this, "inputDetectionFrame", Input::Type::SReceiver, false, 4, { { *DatatypeEnum::ImgFrame*, true } } }
Input *ImgFrame* message on which object detection was performed. Default queue is non-blocking with size 4.

Input **inputDetections** = { *this, "inputDetections", Input::Type::SReceiver, false, 4, { { *DatatypeEnum::ImgDetections*, false } } }
 Input message with image detection from neural network. Default queue is non-blocking with size 4.

Output **out** = { *this, "out", Output::Type::MSender, { { *DatatypeEnum::Tracklets*, false } } }
 Outputs *Tracklets* message that carries object tracking results.

Output **passthroughTrackerFrame** = { *this, "passthroughTrackerFrame", Output::Type::MSender, { { *DatatypeEnum::TrackerFrame*, false } } }
 Passthrough *ImgFrame* message on which tracking was performed. Suitable for when input queue is set to non-blocking behavior.

Output **passthroughDetectionFrame** = { *this, "passthroughDetectionFrame", Output::Type::MSender, { { *DatatypeEnum::DetectionFrame*, false } } }
 Passthrough *ImgFrame* message on which object detection was performed. Suitable for when input queue is set to non-blocking behavior.

Output **passthroughDetections** = { *this, "passthroughDetections", Output::Type::MSender, { { *DatatypeEnum::Detections*, false } } }
 Passthrough image detections message from neural network output. Suitable for when input queue is set to non-blocking behavior.

class SpatialDetectionNetwork : public *dai::node::DetectionNetwork*
#include <SpatialDetectionNetwork.hpp> SpatialDetectionNetwork node. Runs a neural inference on input image and calculates spatial location data.
 Subclassed by *dai::node::MobileNetSpatialDetectionNetwork*, *dai::node::YoloSpatialDetectionNetwork*

Public Functions

void **setBoundingBoxScaleFactor** (float *scaleFactor*)
 Specifies scale factor for detected bounding boxes.

Parameters

- *scaleFactor*: Scale factor must be in the interval (0,1].

void **setDepthLowerThreshold** (uint32_t *lowerThreshold*)
 Specifies lower threshold in millimeters for depth values which will be used to calculate spatial data

Parameters

- *lowerThreshold*: LowerThreshold must be in the interval [0,upperThreshold] and less than upperThreshold.

void **setDepthUpperThreshold** (uint32_t *upperThreshold*)
 Specifies upper threshold in millimeters for depth values which will be used to calculate spatial data

Parameters

- *upperThreshold*: UpperThreshold must be in the interval (lowerThreshold,65535].

Public Members

Input **input** = { *this, "in", Input::Type::SReceiver, true, 5, { { *DatatypeEnum::ImgFrame*, false } } }
 Input message with data to be inferred upon. Default queue is blocking with size 5

Input **inputDepth** = { *this, "inputDepth", Input::Type::SReceiver, false, 4, { { *DatatypeEnum::ImgFrame*, false } } }
 Input message with depth data used to retrieve spatial information about detected object. Default queue is non-blocking with size 4

Output **out** = { *this, "out", Output::Type::MSender, { { *DatatypeEnum::SpatialImgDetections*, false } } }
 Outputs *ImgDetections* message that carries parsed detection results.

Output **boundingBoxMapping** = { *this, "boundingBoxMapping", Output::Type::MSender, { { *DatatypeEnum::SpatialBoundingBoxMapping*, false } } }
 Outputs mapping of detected bounding boxes relative to depth map
 Suitable for when displaying remapped bounding boxes on depth frame

Output **passthrough** = { *this, "passthrough", Output::Type::MSender, { { *DatatypeEnum::ImgFrame*, false } } }
Passthrough message on which the inference was performed.

Suitable for when input queue is set to non-blocking behavior.

Output **passthroughDepth** = { *this, "passthroughDepth", Output::Type::MSender, { { *DatatypeEnum::ImgFrame*, false } } }
Passthrough message for depth frame on which the spatial location calculation was performed.

Suitable for when input queue is set to non-blocking behavior.

class SpatialLocationCalculator : public dai::Node
#include <SpatialLocationCalculator.hpp> *SpatialLocationCalculator* node. Calculates spatial location data on a set of ROIs on depth map.

Public Functions

void **setWaitForConfigInput** (bool *wait*)
Specify whether or not wait until configuration message arrives to inputConfig Input.

Parameters

- *wait*: True to wait for configuration message, false otherwise.

Public Members

SpatialLocationCalculatorConfig **initialConfig**
Initial config to use when calculating spatial location data.

Input **inputConfig** = { *this, "inputConfig", Input::Type::SReceiver, false, 4, { { *DatatypeEnum::SpatialLocationCalculatorConfig*, false } } }
Input *SpatialLocationCalculatorConfig* message with ability to modify parameters in runtime.
Default queue is non-blocking with size 4.

Input **inputDepth** = { *this, "inputDepth", Input::Type::SReceiver, false, 4, { { *DatatypeEnum::ImgFrame*, false } } }
Input message with depth data used to retrieve spatial information about detected object. Default queue is non-blocking with size 4.

Output **out** = { *this, "out", Output::Type::MSender, { { *DatatypeEnum::SpatialLocationCalculatorData*, false } } }
Outputs *SpatialLocationCalculatorData* message that carries spatial location results.

Output **passthroughDepth** = { *this, "passthroughDepth", Output::Type::MSender, { { *DatatypeEnum::ImgFrame*, false } } }
Passthrough message on which the calculation was performed. Suitable for when input queue is set to non-blocking behavior.

class SPIOut : public dai::Node
#include <SPIOut.hpp> *SPIOut* node. Sends messages over SPI.

Public Functions

void **setStreamName** (std::string *name*)
Specifies stream name over which the node will send data

Parameters

- *name*: Stream name

void **setBusId** (int *id*)
Specifies SPI Bus number to use

Parameters

- *id*: SPI Bus id

Public Members

Input **input** = { *this, "in", Input::Type::SReceiver, true, 8, { { *DatatypeEnum::Buffer*, true } } }

Input for any type of messages to be transferred over SPI stream

Default queue is blocking with size 8

class StereoDepth : public *dai::Node*
#include <StereoDepth.hpp> *StereoDepth* node. Compute stereo disparity and depth from left-right image pair.

Public Functions

void **loadCalibrationFile** (const std::string &path)
 Specify local filesystem path to the calibration file

Parameters

- path: Path to calibration file. If empty use EEPROM

void **loadCalibrationData** (const std::vector<std::uint8_t> &data)
 Specify calibration data as a vector of bytes

Parameters

- path: Calibration data. If empty use EEPROM

void **setEmptyCalibration** ()
 Specify that a passthrough/dummy calibration should be used, when input frames are already rectified (e.g. sourced from recordings on the host)

void **setInputResolution** (int width, int height)
 Specify input resolution size

Optional if *MonoCamera* exists, otherwise necessary

void **setMedianFilter** (*Properties::MedianFilter* median)

Parameters

- median: Set kernel size for disparity/depth median filtering, or disable

void **setConfidenceThreshold** (int confThr)
 Confidence threshold for disparity calculation

Parameters

- confThr: Confidence threshold value 0..255

void **setLeftRightCheck** (bool enable)
 Computes and combines disparities in both L-R and R-L directions, and combine them.

For better occlusion handling

void **setSubpixel** (bool enable)
 Computes disparity with sub-pixel interpolation (5 fractional bits).

Suitable for long range

void **setExtendedDisparity** (bool enable)
 Disparity range increased from 96 to 192, combined from full resolution and downscaled images.

Suitable for short range objects

void **setRectifyEdgeFillColor** (int color)
 Fill color for missing data at frame edges

Parameters

- color: Grayscale 0..255, or -1 to replicate pixels

void **setRectifyMirrorFrame** (bool *enable*)

Mirror rectified frames

Parameters

- *enable*: True for normal disparity/depth, otherwise mirrored

void **setOutputRectified** (bool *enable*)

Enable outputting rectified frames. Optimizes computation on device side when disabled

void **setOutputDepth** (bool *enable*)

Enable outputting 'depth' stream (converted from disparity). In certain configurations, this will disable 'disparity' stream

Public Members

Input **left** = { *this, "left", Input::Type::SReceiver, false, 8, { { *DatatypeEnum::ImgFrame*, true } } }

Input for left *ImgFrame* of left-right pair

Default queue is non-blocking with size 8

Input **right** = { *this, "right", Input::Type::SReceiver, false, 8, { { *DatatypeEnum::ImgFrame*, true } } }

Input for right *ImgFrame* of left-right pair

Default queue is non-blocking with size 8

Output **depth** = { *this, "depth", Output::Type::MSender, { { *DatatypeEnum::ImgFrame*, false } } }

Outputs *ImgFrame* message that carries RAW16 encoded (0..65535) depth data in millimeters.

Output **disparity** = { *this, "disparity", Output::Type::MSender, { { *DatatypeEnum::ImgFrame*, false } } }

Outputs *ImgFrame* message that carries RAW8 encoded (0..96 or 0..192 for Extended mode) disparity data.

Output **syncedLeft** = { *this, "syncedLeft", Output::Type::MSender, { { *DatatypeEnum::ImgFrame*, false } } }

Passthrough *ImgFrame* message from 'left' Input.

Output **syncedRight** = { *this, "syncedRight", Output::Type::MSender, { { *DatatypeEnum::ImgFrame*, false } } }

Passthrough *ImgFrame* message from 'right' Input.

Output **rectifiedLeft** = { *this, "rectifiedLeft", Output::Type::MSender, { { *DatatypeEnum::ImgFrame*, false } } }

Outputs *ImgFrame* message that carries RAW8 encoded (grayscale) rectified frame data.

Output **rectifiedRight** = { *this, "rectifiedRight", Output::Type::MSender, { { *DatatypeEnum::ImgFrame*, false } } }

Outputs *ImgFrame* message that carries RAW8 encoded (grayscale) rectified frame data.

class SystemLogger: public *dai::Node*

#include <SystemLogger.hpp> *SystemLogger* node. Send system information periodically.

Public Functions

void **setRate** (float *hz*)

Specify logging rate, at which messages will be sent to out output

Parameters

- *hz*: Sending rate in hertz (messages per second)

Public Members

Output **out** = {*this, "out", Output::Type::MSender, {{*DatatypeEnum::SystemInformation*, false}}} Outputs *SystemInformation* message that carries various system information like memory and CPU usage, temperatures, ...

class VideoEncoder : public *dai::Node*
#include <VideoEncoder.hpp> *VideoEncoder* node. Encodes frames into MJPEG, H264 or H265.

Public Functions

void **setDefaultProfilePreset** (int *width*, int *height*, float *fps*, *Properties::Profile profile*)

Sets a default preset based on specified input size, frame rate and profile

Parameters

- *width*: Input frame width
- *height*: Input frame height
- *fps*: Frame rate in frames per second
- *profile*: Encoding profile

void **setDefaultProfilePreset** (std::tuple<int, int> *size*, float *fps*, *Properties::Profile profile*)

Sets a default preset based on specified input size, frame rate and profile

Parameters

- *size*: Input frame size
- *fps*: Frame rate in frames per second
- *profile*: Encoding profile

void **setNumFramesPool** (int *frames*)

Set number of frames in pool

Parameters

- *frames*: Number of pool frames

int **getNumFramesPool** () **const**

Get number of frames in pool

Return Number of pool frames

void **setRateControlMode** (*Properties::RateControlMode mode*)

Set rate control mode.

void **setProfile** (std::tuple<int, int> *size*, *Properties::Profile profile*)

Set encoding profile.

void **setProfile** (int *width*, int *height*, *Properties::Profile profile*)

Set encoding profile.

void **setBitrate** (int *bitrate*)

Set output bitrate in bps. Final bitrate depends on rate control mode.

void **setBitrateKbps** (int *bitrateKbps*)

Set output bitrate in kbps. Final bitrate depends on rate control mode.

void **setKeyframeFrequency** (int *freq*)

Set keyframe frequency. Every Nth frame a keyframe is inserted.

Applicable only to H264 and H265 profiles

Examples:

- 30 FPS video, keyframe frequency: 30. Every 1s a keyframe will be inserted

- 60 FPS video, keyframe frequency: 180. Every 3s a keyframe will be inserted

void **setNumBFrames** (int *numBFrames*)
Set number of B frames to be inserted.

void **setQuality** (int *quality*)
Set quality

Parameters

- *quality*: Value between 0-100%. Approximates quality

void **setLossless** (bool *lossless*)
Set lossless mode. Applies only to [M]JPEG profile

Parameters

- *lossless*: True to enable lossless jpeg encoding, false otherwise

void **setFrameRate** (int *frameRate*)
Sets expected frame rate

Parameters

- *frameRate*: Frame rate in frames per second

Properties::RateControlMode **getRateControlMode** () const
Get rate control mode.

Properties::Profile **getProfile** () const
Get profile.

int **getBitrate** () const
Get bitrate in bps.

int **getBitrateKbps** () const
Get bitrate in kbps.

int **getKeyframeFrequency** () const
Get keyframe frequency.

int **getNumBFrames** () const
Get number of B frames.

int **getQuality** () const
Get quality.

std::tuple<int, int> **getSize** () const
Get input size.

int **getWidth** () const
Get input width.

int **getHeight** () const
Get input height.

int **getFrameRate** () const
Get frame rate.

bool **getLossless** () const
Get lossless mode. Applies only when using [M]JPEG profile.

Public Members

Input **input** = { *this, "in", Input::Type::SReceiver, true, 4, { { *DatatypeEnum::ImgFrame*, true } } }
 Input for NV12 *ImgFrame* to be encoded Default queue is blocking with size set by 'setNumFramesPool' (4).

Output **bitstream** = { *this, "bitstream", Output::Type::MSender, { { *DatatypeEnum::ImgFrame*, false } } }
 Outputs *ImgFrame* message that carries BITSTREAM encoded (MJPEG, H264 or H265) frame data.

class XLinkIn : public dai::Node

#include <XLinkIn.hpp> *XLinkIn* node. Receives messages over XLink.

Public Functions

void **setStreamName** (const std::string &name)

Specifies XLink stream name to use.

The name should not start with double underscores '__', as those are reserved for internal use.

Parameters

- name: Stream name

void **setMaxDataSize** (std::uint32_t maxSize)

Set maximum message size it can receive

Parameters

- maxSize: Maximum size in bytes

void **setNumFrames** (std::uint32_t numFrames)

Set number of frames in pool for sending messages forward

Parameters

- numFrames: Maximum number of frames in pool

std::string **getStreamName** () const

Get stream name.

std::uint32_t **getMaxDataSize** () const

Get maximum messages size in bytes.

std::uint32_t **getNumFrames** () const

Get number of frames in pool.

Public Members

Output **out** = { *this, "out", Output::Type::MSender, { { *DatatypeEnum::Buffer*, true } } }

Outputs message of same type as send from host.

class XLinkOut : public dai::Node

#include <XLinkOut.hpp> *XLinkOut* node. Sends messages over XLink.

Public Functions

void **setStreamName** (const std::string &name)

Specifies XLink stream name to use.

The name should not start with double underscores '___', as those are reserved for internal use.

Parameters

- name: Stream name

void **setFpsLimit** (float fps)

Specifies a message sending limit. It's approximated from specified rate.

Parameters

- fps: Approximate rate limit in messages per second

void **setMetadataOnly** (bool metadataOnly)

Specify whether to transfer only messages attributes and not buffer data

std::string **getStreamName** () const

Get stream name.

float **getFpsLimit** () const

Get rate limit in messages per second.

bool **getMetadataOnly** () const

Get whether to transfer only messages attributes and not buffer data.

Public Members

Input **input** = { *this, "in", Input::Type::SReceiver, true, 8, {{ *DatatypeEnum::Buffer*, true }} }

Input for any type of messages to be transfered over XLink stream

Default queue is blocking with size 8

```
class YoloDetectionNetwork : public dai::node::DetectionNetwork
#include <DetectionNetwork.hpp> YoloDetectionNetwork node. Parses Yolo results.
```

Public Functions

void **setNumClasses** (const int numClasses)

Set num classes.

void **setCoordinateSize** (const int coordinates)

Set coordinate size.

void **setAnchors** (std::vector<float> anchors)

Set anchors.

void **setAnchorMasks** (std::map<std::string, std::vector<int>> anchorMasks)

Set anchor masks.

void **setIouThreshold** (float thresh)

Set Iou threshold.

```
class YoloSpatialDetectionNetwork : public dai::node::SpatialDetectionNetwork
#include <SpatialDetectionNetwork.hpp> YoloSpatialDetectionNetwork. (tiny)Yolov3/v4 based net-
work with spatial location data.
```

Public Functions

void **setNumClasses** (**const** int *numClasses*)

Set num classes.

void **setCoordinateSize** (**const** int *coordinates*)

Set coordinate size.

void **setAnchors** (std::vector<float> *anchors*)

Set anchors.

void **setAnchorMasks** (std::map<std::string, std::vector<int>> *anchorMasks*)

Set anchor masks.

void **setIouThreshold** (float *thresh*)

Set Iou threshold.

We're always happy to help with code or other questions you might have.

PYTHON MODULE INDEX

d

depthai, [247](#)

Symbols

- `__init__()` (*depthai.ADatatype* method), 249
- `__init__()` (*depthai.Asset* method), 249
- `__init__()` (*depthai.AssetManager* method), 249
- `__init__()` (*depthai.Buffer* method), 251
- `__init__()` (*depthai.CameraBoardSocket* method), 252
- `__init__()` (*depthai.CameraControl* method), 258
- `__init__()` (*depthai.CameraControl.AntiBandingMode* method), 253
- `__init__()` (*depthai.CameraControl.AutoFocusMode* method), 254
- `__init__()` (*depthai.CameraControl.AutoWhiteBalanceMode* method), 255
- `__init__()` (*depthai.CameraControl.EffectMode* method), 256
- `__init__()` (*depthai.CameraControl.SceneMode* method), 258
- `__init__()` (*depthai.CameraImageOrientation* method), 261
- `__init__()` (*depthai.ChipTemperature* method), 261
- `__init__()` (*depthai.ColorCamera* method), 263
- `__init__()` (*depthai.ColorCameraProperties* method), 268
- `__init__()` (*depthai.ColorCameraProperties.ColorOrder* method), 267
- `__init__()` (*depthai.ColorCameraProperties.SensorResolution* method), 268
- `__init__()` (*depthai.CpuUsage* method), 269
- `__init__()` (*depthai.DataInputQueue* method), 269
- `__init__()` (*depthai.DataOutputQueue* method), 270
- `__init__()` (*depthai.DetectionNetwork* method), 272
- `__init__()` (*depthai.DetectionNetworkProperties* method), 273
- `__init__()` (*depthai.Device* method), 274
- `__init__()` (*depthai.DeviceBootloader* method), 281
- `__init__()` (*depthai.DeviceBootloader.Version* method), 281
- `__init__()` (*depthai.DeviceDesc* method), 283
- `__init__()` (*depthai.DeviceInfo* method), 283
- `__init__()` (*depthai.GlobalProperties* method), 283
- `__init__()` (*depthai.ImageManip* method), 284
- `__init__()` (*depthai.ImageManipConfig* method), 286
- `__init__()` (*depthai.ImgDetection* method), 288
- `__init__()` (*depthai.ImgDetections* method), 289
- `__init__()` (*depthai.ImgFrame* method), 293
- `__init__()` (*depthai.ImgFrame.Specs* method), 290
- `__init__()` (*depthai.ImgFrame.Type* method), 293
- `__init__()` (*depthai.LogLevel* method), 295
- `__init__()` (*depthai.MemoryInfo* method), 295
- `__init__()` (*depthai.MobileNetDetectionNetwork* method), 296
- `__init__()` (*depthai.MobileNetSpatialDetectionNetwork* method), 296
- `__init__()` (*depthai.MonoCamera* method), 297
- `__init__()` (*depthai.MonoCameraProperties* method), 299
- `__init__()` (*depthai.MonoCameraProperties.SensorResolution* method), 299
- `__init__()` (*depthai.NNDData* method), 300
- `__init__()` (*depthai.NeuralNetwork* method), 302
- `__init__()` (*depthai.NeuralNetworkProperties* method), 303
- `__init__()` (*depthai.Node* method), 306
- `__init__()` (*depthai.Node.Connection* method), 304
- `__init__()` (*depthai.Node.Id* method), 304
- `__init__()` (*depthai.Node.Input* method), 305
- `__init__()` (*depthai.Node.Output* method), 305
- `__init__()` (*depthai.ObjectTracker* method), 307
- `__init__()` (*depthai.ObjectTrackerProperties* method), 308
- `__init__()` (*depthai.OpenVINO* method), 310
- `__init__()` (*depthai.OpenVINO.Version* method), 310
- `__init__()` (*depthai.Pipeline* method), 312
- `__init__()` (*depthai.Point2f* method), 313
- `__init__()` (*depthai.Point3f* method), 314
- `__init__()` (*depthai.RawBuffer* method), 314
- `__init__()` (*depthai.RawCameraControl* method), 320
- `__init__()` (*depthai.RawCameraControl.AntiBandingMode* method), 315
- `__init__()` (*depthai.RawCameraControl.AutoFocusMode* method), 316
- `__init__()` (*depthai.RawCameraControl.AutoWhiteBalanceMode* method), 316

`method`), 317
`__init__()` (`depthai.RawCameraControl.EffectMode` method), 318
`__init__()` (`depthai.RawCameraControl.SceneMode` method), 320
`__init__()` (`depthai.RawImageManipConfig` method), 323
`__init__()` (`depthai.RawImageManipConfig.CropConfig` method), 321
`__init__()` (`depthai.RawImageManipConfig.CropRect` method), 321
`__init__()` (`depthai.RawImageManipConfig.FormatConfig` method), 322
`__init__()` (`depthai.RawImageManipConfig.ResizeConfig` method), 322
`__init__()` (`depthai.RawImgDetections` method), 323
`__init__()` (`depthai.RawImgFrame` method), 327
`__init__()` (`depthai.RawImgFrame.Specs` method), 324
`__init__()` (`depthai.RawImgFrame.Type` method), 327
`__init__()` (`depthai.RawNNData` method), 327
`__init__()` (`depthai.RawSpatialImgDetections` method), 328
`__init__()` (`depthai.RawSystemInformation` method), 328
`__init__()` (`depthai.RawTracklets` method), 329
`__init__()` (`depthai.Rect` method), 329
`__init__()` (`depthai.RotatedRect` method), 330
`__init__()` (`depthai.SPIOut` method), 331
`__init__()` (`depthai.Size2f` method), 331
`__init__()` (`depthai.SpatialDetectionNetwork` method), 332
`__init__()` (`depthai.SpatialDetectionNetworkProperties` method), 333
`__init__()` (`depthai.SpatialImgDetection` method), 334
`__init__()` (`depthai.SpatialImgDetections` method), 334
`__init__()` (`depthai.SpatialLocationCalculator` method), 335
`__init__()` (`depthai.SpatialLocationCalculatorConfig` method), 335
`__init__()` (`depthai.SpatialLocationCalculatorConfigData` method), 336
`__init__()` (`depthai.SpatialLocationCalculatorConfigThresholds` method), 336
`__init__()` (`depthai.SpatialLocationCalculatorData` method), 337
`__init__()` (`depthai.SpatialLocationCalculatorProperties` method), 337
`__init__()` (`depthai.SpatialLocations` method), 337
`__init__()` (`depthai.StereoDepth` method), 339
`__init__()` (`depthai.StereoDepthProperties` method), 342
`__init__()` (`depthai.StereoDepthProperties.MedianFilter` method), 342
`__init__()` (`depthai.SystemInformation` method), 342
`__init__()` (`depthai.SystemLogger` method), 343
`__init__()` (`depthai.SystemLoggerProperties` method), 343
`__init__()` (`depthai.TensorInfo` method), 346
`__init__()` (`depthai.TensorInfo.DataType` method), 344
`__init__()` (`depthai.TensorInfo.StorageOrder` method), 346
`__init__()` (`depthai.Timestamp` method), 346
`__init__()` (`depthai.TrackerIdAssignmentPolicy` method), 347
`__init__()` (`depthai.TrackerType` method), 347
`__init__()` (`depthai.Tracklet` method), 348
`__init__()` (`depthai.Tracklet.TrackingStatus` method), 348
`__init__()` (`depthai.Tracklets` method), 349
`__init__()` (`depthai.VideoEncoder` method), 350
`__init__()` (`depthai.VideoEncoderProperties` method), 354
`__init__()` (`depthai.VideoEncoderProperties.Profile` method), 354
`__init__()` (`depthai.VideoEncoderProperties.RateControlMode` method), 354
`__init__()` (`depthai.XLinkConnection` method), 355
`__init__()` (`depthai.XLinkDeviceState` method), 356
`__init__()` (`depthai.XLinkIn` method), 356
`__init__()` (`depthai.XLinkOut` method), 357
`__init__()` (`depthai.XLinkPlatform` method), 358
`__init__()` (`depthai.XLinkProtocol` method), 359
`__init__()` (`depthai.YoloDetectionNetwork` method), 359
`__init__()` (`depthai.YoloSpatialDetectionNetwork` method), 359

A

`ACTION` (`depthai.CameraControl.SceneMode` attribute), 257
`ACTION` (`depthai.RawCameraControl.SceneMode` attribute), 319
`ArrayType` (class in `depthai`), 248
`add()` (`depthai.AssetManager` method), 250
`addCallback()` (`depthai.DataOutputQueue` method), 270
`addExisting()` (`depthai.AssetManager` method), 250
`addLogCallback()` (`depthai.Device` method), 275
`addROI()` (`depthai.SpatialLocationCalculatorConfig` method), 335
`alignment()` (`depthai.Asset` property), 249
`anchorMasks()` (`depthai.DetectionNetworkProperties` property), 273

[anchors\(\)](#) (*depthai.DetectionNetworkProperties* property), 273
[angle\(\)](#) (*depthai.RotatedRect* property), 330
[AQUA](#) (*depthai.CameraControl.EffectMode* attribute), 256
[AQUA](#) (*depthai.RawCameraControl.EffectMode* attribute), 318
[area\(\)](#) (*depthai.Rect* method), 329
[areVersionsBlobCompatible\(\)](#) (*depthai.OpenVINO* static method), 310
[Asset](#) (class in *depthai*), 249
[AssetManager](#) (class in *depthai*), 249
[AUTO](#) (*depthai.CameraBoardSocket* attribute), 252
[AUTO](#) (*depthai.CameraControl.AntiBandingMode* attribute), 253
[AUTO](#) (*depthai.CameraControl.AutoFocusMode* attribute), 254
[AUTO](#) (*depthai.CameraControl.AutoWhiteBalanceMode* attribute), 255
[AUTO](#) (*depthai.CameraImageOrientation* attribute), 261
[AUTO](#) (*depthai.RawCameraControl.AntiBandingMode* attribute), 315
[AUTO](#) (*depthai.RawCameraControl.AutoFocusMode* attribute), 316
[AUTO](#) (*depthai.RawCameraControl.AutoWhiteBalanceMode* attribute), 317
[autoFocusMode\(\)](#) (*depthai.RawCameraControl* property), 320
[average\(\)](#) (*depthai.ChipTemperature* property), 261
[average\(\)](#) (*depthai.CpuUsage* property), 269

B

[BARCODE](#) (*depthai.CameraControl.SceneMode* attribute), 257
[BARCODE](#) (*depthai.RawCameraControl.SceneMode* attribute), 319
[batchSize\(\)](#) (*depthai.RawNNData* property), 328
[BEACH](#) (*depthai.CameraControl.SceneMode* attribute), 257
[BEACH](#) (*depthai.RawCameraControl.SceneMode* attribute), 319
[bgBlue\(\)](#) (*depthai.RawImageManipConfig.ResizeConfig* property), 322
[bgGreen\(\)](#) (*depthai.RawImageManipConfig.ResizeConfig* property), 322
[BGR](#) (*depthai.ColorCameraProperties.ColorOrder* attribute), 267
[BGR888i](#) (*depthai.ImgFrame.Type* attribute), 292
[BGR888i](#) (*depthai.RawImgFrame.Type* attribute), 326
[BGR888p](#) (*depthai.ImgFrame.Type* attribute), 292
[BGR888p](#) (*depthai.RawImgFrame.Type* attribute), 326
[bgRed\(\)](#) (*depthai.RawImageManipConfig.ResizeConfig* property), 322

[BGRF16F16F16i](#) (*depthai.ImgFrame.Type* attribute), 292
[BGRF16F16F16i](#) (*depthai.RawImgFrame.Type* attribute), 326
[BGRF16F16F16p](#) (*depthai.ImgFrame.Type* attribute), 292
[BGRF16F16F16p](#) (*depthai.RawImgFrame.Type* attribute), 326
[bitrate\(\)](#) (*depthai.VideoEncoderProperties* property), 354
[BITSTREAM](#) (*depthai.ImgFrame.Type* attribute), 292
[BITSTREAM](#) (*depthai.RawImgFrame.Type* attribute), 326
[bitstream\(\)](#) (*depthai.VideoEncoder* property), 350
[BLACKBOARD](#) (*depthai.CameraControl.EffectMode* attribute), 256
[BLACKBOARD](#) (*depthai.RawCameraControl.EffectMode* attribute), 318
[blobSize\(\)](#) (*depthai.NeuralNetworkProperties* property), 303
[blobUri\(\)](#) (*depthai.NeuralNetworkProperties* property), 303
[boardSocket\(\)](#) (*depthai.ColorCameraProperties* property), 268
[boardSocket\(\)](#) (*depthai.MonoCameraProperties* property), 299
[bottomRight\(\)](#) (*depthai.Rect* method), 329
[boundingBoxMapping\(\)](#) (*depthai.SpatialDetectionNetwork* property), 332
[Buffer](#) (class in *depthai*), 251
[bytesPP\(\)](#) (*depthai.ImgFrame.Specs* property), 290
[bytesPP\(\)](#) (*depthai.RawImgFrame.Specs* property), 324

C

[C](#) (*depthai.TensorInfo.StorageOrder* attribute), 345
[calibration\(\)](#) (*depthai.StereoDepthProperties* property), 342
[CameraBoardSocket](#) (class in *depthai*), 251
[CameraControl](#) (class in *depthai*), 252
[CameraControl.AntiBandingMode](#) (class in *depthai*), 253
[CameraControl.AutoFocusMode](#) (class in *depthai*), 253
[CameraControl.AutoWhiteBalanceMode](#) (class in *depthai*), 254
[CameraControl.EffectMode](#) (class in *depthai*), 255
[CameraControl.SceneMode](#) (class in *depthai*), 256
[CameraImageOrientation](#) (class in *depthai*), 260
[canConnect\(\)](#) (*depthai.Node.Output* method), 305
[CANDLELIGHT](#) (*depthai.CameraControl.SceneMode* attribute), 257

CANDLELIGHT (*depthai.RawCameraControl.SceneMode attribute*), 319
 category () (*depthai.RawImgFrame property*), 327
 CBR (*depthai.VideoEncoderProperties.RateControlMode attribute*), 354
 center () (*depthai.RotatedRect property*), 330
 ChipTemperature (*class in depthai*), 261
 chipTemperature () (*depthai.RawSystemInformation property*), 328
 chipTemperature () (*depthai.SystemInformation property*), 342
 CHW (*depthai.TensorInfo.StorageOrder attribute*), 345
 classes () (*depthai.DetectionNetworkProperties property*), 273
 close () (*depthai.Device method*), 275
 close () (*depthai.DeviceBootloader method*), 281
 CLOUDY_DAYLIGHT (*depthai.CameraControl.AutoWhiteBalanceMode attribute*), 255
 CLOUDY_DAYLIGHT (*depthai.RawCameraControl.AutoWhiteBalanceMode attribute*), 317
 cmdMask () (*depthai.RawCameraControl property*), 320
 cmxMemoryUsage () (*depthai.RawSystemInformation property*), 328
 cmxMemoryUsage () (*depthai.SystemInformation property*), 342
 CN (*depthai.TensorInfo.StorageOrder attribute*), 345
 ColorCamera (*class in depthai*), 261
 ColorCameraProperties (*class in depthai*), 266
 ColorCameraProperties.ColorOrder (*class in depthai*), 267
 ColorCameraProperties.SensorResolution (*class in depthai*), 267
 colorOrder () (*depthai.ColorCameraProperties property*), 268
 confidence () (*depthai.ImgDetection property*), 288
 confidenceThreshold () (*depthai.DetectionNetworkProperties property*), 273
 confidenceThreshold () (*depthai.StereoDepthProperties property*), 342
 config () (*depthai.SpatialLocations property*), 337
 contains () (*depthai.Rect method*), 330
 CONTINUOUS_PICTURE (*depthai.CameraControl.AutoFocusMode attribute*), 254
 CONTINUOUS_PICTURE (*depthai.RawCameraControl.AutoFocusMode attribute*), 316
 CONTINUOUS_VIDEO (*depthai.CameraControl.AutoFocusMode attribute*), 254
 CONTINUOUS_VIDEO (*depthai.RawCameraControl.AutoFocusMode attribute*), 316
 coordinates () (*depthai.DetectionNetworkProperties property*), 273
 CpuUsage (*class in depthai*), 268
 createColorCamera () (*depthai.Pipeline method*), 312
 createDepthaiApplicationPackage () (*depthai.DeviceBootloader static method*), 281
 createImageManip () (*depthai.Pipeline method*), 312
 createMobileNetDetectionNetwork () (*depthai.Pipeline method*), 312
 createMobileNetSpatialDetectionNetwork () (*depthai.Pipeline method*), 312
 createMonoCamera () (*depthai.Pipeline method*), 312
 createMonocularNetwork () (*depthai.Pipeline method*), 312
 createMotionTracker () (*depthai.Pipeline method*), 312
 createSpatialLocationCalculator () (*depthai.Pipeline method*), 312
 createSPIOut () (*depthai.Pipeline method*), 312
 createStereoDepth () (*depthai.Pipeline method*), 312
 createSystemLogger () (*depthai.Pipeline method*), 312
 createVideoEncoder () (*depthai.Pipeline method*), 312
 createXLinkIn () (*depthai.Pipeline method*), 312
 createXLinkOut () (*depthai.Pipeline method*), 312
 createYoloDetectionNetwork () (*depthai.Pipeline method*), 312
 createYoloSpatialDetectionNetwork () (*depthai.Pipeline method*), 312
 CRITICAL (*depthai.LogLevel attribute*), 295
 cropConfig () (*depthai.RawImageManipConfig property*), 323
 cropRatio () (*depthai.RawImageManipConfig.CropConfig property*), 321
 cropRect () (*depthai.RawImageManipConfig.CropConfig property*), 321
 cropRotatedRect () (*depthai.RawImageManipConfig.CropConfig property*), 321
 css () (*depthai.ChipTemperature property*), 261
 CWH (*depthai.TensorInfo.StorageOrder attribute*), 345

D

dai (C++ type), 360
 dai::ADatatype (C++ class), 362
 dai::Asset (C++ struct), 362
 dai::AssetManager (C++ class), 362

dai::AssetManager::add (C++ *function*), 362
 dai::AssetManager::addExisting (C++ *function*), 362
 dai::AssetManager::get (C++ *function*), 363
 dai::AssetManager::getAll (C++ *function*), 363
 dai::AssetManager::remove (C++ *function*), 363
 dai::AssetManager::serialize (C++ *function*), 363
 dai::AssetManager::set (C++ *function*), 362
 dai::AssetManager::size (C++ *function*), 363
 dai::Assets (C++ *class*), 363
 dai::AssetsMutable (C++ *class*), 363
 dai::AssetView (C++ *struct*), 363
 dai::bootloader (C++ *type*), 399
 dai::bootloader::request (C++ *type*), 399
 dai::bootloader::request::BootApplication (C++ *struct*), 399
 dai::bootloader::request::Command (C++ *enum*), 399
 dai::bootloader::request::Command::BOOT_APPLICATION (C++ *enumerator*), 399
 dai::bootloader::request::Command::GET_BOOTLOADER_VERSION (C++ *enumerator*), 399
 dai::bootloader::request::Command::UPDATE_FLASH (C++ *enumerator*), 399
 dai::bootloader::request::Command::USB_ROM_BOOTLOADER (C++ *enumerator*), 399
 dai::bootloader::request::GetBootloaderVersion (C++ *struct*), 399
 dai::bootloader::request::UpdateFlash (C++ *struct*), 399
 dai::bootloader::request::UsbRomBoot (C++ *struct*), 399
 dai::bootloader::response (C++ *type*), 399
 dai::bootloader::response::BootloaderVersion (C++ *struct*), 399
 dai::bootloader::response::Command (C++ *enum*), 399
 dai::bootloader::response::Command::BOOTLOADER_VERSION (C++ *enumerator*), 399
 dai::bootloader::response::Command::FLASH_COMPLETED (C++ *enumerator*), 399
 dai::bootloader::response::Command::FLASH_STATUS_UPDATE (C++ *enumerator*), 399
 dai::bootloader::response::FlashCompleted (C++ *struct*), 399
 dai::bootloader::response::FlashStatusUpdate (C++ *struct*), 400
 dai::bootloader::XLINK_CHANNEL_BOOTLOADER (C++ *member*), 399
 dai::bootloader::XLINK_CHANNEL_WATCHDOG (C++ *member*), 399
 dai::bootloader::XLINK_STREAM_MAX_SIZE (C++ *member*), 399
 dai::bootloader::XLINK_WATCHDOG_TIMEOUT (C++ *member*), 399
 dai::Buffer (C++ *class*), 100, 363
 dai::Buffer::~~Buffer (C++ *function*), 100
 dai::Buffer::Buffer (C++ *function*), 100, 363
 dai::Buffer::getData (C++ *function*), 100, 363
 dai::Buffer::serialize (C++ *function*), 100
 dai::Buffer::setData (C++ *function*), 100, 363
 dai::build (C++ *type*), 400
 dai::build::VERSION (C++ *member*), 400
 dai::build::VERSION_MAJOR (C++ *member*), 400
 dai::build::VERSION_MINOR (C++ *member*), 400
 dai::build::VERSION_PATCH (C++ *member*), 400
 dai::CallbackHandler (C++ *class*), 363
 dai::CameraBoardSocket (C++ *enum*), 360
 dai::CameraBoardSocket::AUTO (C++ *enumerator*), 360
 dai::CameraBoardSocket::LEFT (C++ *enumerator*), 360
 dai::CameraBoardSocket::RGB (C++ *enumerator*), 360
 dai::CameraBoardSocket::RIGHT (C++ *enumerator*), 360
 dai::CameraControl (C++ *class*), 105, 363
 dai::CameraControl::~~CameraControl (C++ *function*), 105
 dai::CameraControl::AntiBandingMode (C++ *type*), 105
 dai::CameraControl::AutoFocusMode (C++ *type*), 105
 dai::CameraControl::AutoWhiteBalanceMode (C++ *type*), 105
 dai::CameraControl::CameraControl (C++ *function*), 105, 364
 dai::CameraControl::cfg (C++ *member*), 108
 dai::CameraControl::EffectMode (C++ *type*), 105
 dai::CameraControl::getCaptureStill (C++ *function*), 108, 366
 dai::CameraControl::SceneMode (C++ *type*), 105
 dai::CameraControl::serialize (C++ *function*), 108
 dai::CameraControl::setAntiBandingMode (C++ *function*), 106, 365
 dai::CameraControl::setAutoExposureCompensation (C++ *function*), 106, 364
 dai::CameraControl::setAutoExposureEnable (C++ *function*), 106, 364

dai::CameraControl::setAutoExposureLock (C++ function), 106, 364
 dai::CameraControl::setAutoExposureRegion (C++ function), 106, 364
 dai::CameraControl::setAutoFocusMode (C++ function), 105, 364
 dai::CameraControl::setAutoFocusRegion (C++ function), 106, 364
 dai::CameraControl::setAutoFocusTrigger (C++ function), 106, 364
 dai::CameraControl::setAutoWhiteBalanceLock (C++ function), 107, 365
 dai::CameraControl::setAutoWhiteBalanceMode (C++ function), 107, 365
 dai::CameraControl::setBrightness (C++ function), 107, 365
 dai::CameraControl::setCaptureStill (C++ function), 105, 364
 dai::CameraControl::setChromaDenoise (C++ function), 107, 365
 dai::CameraControl::setContrast (C++ function), 107, 365
 dai::CameraControl::setEffectMode (C++ function), 108, 366
 dai::CameraControl::setLumaDenoise (C++ function), 107, 365
 dai::CameraControl::setManualExposure (C++ function), 106, 365
 dai::CameraControl::setManualFocus (C++ function), 106, 364
 dai::CameraControl::setNoiseReductionStrength (C++ function), 107, 365
 dai::CameraControl::setSaturation (C++ function), 107, 365
 dai::CameraControl::setSceneMode (C++ function), 107, 366
 dai::CameraControl::setSharpness (C++ function), 107, 365
 dai::CameraControl::setStartStreaming (C++ function), 105, 364
 dai::CameraControl::setStopStreaming (C++ function), 105, 364
 dai::CameraImageOrientation (C++ enum), 360
 dai::CameraImageOrientation::AUTO (C++ enumerator), 360
 dai::CameraImageOrientation::HORIZONTAL_MIRROR (C++ enumerator), 360
 dai::CameraImageOrientation::NORMAL (C++ enumerator), 360
 dai::CameraImageOrientation::ROTATE_180_DEG (C++ enumerator), 360
 dai::CameraImageOrientation::VERTICAL_FLIP (C++ enumerator), 360
 dai::ChipTemperature (C++ struct), 366
 dai::ChipTemperature::average (C++ member), 366
 dai::ChipTemperature::css (C++ member), 366
 dai::ChipTemperature::dss (C++ member), 366
 dai::ChipTemperature::mss (C++ member), 366
 dai::ChipTemperature::upa (C++ member), 366
 dai::ColorCameraProperties (C++ struct), 366
 dai::ColorCameraProperties::boardSocket (C++ member), 367
 dai::ColorCameraProperties::ColorOrder (C++ enum), 366
 dai::ColorCameraProperties::colorOrder (C++ member), 367
 dai::ColorCameraProperties::ColorOrder::BGR (C++ enumerator), 366
 dai::ColorCameraProperties::ColorOrder::RGB (C++ enumerator), 366
 dai::ColorCameraProperties::fp16 (C++ member), 367
 dai::ColorCameraProperties::fps (C++ member), 367
 dai::ColorCameraProperties::imageOrientation (C++ member), 367
 dai::ColorCameraProperties::inputConfigSync (C++ member), 367
 dai::ColorCameraProperties::interleaved (C++ member), 367
 dai::ColorCameraProperties::previewHeight (C++ member), 367
 dai::ColorCameraProperties::previewKeepAspectRatio (C++ member), 367
 dai::ColorCameraProperties::previewWidth (C++ member), 367
 dai::ColorCameraProperties::resolution (C++ member), 367
 dai::ColorCameraProperties::sensorCropX (C++ member), 367
 dai::ColorCameraProperties::SensorResolution (C++ enum), 366
 dai::ColorCameraProperties::SensorResolution::THE_1_2 (C++ enumerator), 366
 dai::ColorCameraProperties::SensorResolution::THE_1_3 (C++ enumerator), 366
 dai::ColorCameraProperties::SensorResolution::THE_1_4 (C++ enumerator), 366
 dai::ColorCameraProperties::stillHeight (C++ member), 367
 dai::ColorCameraProperties::stillWidth (C++ member), 367

dai::ColorCameraProperties::videoHeight (C++ member), 367
 dai::ColorCameraProperties::videoWidth (C++ member), 367
 dai::CpuUsage (C++ struct), 367
 dai::CpuUsage::average (C++ member), 368
 dai::CpuUsage::msTime (C++ member), 368
 dai::DataInputQueue (C++ class), 368
 dai::DataInputQueue::getBlocking (C++ function), 368
 dai::DataInputQueue::getMaxDataSize (C++ function), 368
 dai::DataInputQueue::getMaxSize (C++ function), 368
 dai::DataInputQueue::getName (C++ function), 368
 dai::DataInputQueue::send (C++ function), 368, 369
 dai::DataInputQueue::setBlocking (C++ function), 368
 dai::DataInputQueue::setMaxDataSize (C++ function), 368
 dai::DataInputQueue::setMaxSize (C++ function), 368
 dai::DataOutputQueue (C++ class), 369
 dai::DataOutputQueue::addCallback (C++ function), 370
 dai::DataOutputQueue::CallbackId (C++ type), 369
 dai::DataOutputQueue::front (C++ function), 371
 dai::DataOutputQueue::get (C++ function), 370, 371
 dai::DataOutputQueue::getAll (C++ function), 371
 dai::DataOutputQueue::getBlocking (C++ function), 369
 dai::DataOutputQueue::getMaxSize (C++ function), 369
 dai::DataOutputQueue::getName (C++ function), 370
 dai::DataOutputQueue::has (C++ function), 370
 dai::DataOutputQueue::removeCallback (C++ function), 370
 dai::DataOutputQueue::setBlocking (C++ function), 369
 dai::DataOutputQueue::setMaxSize (C++ function), 369
 dai::DataOutputQueue::tryGet (C++ function), 370
 dai::DataOutputQueue::tryGetAll (C++ function), 371
 dai::DatatypeEnum (C++ enum), 361
 dai::DatatypeEnum::Buffer (C++ enumerator), 361
 dai::DatatypeEnum::CameraControl (C++ enumerator), 361
 dai::DatatypeEnum::ImageManipConfig (C++ enumerator), 361
 dai::DatatypeEnum::ImgDetections (C++ enumerator), 361
 dai::DatatypeEnum::ImgFrame (C++ enumerator), 361
 dai::DatatypeEnum::NNData (C++ enumerator), 361
 dai::DatatypeEnum::SpatialImgDetections (C++ enumerator), 361
 dai::DatatypeEnum::SpatialLocationCalculatorConfig (C++ enumerator), 361
 dai::DatatypeEnum::SpatialLocationCalculatorData (C++ enumerator), 361
 dai::DatatypeEnum::SystemInformation (C++ enumerator), 361
 dai::DatatypeEnum::Tracklets (C++ enumerator), 361
 dai::DetectionNetworkProperties (C++ struct), 372
 dai::DetectionNetworkProperties::classes (C++ member), 372
 dai::DetectionNetworkProperties::nnFamily (C++ member), 372
 dai::Device (C++ class), 20, 372
 dai::Device::~~Device (C++ function), 21, 373
 dai::Device::addLogCallback (C++ function), 22, 373
 dai::Device::assets (C++ member), 28
 dai::Device::assetStorage (C++ member), 28
 dai::Device::callbackIdMap (C++ member), 27
 dai::Device::checkClosed (C++ function), 27
 dai::Device::client (C++ member), 27
 dai::Device::close (C++ function), 26, 376
 dai::Device::closed (C++ member), 28
 dai::Device::connection (C++ member), 27
 dai::Device::DEFAULT_SEARCH_TIME (C++ member), 27, 377
 dai::Device::DEFAULT_SYSTEM_INFORMATION_LOGGING_RATE (C++ member), 27, 377
 dai::Device::Device (C++ function), 20, 21, 372, 373
 dai::Device::deviceInfo (C++ member), 27
 dai::Device::EVENT_QUEUE_MAXIMUM_SIZE (C++ member), 27, 377
 dai::Device::eventCv (C++ member), 27
 dai::Device::eventMtx (C++ member), 27
 dai::Device::eventQueue (C++ member), 27
 dai::Device::getAllAvailableDevices

(C++ function), 26, 377

dai::Device::getAnyAvailableDevice (C++ function), 26, 376, 377

dai::Device::getChipTemperature (C++ function), 25, 376

dai::Device::getCmxMemoryUsage (C++ function), 25, 376

dai::Device::getDdrMemoryUsage (C++ function), 25, 376

dai::Device::getDeviceByMxId (C++ function), 26, 377

dai::Device::getEmbeddedDeviceBinary (C++ function), 26, 377

dai::Device::getFirstAvailableDevice (C++ function), 26, 377

dai::Device::getInputQueue (C++ function), 23, 374

dai::Device::getInputQueueNames (C++ function), 23, 374

dai::Device::getLeonCssCpuUsage (C++ function), 25, 376

dai::Device::getLeonCssHeapUsage (C++ function), 25, 376

dai::Device::getLeonMssCpuUsage (C++ function), 25, 376

dai::Device::getLeonMssHeapUsage (C++ function), 25, 376

dai::Device::getLogLevel (C++ function), 22, 373

dai::Device::getLogOutputLevel (C++ function), 22, 373

dai::Device::getOutputQueue (C++ function), 22, 23, 374

dai::Device::getOutputQueueNames (C++ function), 23, 374

dai::Device::getQueueEvent (C++ function), 24, 25, 375, 376

dai::Device::getQueueEvents (C++ function), 23, 24, 375

dai::Device::getSystemInformationLoggingRate (C++ function), 22, 374

dai::Device::init (C++ function), 27

dai::Device::inputQueueMap (C++ member), 27

dai::Device::isClosed (C++ function), 26, 376

dai::Device::isPipelineRunning (C++ function), 21, 373

dai::Device::logCallbackMap (C++ member), 27

dai::Device::logCallbackMapMtx (C++ member), 27

dai::Device::loggingRunning (C++ member), 28

dai::Device::loggingThread (C++ member), 27

dai::Device::outputQueueMap (C++ member), 27

dai::Device::patchedCmd (C++ member), 27

dai::Device::pimpl (C++ member), 28

dai::Device::removeLogCallback (C++ function), 22, 373

dai::Device::rpcMutex (C++ member), 27

dai::Device::rpcStream (C++ member), 28

dai::Device::schema (C++ member), 28

dai::Device::setLogLevel (C++ function), 21, 373

dai::Device::setLogOutputLevel (C++ function), 22, 373

dai::Device::setSystemInformationLoggingRate (C++ function), 22, 374

dai::Device::startPipeline (C++ function), 21, 373

dai::Device::timesyncRunning (C++ member), 27

dai::Device::timesyncThread (C++ member), 27

dai::Device::uniqueCallbackId (C++ member), 27

dai::Device::version (C++ member), 28

dai::Device::watchdogRunning (C++ member), 27

dai::Device::watchdogThread (C++ member), 27

dai::DeviceBootloader (C++ class), 377

dai::DeviceBootloader::close (C++ function), 378

dai::DeviceBootloader::createDepthaiApplicationPack (C++ function), 379

dai::DeviceBootloader::DeviceBootloader (C++ function), 378

dai::DeviceBootloader::flash (C++ function), 378

dai::DeviceBootloader::flashBootloader (C++ function), 378

dai::DeviceBootloader::flashDepthaiApplicationPack (C++ function), 378

dai::DeviceBootloader::getAllAvailableDevices (C++ function), 379

dai::DeviceBootloader::getEmbeddedBootloaderBinary (C++ function), 379

dai::DeviceBootloader::getEmbeddedBootloaderVersion (C++ function), 379

dai::DeviceBootloader::getFirstAvailableDevice (C++ function), 379

dai::DeviceBootloader::getVersion (C++ function), 378

dai::DeviceBootloader::isClosed (C++ function), 378

dai::DeviceBootloader::isEmbeddedVersion (C++ function), 112, 381
 (C++ function), 378
 dai::DeviceBootloader::saveDepthaiApplicationParameters (C++ function), 113, 381
 (C++ function), 379
 dai::DeviceBootloader::Version (C++ struct), 379
 dai::DeviceBootloader::Version::toString (C++ function), 379
 dai::DeviceBootloader::Version::Version (C++ function), 379
 dai::DeviceInfo (C++ struct), 379
 dai::GlobalProperties (C++ struct), 379
 dai::GlobalProperties::leonCssFrequencyHz (C++ member), 380
 dai::GlobalProperties::leonMssFrequencyHz (C++ member), 380
 dai::ImageManipConfig (C++ class), 111, 380
 dai::ImageManipConfig::~ImageManipConfig (C++ function), 111
 dai::ImageManipConfig::cfg (C++ member), 114
 dai::ImageManipConfig::getCropXMax (C++ function), 113, 382
 dai::ImageManipConfig::getCropXMin (C++ function), 113, 381
 dai::ImageManipConfig::getCropYMax (C++ function), 113, 382
 dai::ImageManipConfig::getCropYMin (C++ function), 113, 382
 dai::ImageManipConfig::getResizeHeight (C++ function), 113, 382
 dai::ImageManipConfig::getResizeWidth (C++ function), 113, 382
 dai::ImageManipConfig::ImageManipConfig (C++ function), 111, 380
 dai::ImageManipConfig::isResizeThumbnail (C++ function), 113, 382
 dai::ImageManipConfig::serialize (C++ function), 114
 dai::ImageManipConfig::setCenterCrop (C++ function), 111, 380
 dai::ImageManipConfig::setCropRect (C++ function), 111, 380
 dai::ImageManipConfig::setCropRotatedRect (C++ function), 111, 380
 dai::ImageManipConfig::setFrameType (C++ function), 113, 381
 dai::ImageManipConfig::setHorizontalFlip (C++ function), 113, 381
 dai::ImageManipConfig::setKeepAspectRatio (C++ function), 113, 381
 dai::ImageManipConfig::setResize (C++ function), 112, 381
 dai::ImageManipConfig::setResizeThumbnail (C++ function), 112, 381
 dai::ImageManipConfig::setReusePreviousImage (C++ function), 113, 381
 dai::ImageManipConfig::setRotationDegrees (C++ function), 112, 381
 dai::ImageManipConfig::setRotationRadians (C++ function), 112, 381
 dai::ImageManipConfig::setSkipCurrentImage (C++ function), 113, 381
 dai::ImageManipConfig::setWarpBorderFillColor (C++ function), 112, 380
 dai::ImageManipConfig::setWarpBorderReplicatePixels (C++ function), 112, 380
 dai::ImageManipConfig::setWarpTransformFourPoints (C++ function), 111, 380
 dai::ImageManipConfig::setWarpTransformMatrix3x3 (C++ function), 112, 380
 dai::ImageManipProperties (C++ struct), 382
 dai::ImageManipProperties::initialConfig (C++ member), 382
 dai::ImageManipProperties::inputConfigSync (C++ member), 382
 dai::ImageManipProperties::numFramesPool (C++ member), 382
 dai::ImageManipProperties::outputFrameSize (C++ member), 382
 dai::ImgDetection (C++ struct), 382
 dai::ImgDetections (C++ class), 114, 382
 dai::ImgDetections::~ImgDetections (C++ function), 115
 dai::ImgDetections::detections (C++ member), 115, 383
 dai::ImgDetections::dets (C++ member), 115
 dai::ImgDetections::ImgDetections (C++ function), 115, 382
 dai::ImgDetections::serialize (C++ function), 115
 dai::ImgFrame (C++ class), 118, 383
 dai::ImgFrame::~ImgFrame (C++ function), 118
 dai::ImgFrame::getCategory (C++ function), 118, 383
 dai::ImgFrame::getCvFrame (C++ function), 119, 384
 dai::ImgFrame::getFrame (C++ function), 119, 384
 dai::ImgFrame::getHeight (C++ function), 118, 383
 dai::ImgFrame::getInstanceNum (C++ function), 118, 383
 dai::ImgFrame::getSequenceNum (C++ function), 118, 383
 dai::ImgFrame::getTimestamp (C++ function), 118, 383
 dai::ImgFrame::getType (C++ function), 118, 383

383
dai::ImgFrame::getWidth (C++ *function*), 118, 383
dai::ImgFrame::img (C++ *member*), 120
dai::ImgFrame::ImgFrame (C++ *function*), 118, 383
dai::ImgFrame::serialize (C++ *function*), 120
dai::ImgFrame::setCategory (C++ *function*), 118, 383
dai::ImgFrame::setFrame (C++ *function*), 119, 384
dai::ImgFrame::setHeight (C++ *function*), 119, 383
dai::ImgFrame::setInstanceNum (C++ *function*), 118, 383
dai::ImgFrame::setSequenceNum (C++ *function*), 118, 383
dai::ImgFrame::setTimestamp (C++ *function*), 118, 383
dai::ImgFrame::setType (C++ *function*), 119, 384
dai::ImgFrame::setWidth (C++ *function*), 119, 383
dai::ImgFrame::Specs (C++ *type*), 118
dai::ImgFrame::Type (C++ *type*), 118
dai::initialize (C++ *function*), 362
dai::isDatatypeSubclassOf (C++ *function*), 362
dai::LockingQueue (C++ *class*), 384
dai::LOG_DEFAULT_PATTERN (C++ *member*), 362
dai::LogLevel (C++ *enum*), 361
dai::LogLevel::CRITICAL (C++ *enumerator*), 361
dai::LogLevel::DEBUG (C++ *enumerator*), 361
dai::LogLevel::ERR (C++ *enumerator*), 361
dai::LogLevel::INFO (C++ *enumerator*), 361
dai::LogLevel::OFF (C++ *enumerator*), 361
dai::LogLevel::TRACE (C++ *enumerator*), 361
dai::LogLevel::WARN (C++ *enumerator*), 361
dai::LogMessage (C++ *struct*), 384
dai::MemoryInfo (C++ *struct*), 384
dai::MonoCameraProperties (C++ *struct*), 384
dai::MonoCameraProperties::boardSocket (C++ *member*), 385
dai::MonoCameraProperties::fps (C++ *member*), 385
dai::MonoCameraProperties::imageOrientation (C++ *member*), 385
dai::MonoCameraProperties::resolution (C++ *member*), 385
dai::MonoCameraProperties::SensorResolution (C++ *enum*), 384
dai::MonoCameraProperties::SensorResolution::THE_720P (C++ *enumerator*), 384
dai::MonoCameraProperties::SensorResolution::THE_800P (C++ *enumerator*), 384
dai::MyConsumerProperties (C++ *struct*), 385
dai::MyConsumerProperties::processorPlacement (C++ *member*), 385
dai::MyProducerProperties (C++ *struct*), 385
dai::MyProducerProperties::message (C++ *member*), 385
dai::MyProducerProperties::processorPlacement (C++ *member*), 385
dai::NeuralNetworkProperties (C++ *struct*), 385
dai::NeuralNetworkProperties::blobSize (C++ *member*), 385
dai::NeuralNetworkProperties::blobUri (C++ *member*), 385
dai::NeuralNetworkProperties::numFrames (C++ *member*), 385
dai::NeuralNetworkProperties::numNCEPerThread (C++ *member*), 385
dai::NeuralNetworkProperties::numThreads (C++ *member*), 385
dai::NNData (C++ *class*), 122, 385
dai::NNData::~~NNData (C++ *function*), 122
dai::NNData::DATA_ALIGNMENT (C++ *member*), 124
dai::NNData::fp16Data (C++ *member*), 124
dai::NNData::getAllLayerNames (C++ *function*), 123, 386
dai::NNData::getAllLayers (C++ *function*), 123, 386
dai::NNData::getFirstLayerFp16 (C++ *function*), 124, 387
dai::NNData::getFirstLayerInt32 (C++ *function*), 124, 387
dai::NNData::getFirstLayerUInt8 (C++ *function*), 124, 387
dai::NNData::getLayer (C++ *function*), 123, 386
dai::NNData::getLayerDatatype (C++ *function*), 123, 386
dai::NNData::getLayerFp16 (C++ *function*), 123, 387
dai::NNData::getLayerInt32 (C++ *function*), 123, 387
dai::NNData::getLayerUInt8 (C++ *function*), 123, 386
dai::NNData::hasLayer (C++ *function*), 123, 386
dai::NNData::NNData (C++ *function*), 122, 386
dai::NNData::rawNn (C++ *member*), 124
dai::NNData::serialize (C++ *function*), 124
dai::NNData::setLayer (C++ *function*), 122, 386
dai::NNData::u8Data (C++ *member*), 124

dai::Node (C++ class), 387
 dai::node (C++ type), 400
 dai::node::ColorCamera (C++ class), 40, 400
 dai::node::ColorCamera::clone (C++ function), 43
 dai::node::ColorCamera::ColorCamera (C++ function), 40, 400
 dai::node::ColorCamera::getBoardSocket (C++ function), 40, 400
 dai::node::ColorCamera::getCamId (C++ function), 40, 400
 dai::node::ColorCamera::getColorOrder (C++ function), 41, 400
 dai::node::ColorCamera::getFp16 (C++ function), 41, 401
 dai::node::ColorCamera::getFps (C++ function), 41, 401
 dai::node::ColorCamera::getImageOrientation (C++ function), 40, 400
 dai::node::ColorCamera::getInputs (C++ function), 43
 dai::node::ColorCamera::getInterleaved (C++ function), 41, 400
 dai::node::ColorCamera::getName (C++ function), 43
 dai::node::ColorCamera::getOutputs (C++ function), 43
 dai::node::ColorCamera::getPreviewHeight (C++ function), 41, 401
 dai::node::ColorCamera::getPreviewKeepAspectRatio (C++ function), 42, 402
 dai::node::ColorCamera::getPreviewSize (C++ function), 41, 401
 dai::node::ColorCamera::getPreviewWidth (C++ function), 41, 401
 dai::node::ColorCamera::getProperties (C++ function), 43
 dai::node::ColorCamera::getResolution (C++ function), 41, 401
 dai::node::ColorCamera::getResolutionHeight (C++ function), 42, 401
 dai::node::ColorCamera::getResolutionSize (C++ function), 42, 401
 dai::node::ColorCamera::getResolutionWidth (C++ function), 42, 401
 dai::node::ColorCamera::getSensorCrop (C++ function), 42, 402
 dai::node::ColorCamera::getSensorCropX (C++ function), 42, 402
 dai::node::ColorCamera::getSensorCropY (C++ function), 42, 402
 dai::node::ColorCamera::getStillHeight (C++ function), 42, 401
 dai::node::ColorCamera::getStillSize (C++ function), 41, 401
 dai::node::ColorCamera::getStillWidth (C++ function), 41, 401
 dai::node::ColorCamera::getVideoHeight (C++ function), 41, 401
 dai::node::ColorCamera::getVideoSize (C++ function), 41, 401
 dai::node::ColorCamera::getVideoWidth (C++ function), 41, 401
 dai::node::ColorCamera::getWaitForConfigInput (C++ function), 42, 402
 dai::node::ColorCamera::initialControl (C++ member), 43, 402
 dai::node::ColorCamera::inputConfig (C++ member), 43, 402
 dai::node::ColorCamera::inputControl (C++ member), 43, 402
 dai::node::ColorCamera::preview (C++ member), 43, 402
 dai::node::ColorCamera::properties (C++ member), 43
 dai::node::ColorCamera::Properties (C++ type), 40
 dai::node::ColorCamera::rawControl (C++ member), 43
 dai::node::ColorCamera::sensorCenterCrop (C++ function), 42, 401
 dai::node::ColorCamera::setBoardSocket (C++ function), 40, 400
 dai::node::ColorCamera::setCamId (C++ function), 40, 400
 dai::node::ColorCamera::setColorOrder (C++ function), 41, 400
 dai::node::ColorCamera::setFp16 (C++ function), 41, 400
 dai::node::ColorCamera::setFps (C++ function), 41, 401
 dai::node::ColorCamera::setImageOrientation (C++ function), 40, 400
 dai::node::ColorCamera::setInterleaved (C++ function), 41, 400
 dai::node::ColorCamera::setPreviewKeepAspectRatio (C++ function), 42, 402
 dai::node::ColorCamera::setPreviewSize (C++ function), 41, 401
 dai::node::ColorCamera::setResolution (C++ function), 41, 401
 dai::node::ColorCamera::setSensorCrop (C++ function), 42, 402
 dai::node::ColorCamera::setStillSize (C++ function), 41, 401
 dai::node::ColorCamera::setVideoSize (C++ function), 41, 401
 dai::node::ColorCamera::setWaitForConfigInput (C++ function), 42, 402

(C++ function), [42](#), [402](#)
 dai::node::ColorCamera::still (C++ member), [43](#), [403](#)
 dai::node::ColorCamera::video (C++ member), [43](#), [402](#)
 dai::Node::Connection (C++ struct), [388](#)
 dai::node::DetectionNetwork (C++ class), [403](#)
 dai::node::DetectionNetwork::input (C++ member), [403](#)
 dai::node::DetectionNetwork::out (C++ member), [403](#)
 dai::node::DetectionNetwork::passthrough (C++ member), [403](#)
 dai::node::DetectionNetwork::setConfidenceThreshold (C++ function), [403](#)
 dai::Node::getAssets (C++ function), [387](#)
 dai::Node::getInputs (C++ function), [387](#)
 dai::Node::getName (C++ function), [387](#)
 dai::Node::getOutputs (C++ function), [387](#)
 dai::Node::id (C++ member), [388](#)
 dai::Node::Id (C++ type), [387](#)
 dai::node::ImageManip (C++ class), [46](#), [403](#)
 dai::node::ImageManip::clone (C++ function), [47](#)
 dai::node::ImageManip::getInputs (C++ function), [47](#)
 dai::node::ImageManip::getName (C++ function), [47](#)
 dai::node::ImageManip::getOutputs (C++ function), [47](#)
 dai::node::ImageManip::getProperties (C++ function), [47](#)
 dai::node::ImageManip::ImageManip (C++ function), [46](#)
 dai::node::ImageManip::initialConfig (C++ member), [47](#), [404](#)
 dai::node::ImageManip::inputConfig (C++ member), [47](#), [404](#)
 dai::node::ImageManip::inputImage (C++ member), [47](#), [404](#)
 dai::node::ImageManip::out (C++ member), [47](#), [404](#)
 dai::node::ImageManip::properties (C++ member), [47](#)
 dai::node::ImageManip::Properties (C++ type), [46](#)
 dai::node::ImageManip::rawConfig (C++ member), [47](#)
 dai::node::ImageManip::setCenterCrop (C++ function), [46](#)
 dai::node::ImageManip::setCropRect (C++ function), [46](#)
 dai::node::ImageManip::setFrameType (C++ function), [46](#)
 dai::node::ImageManip::setHorizontalFlip (C++ function), [46](#)
 dai::node::ImageManip::setKeepAspectRatio (C++ function), [46](#)
 dai::node::ImageManip::setMaxOutputFrameSize (C++ function), [46](#), [403](#)
 dai::node::ImageManip::setNumFramesPool (C++ function), [46](#), [403](#)
 dai::node::ImageManip::setResize (C++ function), [46](#)
 dai::node::ImageManip::setResizeThumbnail (C++ function), [46](#)
 dai::node::ImageManip::setWaitForConfigInput (C++ function), [46](#), [403](#)
 dai::node::MobileNetDetectionNetwork (C++ class), [50](#), [404](#)
 dai::node::MobileNetDetectionNetwork::MobileNetDetectionNetwork (C++ function), [50](#)
 dai::node::MobileNetSpatialDetectionNetwork (C++ class), [53](#), [404](#)
 dai::node::MobileNetSpatialDetectionNetwork::MobileNetSpatialDetectionNetwork (C++ function), [54](#)
 dai::node::MonoCamera (C++ class), [56](#), [404](#)
 dai::node::MonoCamera::clone (C++ function), [58](#)
 dai::node::MonoCamera::getBoardSocket (C++ function), [56](#), [404](#)
 dai::node::MonoCamera::getCamId (C++ function), [57](#)
 dai::node::MonoCamera::getFps (C++ function), [57](#), [404](#)
 dai::node::MonoCamera::getImageOrientation (C++ function), [57](#), [404](#)
 dai::node::MonoCamera::getInputs (C++ function), [58](#)
 dai::node::MonoCamera::getName (C++ function), [58](#)
 dai::node::MonoCamera::getOutputs (C++ function), [58](#)
 dai::node::MonoCamera::getProperties (C++ function), [58](#)
 dai::node::MonoCamera::getResolution (C++ function), [57](#), [404](#)
 dai::node::MonoCamera::getResolutionHeight (C++ function), [57](#), [405](#)
 dai::node::MonoCamera::getResolutionSize (C++ function), [57](#), [404](#)
 dai::node::MonoCamera::getResolutionWidth (C++ function), [57](#), [405](#)
 dai::node::MonoCamera::initialControl (C++ member), [57](#), [405](#)
 dai::node::MonoCamera::inputControl (C++ member), [57](#), [405](#)

dai::node::MonoCamera::MonoCamera (C++ function), 56
 dai::node::MonoCamera::out (C++ member), 57, 405
 dai::node::MonoCamera::properties (C++ member), 58
 dai::node::MonoCamera::Properties (C++ type), 56
 dai::node::MonoCamera::rawControl (C++ member), 58
 dai::node::MonoCamera::setBoardSocket (C++ function), 56, 404
 dai::node::MonoCamera::setCamId (C++ function), 57
 dai::node::MonoCamera::setFps (C++ function), 57, 404
 dai::node::MonoCamera::setImageOrientation (C++ function), 57, 404
 dai::node::MonoCamera::setResolution (C++ function), 57, 404
 dai::node::MyProducer (C++ class), 405
 dai::node::NeuralNetwork (C++ class), 61, 405
 dai::node::NeuralNetwork::clone (C++ function), 62
 dai::node::NeuralNetwork::getInputs (C++ function), 62
 dai::node::NeuralNetwork::getName (C++ function), 62
 dai::node::NeuralNetwork::getNumInferenceThreads (C++ function), 62, 405
 dai::node::NeuralNetwork::getOutputs (C++ function), 62
 dai::node::NeuralNetwork::getProperties (C++ function), 62
 dai::node::NeuralNetwork::getRequiredOpenVINOVersion (C++ function), 62
 dai::node::NeuralNetwork::input (C++ member), 62, 406
 dai::node::NeuralNetwork::NeuralNetwork (C++ function), 61
 dai::node::NeuralNetwork::out (C++ member), 62, 406
 dai::node::NeuralNetwork::passthrough (C++ member), 62, 406
 dai::node::NeuralNetwork::properties (C++ member), 62
 dai::node::NeuralNetwork::Properties (C++ type), 61
 dai::node::NeuralNetwork::setBlobPath (C++ function), 61, 405
 dai::node::NeuralNetwork::setNumInferenceThreads (C++ function), 61, 405
 dai::node::NeuralNetwork::setNumNCEPerInferenceThread (C++ function), 62, 405
 dai::node::NeuralNetwork::setNumPoolFrames (C++ function), 61, 405
 dai::node::ObjectTracker (C++ class), 406
 dai::node::ObjectTracker::inputDetectionFrame (C++ member), 406
 dai::node::ObjectTracker::inputDetections (C++ member), 406
 dai::node::ObjectTracker::inputTrackerFrame (C++ member), 406
 dai::node::ObjectTracker::out (C++ member), 407
 dai::node::ObjectTracker::passthroughDetectionFrame (C++ member), 407
 dai::node::ObjectTracker::passthroughDetections (C++ member), 407
 dai::node::ObjectTracker::passthroughTrackerFrame (C++ member), 407
 dai::node::ObjectTracker::setDetectionLabelsToTrack (C++ function), 406
 dai::node::ObjectTracker::setMaxObjectsToTrack (C++ function), 406
 dai::node::ObjectTracker::setTrackerIdAssignmentPolicy (C++ function), 406
 dai::node::ObjectTracker::setTrackerThreshold (C++ function), 406
 dai::node::ObjectTracker::setTrackerType (C++ function), 406
 dai::node::SpatialDetectionNetwork (C++ class), 407
 dai::node::SpatialDetectionNetwork::boundingBoxMapping (C++ member), 407
 dai::node::SpatialDetectionNetwork::input (C++ member), 407
 dai::node::SpatialDetectionNetwork::inputDepth (C++ member), 407
 dai::node::SpatialDetectionNetwork::out (C++ member), 407
 dai::node::SpatialDetectionNetwork::passthrough (C++ member), 408
 dai::node::SpatialDetectionNetwork::passthroughDepth (C++ member), 408
 dai::node::SpatialDetectionNetwork::setBoundingBoxMapping (C++ function), 407
 dai::node::SpatialDetectionNetwork::setDepthLowerThreshold (C++ function), 407
 dai::node::SpatialDetectionNetwork::setDepthUpperThreshold (C++ function), 407
 dai::node::SpatialLocationCalculator (C++ class), 65, 408
 dai::node::SpatialLocationCalculator::clone (C++ function), 66
 dai::node::SpatialLocationCalculator::getInputs (C++ function), 66
 dai::node::SpatialLocationCalculator::getName

(C++ function), 66
 dai::node::SpatialLocationCalculator::getInputs (C++ function), 66
 dai::node::SpatialLocationCalculator::getProperties (C++ function), 66
 dai::node::SpatialLocationCalculator::index (C++ member), 66, 408
 dai::node::SpatialLocationCalculator::input (C++ member), 66, 408
 dai::node::SpatialLocationCalculator::inputDepth (C++ member), 66, 408
 dai::node::SpatialLocationCalculator::output (C++ member), 66, 408
 dai::node::SpatialLocationCalculator::passThroughDepth (C++ member), 66, 408
 dai::node::SpatialLocationCalculator::properties (C++ member), 66
 dai::node::SpatialLocationCalculator::Property (C++ type), 65
 dai::node::SpatialLocationCalculator::rawConfig (C++ member), 66
 dai::node::SpatialLocationCalculator::setWaitForConfig (C++ function), 65, 408
 dai::node::SpatialLocationCalculator::SpatialNode (C++ function), 65
 dai::node::SPIOut (C++ class), 68, 408
 dai::node::SPIOut::clone (C++ function), 69
 dai::node::SPIOut::getInputs (C++ function), 69
 dai::node::SPIOut::getName (C++ function), 69
 dai::node::SPIOut::getOutputs (C++ function), 69
 dai::node::SPIOut::getProperties (C++ function), 69
 dai::node::SPIOut::input (C++ member), 69, 409
 dai::node::SPIOut::properties (C++ member), 69
 dai::node::SPIOut::setBusId (C++ function), 68, 408
 dai::node::SPIOut::setStreamName (C++ function), 68, 408
 dai::node::SPIOut::SPIOut (C++ function), 68
 dai::node::StereoDepth (C++ class), 73, 409
 dai::node::StereoDepth::clone (C++ function), 75
 dai::node::StereoDepth::depth (C++ member), 75, 410
 dai::node::StereoDepth::disparity (C++ member), 75, 410
 dai::node::StereoDepth::getInputs (C++ function), 75
 dai::node::StereoDepth::getName (C++ function), 75
 dai::node::StereoDepth::getOutputs (C++ function), 75
 dai::node::StereoDepth::getProperties (C++ function), 75
 dai::node::StereoDepth::left (C++ member), 75, 410
 dai::node::StereoDepth::loadCalibrationData (C++ function), 73, 409
 dai::node::StereoDepth::loadCalibrationFile (C++ function), 73, 409
 dai::node::StereoDepth::properties (C++ member), 75
 dai::node::StereoDepth::Properties (C++ type), 73
 dai::node::StereoDepth::rectifiedLeft (C++ member), 75, 410
 dai::node::StereoDepth::rectifiedRight (C++ member), 75, 410
 dai::node::StereoDepth::right (C++ member), 75, 410
 dai::node::StereoDepth::setConfidenceThreshold (C++ function), 74, 409
 dai::node::StereoDepth::setEmptyCalibration (C++ function), 74, 409
 dai::node::StereoDepth::setExtendedDisparity (C++ function), 74, 409
 dai::node::StereoDepth::setInputResolution (C++ function), 74, 409
 dai::node::StereoDepth::setLeftRightCheck (C++ function), 74, 409
 dai::node::StereoDepth::setMedianFilter (C++ function), 74, 409
 dai::node::StereoDepth::setOutputDepth (C++ function), 74, 410
 dai::node::StereoDepth::setOutputRectified (C++ function), 74, 410
 dai::node::StereoDepth::setRectifyEdgeFillColor (C++ function), 74, 409
 dai::node::StereoDepth::setRectifyMirrorFrame (C++ function), 74, 409
 dai::node::StereoDepth::setSubpixel (C++ function), 74, 409
 dai::node::StereoDepth::StereoDepth (C++ function), 73
 dai::node::StereoDepth::syncedLeft (C++ member), 75, 410
 dai::node::StereoDepth::syncedRight (C++ member), 75, 410
 dai::node::SystemLogger (C++ class), 77, 410
 dai::node::SystemLogger::clone (C++ function), 78
 dai::node::SystemLogger::getInputs (C++ function), 78

dai::node::SystemLogger::getName (C++ function), 78
 dai::node::SystemLogger::getOutputs (C++ function), 78
 dai::node::SystemLogger::getProperties (C++ function), 78
 dai::node::SystemLogger::out (C++ member), 78, 411
 dai::node::SystemLogger::properties (C++ member), 78
 dai::node::SystemLogger::setRate (C++ function), 77, 410
 dai::node::SystemLogger::SystemLogger (C++ function), 77
 dai::node::VideoEncoder (C++ class), 82, 411
 dai::node::VideoEncoder::bitstream (C++ member), 84, 413
 dai::node::VideoEncoder::clone (C++ function), 84
 dai::node::VideoEncoder::getBitrate (C++ function), 84, 412
 dai::node::VideoEncoder::getBitrateKbps (C++ function), 84, 412
 dai::node::VideoEncoder::getFrameRate (C++ function), 84, 412
 dai::node::VideoEncoder::getHeight (C++ function), 84, 412
 dai::node::VideoEncoder::getInputs (C++ function), 84
 dai::node::VideoEncoder::getKeyframeFrequency (C++ function), 84, 412
 dai::node::VideoEncoder::getLossless (C++ function), 84, 412
 dai::node::VideoEncoder::getName (C++ function), 84
 dai::node::VideoEncoder::getNumBFrames (C++ function), 84, 412
 dai::node::VideoEncoder::getNumFramesPool (C++ function), 83, 411
 dai::node::VideoEncoder::getOutputs (C++ function), 84
 dai::node::VideoEncoder::getProfile (C++ function), 84, 412
 dai::node::VideoEncoder::getProperties (C++ function), 84
 dai::node::VideoEncoder::getQuality (C++ function), 84, 412
 dai::node::VideoEncoder::getRateControlMode (C++ function), 84, 412
 dai::node::VideoEncoder::getSize (C++ function), 84, 412
 dai::node::VideoEncoder::getWidth (C++ function), 84, 412
 dai::node::VideoEncoder::input (C++ member), 84, 413
 dai::node::VideoEncoder::properties (C++ member), 85
 dai::node::VideoEncoder::Properties (C++ type), 82
 dai::node::VideoEncoder::setBitrate (C++ function), 83, 411
 dai::node::VideoEncoder::setBitrateKbps (C++ function), 83, 411
 dai::node::VideoEncoder::setDefaultProfilePreset (C++ function), 82, 411
 dai::node::VideoEncoder::setFrameRate (C++ function), 83, 412
 dai::node::VideoEncoder::setKeyframeFrequency (C++ function), 83, 411
 dai::node::VideoEncoder::setLossless (C++ function), 83, 412
 dai::node::VideoEncoder::setNumBFrames (C++ function), 83, 412
 dai::node::VideoEncoder::setNumFramesPool (C++ function), 83, 411
 dai::node::VideoEncoder::setProfile (C++ function), 83, 411
 dai::node::VideoEncoder::setQuality (C++ function), 83, 412
 dai::node::VideoEncoder::setRateControlMode (C++ function), 83, 411
 dai::node::VideoEncoder::VideoEncoder (C++ function), 82
 dai::node::XLinkIn (C++ class), 87, 413
 dai::node::XLinkIn::clone (C++ function), 88
 dai::node::XLinkIn::getInputs (C++ function), 88
 dai::node::XLinkIn::getMaxDataSize (C++ function), 88, 413
 dai::node::XLinkIn::getName (C++ function), 88
 dai::node::XLinkIn::getNumFrames (C++ function), 88, 413
 dai::node::XLinkIn::getOutputs (C++ function), 88
 dai::node::XLinkIn::getProperties (C++ function), 88
 dai::node::XLinkIn::getStreamName (C++ function), 88, 413
 dai::node::XLinkIn::out (C++ member), 88, 413
 dai::node::XLinkIn::properties (C++ member), 88
 dai::node::XLinkIn::setMaxDataSize (C++ function), 87, 413
 dai::node::XLinkIn::setNumFrames (C++ function), 87, 413
 dai::node::XLinkIn::setStreamName (C++

function), 87, 413
 dai::node::XLinkIn::XLinkIn (C++ *function*), 87
 dai::node::XLinkOut (C++ *class*), 90, 413
 dai::node::XLinkOut::clone (C++ *function*), 91
 dai::node::XLinkOut::getFpsLimit (C++ *function*), 91, 414
 dai::node::XLinkOut::getInputs (C++ *function*), 91
 dai::node::XLinkOut::getMetadataOnly (C++ *function*), 91, 414
 dai::node::XLinkOut::getName (C++ *function*), 91
 dai::node::XLinkOut::getOutputs (C++ *function*), 91
 dai::node::XLinkOut::getProperties (C++ *function*), 91
 dai::node::XLinkOut::getStreamName (C++ *function*), 91, 414
 dai::node::XLinkOut::input (C++ *member*), 91, 414
 dai::node::XLinkOut::properties (C++ *member*), 91
 dai::node::XLinkOut::setFpsLimit (C++ *function*), 90, 414
 dai::node::XLinkOut::setMetadataOnly (C++ *function*), 91, 414
 dai::node::XLinkOut::setStreamName (C++ *function*), 90, 414
 dai::node::XLinkOut::XLinkOut (C++ *function*), 90
 dai::node::YoloDetectionNetwork (C++ *class*), 94, 414
 dai::node::YoloDetectionNetwork::setAnchorMasks (C++ *enumerator*), 94, 414
 dai::node::YoloDetectionNetwork::setAnchors (C++ *enumerator*), 94, 414
 dai::node::YoloDetectionNetwork::setCoordinates (C++ *enumerator*), 94, 414
 dai::node::YoloDetectionNetwork::setIouThreshold (C++ *function*), 94, 414
 dai::node::YoloDetectionNetwork::setNumClasses (C++ *function*), 94, 414
 dai::node::YoloDetectionNetwork::YoloDetectionNetwork (C++ *member*), 94
 dai::node::YoloSpatialDetectionNetwork (C++ *class*), 98, 414
 dai::node::YoloSpatialDetectionNetwork::setAnchorMasks (C++ *function*), 99, 415
 dai::node::YoloSpatialDetectionNetwork::setAnchors (C++ *function*), 99, 415
 dai::node::YoloSpatialDetectionNetwork::setCoordinates (C++ *function*), 99, 415
 dai::node::YoloSpatialDetectionNetwork::setIouThreshold (C++ *function*), 99, 415
 dai::node::YoloSpatialDetectionNetwork::setNumClasses (C++ *function*), 99, 415
 dai::node::YoloSpatialDetectionNetwork::YoloSpatialDetectionNetwork (C++ *function*), 99
 dai::NodeConnectionSchema (C++ *struct*), 388
 dai::NodeIoInfo (C++ *struct*), 388
 dai::NodeObjInfo (C++ *struct*), 388
 dai::ObjectTrackerProperties (C++ *struct*), 388
 dai::OpenVINO (C++ *class*), 388
 dai::OpenVINO::areVersionsBlobCompatible (C++ *function*), 389
 dai::OpenVINO::getBlobLatestSupportedVersion (C++ *function*), 389
 dai::OpenVINO::getBlobSupportedVersions (C++ *function*), 388
 dai::OpenVINO::getVersionName (C++ *function*), 388
 dai::OpenVINO::getVersions (C++ *function*), 388
 dai::OpenVINO::parseVersionName (C++ *function*), 388
 dai::OpenVINO::Version (C++ *enum*), 388
 dai::OpenVINO::Version::VERSION_2020_1 (C++ *enumerator*), 388
 dai::OpenVINO::Version::VERSION_2020_2 (C++ *enumerator*), 388
 dai::OpenVINO::Version::VERSION_2020_3 (C++ *enumerator*), 388
 dai::OpenVINO::Version::VERSION_2020_4 (C++ *enumerator*), 388
 dai::OpenVINO::Version::VERSION_2021_1 (C++ *enumerator*), 388
 dai::OpenVINO::Version::VERSION_2021_2 (C++ *enumerator*), 388
 dai::OpenVINO::Version::VERSION_2021_3 (C++ *enumerator*), 388
 dai::operator<< (C++ *function*), 362
 dai::reshnet101 (C++ *class*), 389
 dai::Pipeline (C++ *class*), 32, 389
 dai::Pipeline::create (C++ *function*), 33, 389
 dai::Pipeline::DEFAULT_OPENVINO_VERSION (C++ *member*), 34, 390
 dai::Pipeline::getAllAssets (C++ *function*), 34, 390
 dai::Pipeline::getAllNodes (C++ *function*), 34, 390
 dai::Pipeline::getAssetManager (C++ *function*), 34, 390
 dai::Pipeline::getConnectionMap (C++ *function*), 33, 389
 dai::Pipeline::getConnections (C++ *function*), 33, 389

tion), 33, 389

dai::Pipeline::getGlobalProperties (C++ function), 33, 389

dai::Pipeline::getNode (C++ function), 33, 389

dai::Pipeline::getNodeMap (C++ function), 33, 390

dai::Pipeline::getPipelineSchema (C++ function), 33, 389

dai::Pipeline::impl (C++ function), 34

dai::Pipeline::link (C++ function), 33, 390

dai::Pipeline::NodeConnectionMap (C++ type), 33

dai::Pipeline::NodeMap (C++ type), 33

dai::Pipeline::pimpl (C++ member), 34

dai::Pipeline::Pipeline (C++ function), 33, 389

dai::Pipeline::remove (C++ function), 33, 389

dai::Pipeline::serialize (C++ function), 33

dai::Pipeline::setOpenVINOVersion (C++ function), 34, 390

dai::Pipeline::unlink (C++ function), 34, 390

dai::PipelineImpl (C++ class), 390

dai::PipelineSchema (C++ struct), 390

dai::Point2f (C++ struct), 390

dai::Point3f (C++ struct), 390

dai::ProcessorType (C++ enum), 360

dai::ProcessorType::LOS (C++ enumerator), 360

dai::ProcessorType::LRT (C++ enumerator), 361

dai::RawBuffer (C++ struct), 390

dai::RawCameraControl (C++ struct), 390

dai::RawCameraControl::lensPosition (C++ member), 391

dai::RawCameraControl::ManualExposureParams (C++ struct), 391

dai::RawCameraControl::RegionParams (C++ struct), 391

dai::RawImageManipConfig (C++ struct), 391

dai::RawImageManipConfig::CropConfig (C++ struct), 391

dai::RawImageManipConfig::CropRect (C++ struct), 391

dai::RawImageManipConfig::FormatConfig (C++ struct), 391

dai::RawImageManipConfig::ResizeConfig (C++ struct), 391

dai::RawImageManipConfig::ResizeConfig::keepAspectRatio (C++ function), 391

dai::RawImageManipConfig::ResizeConfig::keepAspectRatio (C++ member), 391

dai::RawImgDetections (C++ struct), 391

dai::RawImgFrame (C++ struct), 391

dai::RawImgFrame::Specs (C++ struct), 391

dai::RawNNData (C++ struct), 391

dai::RawSpatialImgDetections (C++ struct), 391

dai::RawSpatialLocationCalculatorConfig (C++ struct), 391

dai::RawSpatialLocations (C++ struct), 391

dai::RawSystemInformation (C++ struct), 391

dai::RawSystemInformation::chipTemperature (C++ member), 392

dai::RawSystemInformation::cmxMemoryUsage (C++ member), 392

dai::RawSystemInformation::ddrMemoryUsage (C++ member), 392

dai::RawSystemInformation::leonCssCpuUsage (C++ member), 392

dai::RawSystemInformation::leonCssMemoryUsage (C++ member), 392

dai::RawSystemInformation::leonMssCpuUsage (C++ member), 392

dai::RawSystemInformation::leonMssMemoryUsage (C++ member), 392

dai::RawTracklets (C++ struct), 392

dai::Rect (C++ struct), 392

dai::Rect::area (C++ function), 392

dai::Rect::bottomRight (C++ function), 392

dai::Rect::contains (C++ function), 392

dai::Rect::denormalize (C++ function), 392

dai::Rect::empty (C++ function), 392

dai::Rect::isNormalized (C++ function), 392

dai::Rect::normalize (C++ function), 392

dai::Rect::size (C++ function), 392

dai::Rect::topLeft (C++ function), 392

dai::RotatedRect (C++ struct), 393

dai::RotatedRect::angle (C++ member), 393

dai::Size2f (C++ struct), 393

dai::SpatialDetectionNetworkProperties (C++ struct), 393

dai::SpatialImgDetection (C++ struct), 393

dai::SpatialImgDetections (C++ class), 125, 393

dai::SpatialImgDetections::~~SpatialImgDetections (C++ function), 125

dai::SpatialImgDetections::detections (C++ member), 125, 393

dai::SpatialImgDetections::dets (C++ member), 126

dai::SpatialImgDetections::serialize (C++ function), 125

dai::SpatialImgDetections::SpatialImgDetections (C++ function), 125, 393

dai::SpatialLocationCalculatorConfig (C++ class), 126, 393

dai::SpatialLocationCalculatorConfig::~~SpatialLocationCalculatorConfig (C++ function), 127

dai::SpatialLocationCalculatorConfig::addROI (C++ function), 127, 393

dai::SpatialLocationCalculatorConfig::cfg (C++ member), 127
 dai::SpatialLocationCalculatorConfig::getConfig (C++ function), 127, 393
 dai::SpatialLocationCalculatorConfig::serialize (C++ function), 127
 dai::SpatialLocationCalculatorConfig::setROIs (C++ function), 127, 393
 dai::SpatialLocationCalculatorConfig::SpatialLocationCalculatorConfig (C++ function), 127, 393
 dai::SpatialLocationCalculatorConfigData (C++ struct), 394
 dai::SpatialLocationCalculatorConfigThresholds (C++ struct), 394
 dai::SpatialLocationCalculatorData (C++ class), 128, 394
 dai::SpatialLocationCalculatorData::~~SpatialLocationCalculatorData (C++ function), 128
 dai::SpatialLocationCalculatorData::getSpatialLocationCalculatorData (C++ function), 128, 394
 dai::SpatialLocationCalculatorData::rawdata (C++ member), 129
 dai::SpatialLocationCalculatorData::serialize (C++ function), 129
 dai::SpatialLocationCalculatorData::SpatialLocationCalculatorData (C++ function), 128, 394
 dai::SpatialLocationCalculatorData::spatialLocationCalculatorData (C++ member), 129
 dai::SpatialLocationCalculatorProperties (C++ struct), 394
 dai::SpatialLocationCalculatorProperties::SpatialLocationCalculatorProperties (C++ member), 394
 dai::SpatialLocations (C++ struct), 394
 dai::SPIOutProperties (C++ struct), 394
 dai::SPIOutProperties::busId (C++ member), 394
 dai::SPIOutProperties::streamName (C++ member), 394
 dai::StereoDepthProperties (C++ struct), 394
 dai::StereoDepthProperties::calibration (C++ member), 395
 dai::StereoDepthProperties::confidenceThreshold (C++ member), 395
 dai::StereoDepthProperties::enableExtendedDisplay (C++ member), 395
 dai::StereoDepthProperties::enableLeftRightCheck (C++ member), 395
 dai::StereoDepthProperties::enableOutputDepth (C++ member), 395
 dai::StereoDepthProperties::enableOutputRectified (C++ member), 395
 dai::StereoDepthProperties::enableSubpixel (C++ member), 395
 dai::StereoDepthProperties::height (C++ member), 395
 dai::StereoDepthProperties::median (C++ member), 395
 dai::StereoDepthProperties::MedianFilter (C++ member), 395
 dai::StereoDepthProperties::MedianFilter::KERNEL_3x3 (C++ member), 395
 dai::StereoDepthProperties::MedianFilter::KERNEL_5x5 (C++ member), 395
 dai::StereoDepthProperties::MedianFilter::KERNEL_7x7 (C++ member), 395
 dai::StereoDepthProperties::MedianFilter::MEDIAN_ONLY (C++ member), 395
 dai::StereoDepthProperties::rectifyEdgeFillColor (C++ member), 395
 dai::StereoDepthProperties::rectifyMirrorFrame (C++ member), 395
 dai::StereoDepthProperties::width (C++ member), 395
 dai::SystemInformation (C++ class), 129, 395
 dai::SystemInformation::~~SystemInformation (C++ function), 130
 dai::SystemInformation::chipTemperature (C++ member), 130
 dai::SystemInformation::dramMemoryUsage (C++ member), 130
 dai::SystemInformation::ddrMemoryUsage (C++ member), 130
 dai::SystemInformation::leonCssCpuUsage (C++ member), 130
 dai::SystemInformation::leonCssMemoryUsage (C++ member), 130
 dai::SystemInformation::leonMssCpuUsage (C++ member), 130
 dai::SystemInformation::leonMssMemoryUsage (C++ member), 130
 dai::SystemInformation::serialize (C++ function), 130
 dai::SystemInformation::SystemInformation (C++ function), 130, 396
 dai::SystemInformation::systemInformation (C++ member), 130
 dai::SystemLoggerProperties (C++ struct), 396
 dai::SystemLoggerProperties::rateHz (C++ member), 396
 dai::TensorInfo (C++ struct), 396
 dai::Timestamp (C++ struct), 396
 dai::TrackerIdAssignmentPolicy (C++ enum), 361
 dai::TrackerIdAssignmentPolicy::SMALLEST_ID (C++ member), 361
 dai::TrackerIdAssignmentPolicy::UNIQUE_ID (C++ member), 361

dai::TrackerType (C++ enum), 361	dai::VideoEncoderProperties::RateControlMode
dai::TrackerType::ZERO_TERM_COLOR_HISTOGRAM (C++ enumerator), 361	(C++ enum), 397
dai::TrackerType::ZERO_TERM_IMAGELESS (C++ enumerator), 361	dai::VideoEncoderProperties::RateControlMode::CBR (C++ enumerator), 397
dai::Tracklet (C++ struct), 396	dai::VideoEncoderProperties::RateControlMode::VBR (C++ enumerator), 397
dai::Tracklet::age (C++ member), 396	dai::VideoEncoderProperties::rateCtrlMode (C++ member), 398
dai::Tracklet::id (C++ member), 396	dai::VideoEncoderProperties::width (C++ member), 398
dai::Tracklet::label (C++ member), 396	dai::XLINK_CHANNEL_LOG (C++ member), 362
dai::Tracklet::roi (C++ member), 396	dai::XLINK_CHANNEL_MAIN_RPC (C++ member), 362
dai::Tracklet::spatialCoordinates (C++ member), 396	dai::XLINK_CHANNEL_PIPELINE_CONFIG (C++ member), 362
dai::Tracklet::srcImgDetection (C++ member), 396	dai::XLINK_CHANNEL_TIMESYNC (C++ member), 362
dai::Tracklet::status (C++ member), 396	dai::XLINK_USB_BUFFER_MAX_SIZE (C++ member), 362
dai::Tracklets (C++ class), 396	dai::XLINK_WATCHDOG_TIMEOUT (C++ member), 362
dai::Tracklets::Tracklets (C++ function), 397	dai::XLinkConnection (C++ class), 398
dai::Tracklets::tracklets (C++ member), 397	dai::XLinkConnection::close (C++ function), 398
dai::VideoEncoderProperties (C++ struct), 397	dai::XLinkConnection::isClosed (C++ function), 398
dai::VideoEncoderProperties::bitrate (C++ member), 397	dai::XLinkInProperties (C++ struct), 398
dai::VideoEncoderProperties::frameRate (C++ member), 398	dai::XLinkInProperties::maxDataSize
dai::VideoEncoderProperties::height (C++ member), 398	dai::XLinkInProperties::numFrames (C++ member), 398
dai::VideoEncoderProperties::keyframeFrequency (C++ member), 398	dai::XLinkInProperties::streamName (C++ member), 398
dai::VideoEncoderProperties::lossless (C++ member), 398	dai::XLinkOutProperties (C++ struct), 398
dai::VideoEncoderProperties::maxBitrate (C++ member), 397	dai::XLinkOutProperties::maxFpsLimit (C++ member), 399
dai::VideoEncoderProperties::numBFrames (C++ member), 397	dai::XLinkOutProperties::metadataOnly (C++ member), 399
dai::VideoEncoderProperties::numFramesPoc (C++ member), 397	dai::XLinkOutProperties::streamName (C++ member), 399
dai::VideoEncoderProperties::Profile (C++ enum), 397	dai::XLinkStream (C++ class), 399
dai::VideoEncoderProperties::profile (C++ member), 397	data () (depthai.Asset property), 249
dai::VideoEncoderProperties::Profile::H264 (C++ enumerator), 397	dataInputQueue (class in depthai), 269
dai::VideoEncoderProperties::Profile::H265 (C++ enumerator), 397	dataOutputQueue (class in depthai), 270
dai::VideoEncoderProperties::Profile::H265_High (C++ enumerator), 397	dataType () (depthai.TensorInfo property), 346
dai::VideoEncoderProperties::Profile::H265_MAIN (C++ enumerator), 397	depthai.CameraControl.AutoWhiteBalanceMode attribute), 255
dai::VideoEncoderProperties::Profile::H265_MAIN_High (C++ enumerator), 397	depthai.RawCameraControl.AutoWhiteBalanceMode attribute), 317
dai::VideoEncoderProperties::Profile::MJPEG (C++ enumerator), 397	depthai.RawSystemInformation property), 328
dai::VideoEncoderProperties::quality (C++ member), 397	ddrMemoryUsage () (depthai.SystemInformation property), 342

DEBUG (*depthai.LogLevel* attribute), 295
 denormalize() (*depthai.Rect* method), 330
 depth() (*depthai.StereoDepth* property), 339
 depthai
 module, 247
 depthAverage() (*depthai.SpatialLocations* property), 337
 depthAveragePixelCount() (*depthai.SpatialLocations* property), 337
 depthThresholds() (*depthai.SpatialDetectionNetworkProperties* property), 333
 depthThresholds() (*depthai.SpatialLocationCalculatorConfigData* property), 336
 desc() (*depthai.DeviceInfo* property), 283
 detectedBBScaleFactor() (*depthai.SpatialDetectionNetworkProperties* property), 333
 detectionLabelsToTrack() (*depthai.ObjectTrackerProperties* property), 308
 DetectionNetwork (class in *depthai*), 271
 DetectionNetworkProperties (class in *depthai*), 273
 detections() (*depthai.ImgDetections* property), 289
 detections() (*depthai.RawImgDetections* property), 323
 detections() (*depthai.RawSpatialImgDetections* property), 328
 detections() (*depthai.SpatialImgDetections* property), 334
 Device (class in *depthai*), 273
 DeviceBootloader (class in *depthai*), 280
 DeviceBootloader.Version (class in *depthai*), 281
 DeviceDesc (class in *depthai*), 282
 DeviceInfo (class in *depthai*), 283
 dims() (*depthai.TensorInfo* property), 346
 disparity() (*depthai.StereoDepth* property), 339
 dss() (*depthai.ChipTemperature* property), 261

E

EDOF (*depthai.CameraControl.AutoFocusMode* attribute), 254
 EDOF (*depthai.RawCameraControl.AutoFocusMode* attribute), 316
 empty() (*depthai.Rect* method), 330
 enableCenterCropRectangle() (*depthai.RawImageManipConfig.CropConfig* property), 321
 enableCrop() (*depthai.RawImageManipConfig* property), 323

enableExtendedDisparity() (*depthai.StereoDepthProperties* property), 342
 enableFormat() (*depthai.RawImageManipConfig* property), 323
 enableLeftRightCheck() (*depthai.StereoDepthProperties* property), 342
 enableOutputDepth() (*depthai.StereoDepthProperties* property), 342
 enableOutputRectified() (*depthai.StereoDepthProperties* property), 342
 enableResize() (*depthai.RawImageManipConfig* property), 323
 enableRotatedRect() (*depthai.RawImageManipConfig.CropConfig* property), 321
 enableRotation() (*depthai.RawImageManipConfig.ResizeConfig* property), 322
 enableSubpixel() (*depthai.StereoDepthProperties* property), 342
 enableWarp4pt() (*depthai.RawImageManipConfig.ResizeConfig* property), 322
 enableWarpMatrix() (*depthai.RawImageManipConfig.ResizeConfig* property), 322
 ERR (*depthai.LogLevel* attribute), 295

F

FACE_PRIORITY (*depthai.CameraControl.SceneMode* attribute), 257
 FACE_PRIORITY (*depthai.RawCameraControl.SceneMode* attribute), 319
 fb() (*depthai.RawImgFrame* property), 327
 FIREWORKS (*depthai.CameraControl.SceneMode* attribute), 257
 FIREWORKS (*depthai.RawCameraControl.SceneMode* attribute), 320
 flash() (*depthai.DeviceBootloader* method), 281
 flashBootloader() (*depthai.DeviceBootloader* method), 281
 flashDepthaiApplicationPackage() (*depthai.DeviceBootloader* method), 282
 flipHorizontal() (*depthai.RawImageManipConfig.FormatConfig* property), 322
 FLUORESCENT (*depthai.CameraControl.AutoWhiteBalanceMode* attribute), 255
 FLUORESCENT (*depthai.RawCameraControl.AutoWhiteBalanceMode* attribute), 317
 formatConfig() (*depthai.RawImageManipConfig* property), 323
 FP16 (*depthai.TensorInfo.DataType* attribute), 344

FP32 (*depthai.TensorInfo.DataType* attribute), 344
 fps () (*depthai.ColorCameraProperties* property), 268
 fps () (*depthai.MonoCameraProperties* property), 299

G

get () (*depthai.AssetManager* method), 250
 get () (*depthai.DataOutputQueue* method), 271
 getAll () (*depthai.AssetManager* method), 250
 getAll () (*depthai.DataOutputQueue* method), 271
 getAllAssets () (*depthai.Pipeline* method), 312
 getAllAvailableDevices () (*depthai.Device* static method), 275
 getAllAvailableDevices () (*depthai.DeviceBootloader* static method), 282
 getAllConnectedDevices () (*depthai.XLinkConnection* static method), 355
 getAllLayerNames () (*depthai.NNData* method), 300
 getAllLayers () (*depthai.NNData* method), 300
 getAllNodes () (*depthai.Pipeline* method), 312
 getAnyAvailableDevice () (*depthai.Device* static method), 275
 getAssetManager () (*depthai.Pipeline* method), 312
 getAssets () (*depthai.Node* method), 306
 getBitrate () (*depthai.VideoEncoder* method), 350
 getBitrateKbps () (*depthai.VideoEncoder* method), 351
 getBlocking () (*depthai.DataInputQueue* method), 269
 getBlocking () (*depthai.DataOutputQueue* method), 271
 getBlocking () (*depthai.Node.Input* method), 305
 getBoardSocket () (*depthai.ColorCamera* method), 263
 getBoardSocket () (*depthai.MonoCamera* method), 297
 getCamId () (*depthai.ColorCamera* method), 263
 getCamId () (*depthai.MonoCamera* method), 297
 getCaptureStill () (*depthai.CameraControl* method), 258
 getCategory () (*depthai.ImgFrame* method), 293
 getChipTemperature () (*depthai.Device* method), 276
 getCmxMemoryUsage () (*depthai.Device* method), 276
 getColorOrder () (*depthai.ColorCamera* method), 263
 getConfigData () (*depthai.SpatialLocationCalculator* method), 336
 getConnectionMap () (*depthai.Pipeline* method), 312
 getConnections () (*depthai.Node.Output* method), 305
 getConnections () (*depthai.Pipeline* method), 313
 getCropXMax () (*depthai.ImageManipConfig* method), 286
 getCropXMin () (*depthai.ImageManipConfig* method), 286
 getCropYMax () (*depthai.ImageManipConfig* method), 286
 getCropYMin () (*depthai.ImageManipConfig* method), 286
 getCvFrame () (*depthai.ImgFrame* method), 293
 getData () (*depthai.Buffer* method), 251
 getDdrMemoryUsage () (*depthai.Device* method), 276
 getDeviceByMxId () (*depthai.Device* static method), 276
 getDeviceByMxId () (*depthai.XLinkConnection* static method), 355
 getEmbeddedBootloaderBinary () (*depthai.DeviceBootloader* static method), 282
 getEmbeddedBootloaderVersion () (*depthai.DeviceBootloader* static method), 282
 getEmbeddedDeviceBinary () (*depthai.Device* static method), 276
 getFirstAvailableDevice () (*depthai.Device* static method), 276
 getFirstAvailableDevice () (*depthai.DeviceBootloader* static method), 282
 getFirstDevice () (*depthai.XLinkConnection* static method), 355
 getFirstLayerFp16 () (*depthai.NNData* method), 300
 getFirstLayerInt32 () (*depthai.NNData* method), 300
 getFirstLayerUInt8 () (*depthai.NNData* method), 300
 getFp16 () (*depthai.ColorCamera* method), 263
 getFps () (*depthai.ColorCamera* method), 264
 getFps () (*depthai.MonoCamera* method), 297
 getFpsLimit () (*depthai.XLinkOut* method), 357
 getFrame () (*depthai.ImgFrame* method), 293
 getFrameRate () (*depthai.VideoEncoder* method), 351
 getGlobalProperties () (*depthai.Pipeline* method), 313
 getHeight () (*depthai.ImgFrame* method), 293
 getHeight () (*depthai.VideoEncoder* method), 351
 getImageOrientation () (*depthai.ColorCamera* method), 264
 getImageOrientation () (*depthai.MonoCamera*

method), 297
 getInputQueue () (*depthai.Device method*), 276
 getInputQueueNames () (*depthai.Device method*), 277
 getInputs () (*depthai.Node method*), 306
 getInstanceNum () (*depthai.ImgFrame method*), 293
 getInterleaved () (*depthai.ColorCamera method*), 264
 getKeyframeFrequency () (*depthai.VideoEncoder method*), 351
 getLayer () (*depthai.NNData method*), 300
 getLayerDatatype () (*depthai.NNData method*), 300
 getLayerFp16 () (*depthai.NNData method*), 300
 getLayerInt32 () (*depthai.NNData method*), 300
 getLayerUInt8 () (*depthai.NNData method*), 300
 getLeonCssCpuUsage () (*depthai.Device method*), 277
 getLeonCssHeapUsage () (*depthai.Device method*), 277
 getLeonMssCpuUsage () (*depthai.Device method*), 277
 getLeonMssHeapUsage () (*depthai.Device method*), 277
 getLogLevel () (*depthai.Device method*), 277
 getLogOutputLevel () (*depthai.Device method*), 277
 getLossless () (*depthai.VideoEncoder method*), 351
 getMaxDataSize () (*depthai.XLinkIn method*), 356
 getMaxSize () (*depthai.DataInputQueue method*), 269
 getMaxSize () (*depthai.DataOutputQueue method*), 271
 getMetadataOnly () (*depthai.XLinkOut method*), 357
 getMxId () (*depthai.DeviceInfo method*), 283
 getName () (*depthai.DataInputQueue method*), 269
 getName () (*depthai.DataOutputQueue method*), 271
 getName () (*depthai.Node method*), 306
 getNode () (*depthai.Pipeline method*), 313
 getNodeMap () (*depthai.Pipeline method*), 313
 getNumBframes () (*depthai.VideoEncoder method*), 351
 getNumFrames () (*depthai.XLinkIn method*), 356
 getNumFramesPool () (*depthai.VideoEncoder method*), 351
 getNumInferenceThreads () (*depthai.NeuralNetwork method*), 302
 getOutputQueue () (*depthai.Device method*), 277
 getOutputQueueNames () (*depthai.Device method*), 278
 getOutputs () (*depthai.Node method*), 306
 getPreviewHeight () (*depthai.ColorCamera method*), 264
 getPreviewKeepAspectRatio () (*depthai.ColorCamera method*), 264
 getPreviewSize () (*depthai.ColorCamera method*), 264
 getPreviewWidth () (*depthai.ColorCamera method*), 264
 getProfile () (*depthai.VideoEncoder method*), 351
 getQuality () (*depthai.VideoEncoder method*), 351
 getQueueEvent () (*depthai.Device method*), 278
 getQueueEvents () (*depthai.Device method*), 278
 getQueueSize () (*depthai.Node.Input method*), 305
 getRateControlMode () (*depthai.VideoEncoder method*), 351
 getRaw () (*depthai.ADatatype method*), 249
 getResizeHeight () (*depthai.ImageManipConfig method*), 286
 getResizeWidth () (*depthai.ImageManipConfig method*), 286
 getResolution () (*depthai.ColorCamera method*), 264
 getResolution () (*depthai.MonoCamera method*), 297
 getResolutionHeight () (*depthai.ColorCamera method*), 264
 getResolutionHeight () (*depthai.MonoCamera method*), 297
 getResolutionSize () (*depthai.ColorCamera method*), 264
 getResolutionSize () (*depthai.MonoCamera method*), 297
 getResolutionWidth () (*depthai.ColorCamera method*), 264
 getResolutionWidth () (*depthai.MonoCamera method*), 297
 getSensorCrop () (*depthai.ColorCamera method*), 264
 getSensorCropX () (*depthai.ColorCamera method*), 264
 getSensorCropY () (*depthai.ColorCamera method*), 264
 getSequenceNum () (*depthai.ImgFrame method*), 293
 getSize () (*depthai.VideoEncoder method*), 351
 getSpatialLocations () (*depthai.SpatialLocationCalculatorData method*), 337
 getStillHeight () (*depthai.ColorCamera method*), 264
 getStillSize () (*depthai.ColorCamera method*), 264
 getStillWidth () (*depthai.ColorCamera method*), 264
 getStreamName () (*depthai.XLinkIn method*), 356
 getStreamName () (*depthai.XLinkOut method*), 357
 getSystemInformationLoggingRate ()

(*depthai.Device* method), 279

getTimestamp() (*depthai.ImgFrame* method), 293

getType() (*depthai.ImgFrame* method), 293

getVersion() (*depthai.DeviceBootloader* method), 282

getVersionName() (*depthai.OpenVINO* static method), 310

getVersions() (*depthai.OpenVINO* static method), 311

getVideoHeight() (*depthai.ColorCamera* method), 264

getVideoSize() (*depthai.ColorCamera* method), 265

getVideoWidth() (*depthai.ColorCamera* method), 265

getWaitForConfigInput() (*depthai.ColorCamera* method), 265

getWidth() (*depthai.ImgFrame* method), 294

getWidth() (*depthai.VideoEncoder* method), 351

GlobalProperties (class in *depthai*), 283

GRAY8 (*depthai.ImgFrame.Type* attribute), 292

GRAY8 (*depthai.RawImgFrame.Type* attribute), 326

GRAYF16 (*depthai.ImgFrame.Type* attribute), 292

GRAYF16 (*depthai.RawImgFrame.Type* attribute), 326

H

H (*depthai.TensorInfo.StorageOrder* attribute), 346

H264_BASELINE (*depthai.VideoEncoderProperties.Profile* attribute), 353

H264_HIGH (*depthai.VideoEncoderProperties.Profile* attribute), 353

H264_MAIN (*depthai.VideoEncoderProperties.Profile* attribute), 354

H265_MAIN (*depthai.VideoEncoderProperties.Profile* attribute), 354

has() (*depthai.DataOutputQueue* method), 271

hasLayer() (*depthai.NNData* method), 301

HCW (*depthai.TensorInfo.StorageOrder* attribute), 346

HDR (*depthai.ImgFrame.Type* attribute), 292

HDR (*depthai.RawImgFrame.Type* attribute), 326

height() (*depthai.ImgFrame.Specs* property), 290

height() (*depthai.RawImageManipConfig.ResizeConfig* property), 323

height() (*depthai.RawImgFrame.Specs* property), 324

height() (*depthai.Rect* property), 330

height() (*depthai.Size2f* property), 331

height() (*depthai.StereoDepthProperties* property), 342

height() (*depthai.VideoEncoderProperties* property), 354

HORIZONTAL_MIRROR (*depthai.CameraImageOrientation* attribute), 261

HWC (*depthai.TensorInfo.StorageOrder* attribute), 346

I

I8 (*depthai.TensorInfo.DataType* attribute), 344

id() (*depthai.Node* property), 306

id() (*depthai.Tracklet* property), 348

ImageManip (class in *depthai*), 284

ImageManipConfig (class in *depthai*), 285

ImgDetection (class in *depthai*), 288

ImgDetections (class in *depthai*), 289

ImgFrame (class in *depthai*), 289

ImgFrame.Specs (class in *depthai*), 290

ImgFrame.Type (class in *depthai*), 290

INCANDESCENT (*depthai.CameraControl.AutoWhiteBalanceMode* attribute), 255

INCANDESCENT (*depthai.RawCameraControl.AutoWhiteBalanceMode* attribute), 317

INFO (*depthai.LogLevel* attribute), 295

initialConfig() (*depthai.ImageManip* property), 284

initialConfig() (*depthai.SpatialLocationCalculator* property), 335

initialControl() (*depthai.ColorCamera* property), 265

initialControl() (*depthai.ColorCameraProperties* property), 268

initialControl() (*depthai.MonoCamera* property), 297

initialControl() (*depthai.MonoCameraProperties* property), 299

input() (*depthai.DetectionNetwork* property), 272

input() (*depthai.NeuralNetwork* property), 302

input() (*depthai.SpatialDetectionNetwork* property), 332

input() (*depthai.SPIOut* property), 331

input() (*depthai.VideoEncoder* property), 351

input() (*depthai.XLinkOut* property), 357

inputConfig() (*depthai.ColorCamera* property), 265

inputConfig() (*depthai.ImageManip* property), 284

inputConfig() (*depthai.SpatialLocationCalculator* property), 335

inputConfigSync() (*depthai.SpatialLocationCalculatorProperties* property), 337

inputControl() (*depthai.ColorCamera* property), 265

inputControl() (*depthai.MonoCamera* property), 297

inputDepth() (*depthai.SpatialDetectionNetwork* property), 332

inputDepth() (*depthai.SpatialLocationCalculator* property), 335

inputDetectionFrame() (*depthai.ObjectTracker* property), 307

() (*depthai.ObjectTracker* property), 307
() (*depthai.Node.Connection* property), 304
() (*depthai.ImageManip* property), 284
() (*depthai.Node.Connection* property), 304
depthai.ObjectTracker property), 307
 instanceNum() (*depthai.RawImgFrame* property), 327
 INT (*depthai.TensorInfo.DataType* attribute), 344
 interleaved() (*depthai.ColorCameraProperties* property), 268
 iouThreshold() (*depthai.DetectionNetworkProperties* property), 273
 isEmbeddedVersion() (*depthai.DeviceBootloader* method), 282
 isNormalized() (*depthai.Rect* method), 330
 isPipelineRunning() (*depthai.Device* method), 279
 isResizeThumbnail() (*depthai.ImageManipConfig* method), 286

K

keepAspectRatio() (*depthai.RawImageManipConfig.ResizeConfig* property), 323
 KERNEL_3x3 (*depthai.StereoDepthProperties.MedianFilter* attribute), 341
 KERNEL_5x5 (*depthai.StereoDepthProperties.MedianFilter* attribute), 341
 KERNEL_7x7 (*depthai.StereoDepthProperties.MedianFilter* attribute), 341
 key() (*depthai.Asset* property), 249
 keyframeFrequency() (*depthai.VideoEncoderProperties* property), 354

L

label() (*depthai.ImgDetection* property), 288
 label() (*depthai.Tracklet* property), 348
 LANDSCAPE (*depthai.CameraControl.SceneMode* attribute), 258
 LANDSCAPE (*depthai.RawCameraControl.SceneMode* attribute), 320
 LEFT (*depthai.CameraBoardSocket* attribute), 252
 left() (*depthai.StereoDepth* property), 339
 lensPosition() (*depthai.RawCameraControl* property), 320
 leonCssCpuUsage() (*depthai.RawSystemInformation* property), 328
 leonCssCpuUsage() (*depthai.SystemInformation* property), 342

leonCssMemoryUsage() (*depthai.RawSystemInformation* property), 328
 leonCssMemoryUsage() (*depthai.SystemInformation* property), 342
 leonMssCpuUsage() (*depthai.RawSystemInformation* property), 328
 leonMssCpuUsage() (*depthai.SystemInformation* property), 342
 leonMssMemoryUsage() (*depthai.RawSystemInformation* property), 328
 leonMssMemoryUsage() (*depthai.SystemInformation* property), 343
 leonOsFrequencyHz() (*depthai.GlobalProperties* property), 283
 leonRtFrequencyHz() (*depthai.GlobalProperties* property), 283
 link() (*depthai.Node.Output* method), 305
 link() (*depthai.Pipeline* method), 313
 loadCalibrationData() (*depthai.StereoDepth* method), 339
 loadCalibrationFile() (*depthai.StereoDepth* method), 339
 lockAspectRatioFill() (*depthai.RawImageManipConfig.ResizeConfig* property), 323
 LogLevel (class in *depthai*), 294
 LOST (*depthai.Tracklet.TrackingStatus* attribute), 348
 lowerThreshold() (*depthai.SpatialLocationCalculatorConfigThreshold* property), 336
 LUT16 (*depthai.ImgFrame.Type* attribute), 292
 LUT16 (*depthai.RawImgFrame.Type* attribute), 326
 LUT2 (*depthai.ImgFrame.Type* attribute), 292
 LUT2 (*depthai.RawImgFrame.Type* attribute), 326
 LUT4 (*depthai.ImgFrame.Type* attribute), 292
 LUT4 (*depthai.RawImgFrame.Type* attribute), 326

M

MACRO (*depthai.CameraControl.AutoFocusMode* attribute), 254
 MACRO (*depthai.RawCameraControl.AutoFocusMode* attribute), 316
 MAINS_50_HZ (*depthai.CameraControl.AntiBandingMode* attribute), 253
 MAINS_50_HZ (*depthai.RawCameraControl.AntiBandingMode* attribute), 315
 MAINS_60_HZ (*depthai.CameraControl.AntiBandingMode* attribute), 253
 MAINS_60_HZ (*depthai.RawCameraControl.AntiBandingMode* attribute), 315
 maxBitrate() (*depthai.VideoEncoderProperties* property), 354

maxObjectsToTrack() (*depthai.ObjectTrackerProperties* property), 308
 median() (*depthai.StereoDepthProperties* property), 342
 MEDIAN_OFF (*depthai.StereoDepthProperties.MedianFilter* attribute), 341
 MemoryInfo (class in *depthai*), 295
 MJPEG (*depthai.VideoEncoderProperties.Profile* attribute), 354
 MobileNetDetectionNetwork (class in *depthai*), 295
 MobileNetSpatialDetectionNetwork (class in *depthai*), 296
 module
 depthai, 247
 MONO (*depthai.CameraControl.EffectMode* attribute), 256
 MONO (*depthai.RawCameraControl.EffectMode* attribute), 318
 MonoCamera (class in *depthai*), 296
 MonoCameraProperties (class in *depthai*), 298
 MonoCameraProperties.SensorResolution (class in *depthai*), 298
 mss() (*depthai.ChipTemperature* property), 261
 msTime() (*depthai.CpuUsage* property), 269

N

name() (*depthai.CameraBoardSocket* property), 252
 name() (*depthai.CameraControl.AntiBandingMode* property), 253
 name() (*depthai.CameraControl.AutoFocusMode* property), 254
 name() (*depthai.CameraControl.AutoWhiteBalanceMode* property), 255
 name() (*depthai.CameraControl.EffectMode* property), 256
 name() (*depthai.CameraControl.SceneMode* property), 258
 name() (*depthai.CameraImageOrientation* property), 261
 name() (*depthai.ColorCameraProperties.ColorOrder* property), 267
 name() (*depthai.ColorCameraProperties.SensorResolution* property), 268
 name() (*depthai.DeviceDesc* property), 283
 name() (*depthai.ImgFrame.Type* property), 293
 name() (*depthai.LogLevel* property), 295
 name() (*depthai.MonoCameraProperties.SensorResolution* property), 299
 name() (*depthai.OpenVINO.Version* property), 310
 name() (*depthai.RawCameraControl.AntiBandingMode* property), 315
 name() (*depthai.RawCameraControl.AutoFocusMode* property), 316
 name() (*depthai.RawCameraControl.AutoWhiteBalanceMode* property), 317
 name() (*depthai.RawCameraControl.EffectMode* property), 318
 name() (*depthai.RawCameraControl.SceneMode* property), 320
 name() (*depthai.RawImgFrame.Type* property), 327
 name() (*depthai.StereoDepthProperties.MedianFilter* property), 342
 name() (*depthai.TensorInfo* property), 346
 name() (*depthai.TensorInfo.DataType* property), 344
 name() (*depthai.TensorInfo.StorageOrder* property), 346
 name() (*depthai.TrackerIdAssignmentPolicy* property), 347
 name() (*depthai.TrackerType* property), 347
 name() (*depthai.Tracklet.TrackingStatus* property), 348
 name() (*depthai.VideoEncoderProperties.Profile* property), 354
 name() (*depthai.VideoEncoderProperties.RateControlMode* property), 354
 name() (*depthai.XLinkDeviceState* property), 356
 name() (*depthai.XLinkPlatform* property), 358
 name() (*depthai.XLinkProtocol* property), 359
 NC (*depthai.TensorInfo.StorageOrder* attribute), 346
 NCHW (*depthai.TensorInfo.StorageOrder* attribute), 346
 NEGATIVE (*depthai.CameraControl.EffectMode* attribute), 256
 NEGATIVE (*depthai.RawCameraControl.EffectMode* attribute), 318
 NeuralNetwork (class in *depthai*), 301
 NeuralNetworkProperties (class in *depthai*), 303
 NEW (*depthai.Tracklet.TrackingStatus* attribute), 348
 NHCW (*depthai.TensorInfo.StorageOrder* attribute), 346
 NHWC (*depthai.TensorInfo.StorageOrder* attribute), 346
 NIGHT (*depthai.CameraControl.SceneMode* attribute), 258
 NIGHT (*depthai.RawCameraControl.SceneMode* attribute), 320
 NIGHT_PORTRAIT (*depthai.CameraControl.SceneMode* attribute), 258
 NIGHT_PORTRAIT (*depthai.RawCameraControl.SceneMode* attribute), 320
 NNData (class in *depthai*), 299
 nnFamily() (*depthai.DetectionNetworkProperties* property), 273
 Node (class in *depthai*), 303
 Node.Connection (class in *depthai*), 304
 Node.Id (class in *depthai*), 304
 Node.Input (class in *depthai*), 304
 Node.Output (class in *depthai*), 305
 NONE (*depthai.ImgFrame.Type* attribute), 292

NONE (*depthai.RawImgFrame.Type* attribute), 326
 NORMAL (*depthai.CameraImageOrientation* attribute), 261
 normalize() (*depthai.Rect* method), 330
 normalizedCoords() (*depthai.RawImageManipConfig.CropConfig* property), 321
 normalizedCoords() (*depthai.RawImageManipConfig.ResizeConfig* property), 323
 nsec() (*depthai.Timestamp* property), 346
 numBFrames() (*depthai.VideoEncoderProperties* property), 354
 numDimensions() (*depthai.TensorInfo* property), 346
 numFrames() (*depthai.NeuralNetworkProperties* property), 303
 numFramesPool() (*depthai.VideoEncoderProperties* property), 354
 numNCEPerThread() (*depthai.NeuralNetworkProperties* property), 303
 numThreads() (*depthai.NeuralNetworkProperties* property), 303
 NV12 (*depthai.ImgFrame.Type* attribute), 292
 NV12 (*depthai.RawImgFrame.Type* attribute), 326
 NV21 (*depthai.ImgFrame.Type* attribute), 292
 NV21 (*depthai.RawImgFrame.Type* attribute), 326

O

ObjectTracker (class in *depthai*), 306
 ObjectTrackerProperties (class in *depthai*), 308
 OFF (*depthai.CameraControl.AntiBandingMode* attribute), 253
 OFF (*depthai.CameraControl.AutoFocusMode* attribute), 254
 OFF (*depthai.CameraControl.AutoWhiteBalanceMode* attribute), 255
 OFF (*depthai.CameraControl.EffectMode* attribute), 256
 OFF (*depthai.LogLevel* attribute), 295
 OFF (*depthai.RawCameraControl.AntiBandingMode* attribute), 315
 OFF (*depthai.RawCameraControl.AutoFocusMode* attribute), 316
 OFF (*depthai.RawCameraControl.AutoWhiteBalanceMode* attribute), 317
 OFF (*depthai.RawCameraControl.EffectMode* attribute), 318
 offset() (*depthai.TensorInfo* property), 346
 OpenVINO (class in *depthai*), 308
 OpenVINO.Version (class in *depthai*), 309
 order() (*depthai.TensorInfo* property), 346
 out() (*depthai.DetectionNetwork* property), 272
 out() (*depthai.ImageManip* property), 284
 out() (*depthai.MonoCamera* property), 297

out() (*depthai.NeuralNetwork* property), 302
 out() (*depthai.ObjectTracker* property), 307
 out() (*depthai.SpatialDetectionNetwork* property), 333
 out() (*depthai.SpatialLocationCalculator* property), 335
 out() (*depthai.SystemLogger* property), 343
 out() (*depthai.XLinkIn* property), 356
 outputId() (*depthai.Node.Connection* property), 304
 outputName() (*depthai.Node.Connection* property), 304

P

p1Offset() (*depthai.ImgFrame.Specs* property), 290
 p1Offset() (*depthai.RawImgFrame.Specs* property), 324
 p2Offset() (*depthai.ImgFrame.Specs* property), 290
 p2Offset() (*depthai.RawImgFrame.Specs* property), 324
 p3Offset() (*depthai.ImgFrame.Specs* property), 290
 p3Offset() (*depthai.RawImgFrame.Specs* property), 324
 PACK10 (*depthai.ImgFrame.Type* attribute), 293
 PACK10 (*depthai.RawImgFrame.Type* attribute), 326
 PACK12 (*depthai.ImgFrame.Type* attribute), 293
 PACK12 (*depthai.RawImgFrame.Type* attribute), 327
 parseVersionName() (*depthai.OpenVINO* static method), 311
 PARTY (*depthai.CameraControl.SceneMode* attribute), 258
 PARTY (*depthai.RawCameraControl.SceneMode* attribute), 320
 passthrough() (*depthai.DetectionNetwork* property), 272
 passthrough() (*depthai.NeuralNetwork* property), 302
 passthrough() (*depthai.SpatialDetectionNetwork* property), 333
 passthroughDepth() (*depthai.SpatialDetectionNetwork* property), 333
 passthroughDepth() (*depthai.SpatialLocationCalculator* property), 335
 passthroughDetectionFrame() (*depthai.ObjectTracker* property), 307
 passthroughDetections() (*depthai.ObjectTracker* property), 307
 passthroughTrackerFrame() (*depthai.ObjectTracker* property), 307
 Pipeline (class in *depthai*), 311
 pipelineName() (*depthai.GlobalProperties* property), 283
 pipelineVersion() (*depthai.GlobalProperties* property), 283

platform() (*depthai.DeviceDesc* property), 283
 Point2f (*class in depthai*), 313
 Point3f (*class in depthai*), 314
 PORTRAIT (*depthai.CameraControl.SceneMode* attribute), 258
 PORTRAIT (*depthai.RawCameraControl.SceneMode* attribute), 320
 POSTERIZE (*depthai.CameraControl.EffectMode* attribute), 256
 POSTERIZE (*depthai.RawCameraControl.EffectMode* attribute), 318
 preview() (*depthai.ColorCamera* property), 265
 previewHeight() (*depthai.ColorCameraProperties* property), 268
 previewWidth() (*depthai.ColorCameraProperties* property), 268
 profile() (*depthai.VideoEncoderProperties* property), 354
 Properties (*depthai.ColorCamera* attribute), 263
 Properties (*depthai.DetectionNetwork* attribute), 272
 Properties (*depthai.MonoCamera* attribute), 296
 Properties (*depthai.NeuralNetwork* attribute), 302
 Properties (*depthai.ObjectTracker* attribute), 307
 Properties (*depthai.SpatialDetectionNetwork* attribute), 332
 Properties (*depthai.SpatialLocationCalculator* attribute), 335
 Properties (*depthai.StereoDepth* attribute), 338
 Properties (*depthai.VideoEncoder* attribute), 350
 protocol() (*depthai.DeviceDesc* property), 283

Q

quality() (*depthai.VideoEncoderProperties* property), 354

R

rateCtrlMode() (*depthai.VideoEncoderProperties* property), 354
 rateHz() (*depthai.SystemLoggerProperties* property), 343
 RAW10 (*depthai.ImgFrame.Type* attribute), 293
 RAW10 (*depthai.RawImgFrame.Type* attribute), 327
 RAW12 (*depthai.ImgFrame.Type* attribute), 293
 RAW12 (*depthai.RawImgFrame.Type* attribute), 327
 RAW14 (*depthai.ImgFrame.Type* attribute), 293
 RAW14 (*depthai.RawImgFrame.Type* attribute), 327
 RAW16 (*depthai.ImgFrame.Type* attribute), 293
 RAW16 (*depthai.RawImgFrame.Type* attribute), 327
 RAW8 (*depthai.ImgFrame.Type* attribute), 293
 RAW8 (*depthai.RawImgFrame.Type* attribute), 327
 RawBuffer (*class in depthai*), 314
 RawCameraControl (*class in depthai*), 314
 RawCameraControl.AntiBandingMode (*class in depthai*), 315
 RawCameraControl.AutoFocusMode (*class in depthai*), 315
 RawCameraControl.AutoWhiteBalanceMode (*class in depthai*), 316
 RawCameraControl.EffectMode (*class in depthai*), 317
 RawCameraControl.SceneMode (*class in depthai*), 318
 RawImageManipConfig (*class in depthai*), 320
 RawImageManipConfig.CropConfig (*class in depthai*), 321
 RawImageManipConfig.CropRect (*class in depthai*), 321
 RawImageManipConfig.FormatConfig (*class in depthai*), 322
 RawImageManipConfig.ResizeConfig (*class in depthai*), 322
 RawImgDetections (*class in depthai*), 323
 RawImgFrame (*class in depthai*), 323
 RawImgFrame.Specs (*class in depthai*), 324
 RawImgFrame.Type (*class in depthai*), 324
 RawNNData (*class in depthai*), 327
 RawSpatialImgDetections (*class in depthai*), 328
 RawSystemInformation (*class in depthai*), 328
 RawTracklets (*class in depthai*), 328
 Rect (*class in depthai*), 329
 rectifiedLeft() (*depthai.StereoDepth* property), 339
 rectifiedRight() (*depthai.StereoDepth* property), 339
 rectifyEdgeFillColor() (*depthai.StereoDepthProperties* property), 342
 rectifyMirrorFrame() (*depthai.StereoDepthProperties* property), 342
 remaining() (*depthai.MemoryInfo* property), 295
 remove() (*depthai.AssetManager* method), 250
 remove() (*depthai.Pipeline* method), 313
 removeCallback() (*depthai.DataOutputQueue* method), 271
 removeLogCallback() (*depthai.Device* method), 279
 resizeConfig() (*depthai.RawImageManipConfig* property), 323
 resolution() (*depthai.ColorCameraProperties* property), 268
 resolution() (*depthai.MonoCameraProperties* property), 299
 RGB (*depthai.CameraBoardSocket* attribute), 252
 RGB (*depthai.ColorCameraProperties.ColorOrder* attribute), 267
 RGB161616 (*depthai.ImgFrame.Type* attribute), 293
 RGB161616 (*depthai.RawImgFrame.Type* attribute),

327
 RGB888i (*depthai.ImgFrame.Type* attribute), 293
 RGB888i (*depthai.RawImgFrame.Type* attribute), 327
 RGB888p (*depthai.ImgFrame.Type* attribute), 293
 RGB888p (*depthai.RawImgFrame.Type* attribute), 327
 RGBA8888 (*depthai.ImgFrame.Type* attribute), 293
 RGBA8888 (*depthai.RawImgFrame.Type* attribute), 327
 RGBF16F16F16i (*depthai.ImgFrame.Type* attribute), 293
 RGBF16F16F16i (*depthai.RawImgFrame.Type* attribute), 327
 RGBF16F16F16p (*depthai.ImgFrame.Type* attribute), 293
 RGBF16F16F16p (*depthai.RawImgFrame.Type* attribute), 327
 RIGHT (*depthai.CameraBoardSocket* attribute), 252
 right () (*depthai.StereoDepth* property), 339
 roi () (*depthai.SpatialLocationCalculatorConfigData* property), 336
 roi () (*depthai.Tracklet* property), 349
 roiConfig () (*depthai.SpatialLocationCalculatorProperties* property), 337
 ROTATE_180_DEG (*depthai.CameraImageOrientation* attribute), 261
 RotatedRect (class in *depthai*), 330
 rotationAngleDeg () (*depthai.RawImageManipConfig.ResizeConfig* property), 323

S

saveDepthaiApplicationPackage () (*depthai.DeviceBootloader* static method), 282
 sec () (*depthai.Timestamp* property), 346
 send () (*depthai.DataInputQueue* method), 269
 sensorCenterCrop () (*depthai.ColorCamera* method), 265
 sensorCropX () (*depthai.ColorCameraProperties* property), 268
 sensorCropY () (*depthai.ColorCameraProperties* property), 268
 SEPIA (*depthai.CameraControl.EffectMode* attribute), 256
 SEPIA (*depthai.RawCameraControl.EffectMode* attribute), 318
 sequenceNum () (*depthai.RawImgFrame* property), 327
 set () (*depthai.AssetManager* method), 250
 setAnchorMasks () (*depthai.YoloDetectionNetwork* method), 359
 setAnchorMasks () (*depthai.YoloSpatialDetectionNetwork* method), 360
 setAnchors () (*depthai.YoloDetectionNetwork* method), 359

setAnchors () (*depthai.YoloSpatialDetectionNetwork* method), 360
 setAntiBandingMode () (*depthai.CameraControl* method), 258
 setAutoExposureCompensation () (*depthai.CameraControl* method), 258
 setAutoExposureEnable () (*depthai.CameraControl* method), 258
 setAutoExposureLock () (*depthai.CameraControl* method), 258
 setAutoExposureRegion () (*depthai.CameraControl* method), 258
 setAutoFocusMode () (*depthai.CameraControl* method), 258
 setAutoFocusRegion () (*depthai.CameraControl* method), 259
 setAutoFocusTrigger () (*depthai.CameraControl* method), 259
 setAutoWhiteBalanceLock () (*depthai.CameraControl* method), 259
 setAutoWhiteBalanceMode () (*depthai.CameraControl* method), 259
 setBitrate () (*depthai.VideoEncoder* method), 351
 setBitrateKbps () (*depthai.VideoEncoder* method), 351
 setBlobPath () (*depthai.NeuralNetwork* method), 302
 setBlocking () (*depthai.DataInputQueue* method), 269
 setBlocking () (*depthai.DataOutputQueue* method), 271
 setBlocking () (*depthai.Node.Input* method), 305
 setBoardSocket () (*depthai.ColorCamera* method), 265
 setBoardSocket () (*depthai.MonoCamera* method), 297
 setBoundingBoxScaleFactor () (*depthai.SpatialDetectionNetwork* method), 333
 setBrightness () (*depthai.CameraControl* method), 259
 setBusId () (*depthai.SPIOut* method), 331
 setCamId () (*depthai.ColorCamera* method), 265
 setCamId () (*depthai.MonoCamera* method), 298
 setCaptureStill () (*depthai.CameraControl* method), 259
 setCategory () (*depthai.ImgFrame* method), 294
 setCenterCrop () (*depthai.ImageManip* method), 284
 setCenterCrop () (*depthai.ImageManipConfig* method), 286
 setChromaDenoise () (*depthai.CameraControl* method), 259
 setColorOrder () (*depthai.ColorCamera* method),

[265](#)
 setConfidenceThreshold() (*depthai.DetectionNetwork* method), [272](#)
 setConfidenceThreshold() (*depthai.StereoDepth* method), [339](#)
 setContrast() (*depthai.CameraControl* method), [259](#)
 setCoordinateSize() (*depthai.YoloDetectionNetwork* method), [359](#)
 setCoordinateSize() (*depthai.YoloSpatialDetectionNetwork* method), [360](#)
 setCropRect() (*depthai.ImageManip* method), [284](#)
 setCropRotatedRect() (*depthai.ImageManipConfig* method), [287](#)
 setData() (*depthai.Buffer* method), [251](#)
 setDefaultProfilePreset() (*depthai.VideoEncoder* method), [351](#)
 setDepthLowerThreshold() (*depthai.SpatialDetectionNetwork* method), [333](#)
 setDepthUpperThreshold() (*depthai.SpatialDetectionNetwork* method), [333](#)
 setDetectionLabelsToTrack() (*depthai.ObjectTracker* method), [307](#)
 setEffectMode() (*depthai.CameraControl* method), [259](#)
 setEmptyCalibration() (*depthai.StereoDepth* method), [340](#)
 setExtendedDisparity() (*depthai.StereoDepth* method), [340](#)
 setFp16() (*depthai.ColorCamera* method), [265](#)
 setFps() (*depthai.ColorCamera* method), [265](#)
 setFps() (*depthai.MonoCamera* method), [298](#)
 setFpsLimit() (*depthai.XLinkOut* method), [357](#)
 setFrame() (*depthai.ImgFrame* method), [294](#)
 setFrameRate() (*depthai.VideoEncoder* method), [352](#)
 setFrameType() (*depthai.ImageManip* method), [284](#)
 setFrameType() (*depthai.ImageManipConfig* method), [287](#)
 setHeight() (*depthai.ImgFrame* method), [294](#)
 setHorizontalFlip() (*depthai.ImageManip* method), [284](#)
 setHorizontalFlip() (*depthai.ImageManipConfig* method), [287](#)
 setImageOrientation() (*depthai.ColorCamera* method), [265](#)
 setImageOrientation() (*depthai.MonoCamera* method), [298](#)
 setInputResolution() (*depthai.StereoDepth* method), [340](#)
 setInstanceNum() (*depthai.ImgFrame* method), [294](#)
 setInterleaved() (*depthai.ColorCamera* method), [265](#)
 setIouThreshold() (*depthai.YoloDetectionNetwork* method), [359](#)
 setIouThreshold() (*depthai.YoloSpatialDetectionNetwork* method), [360](#)
 setKeepAspectRatio() (*depthai.ImageManip* method), [284](#)
 setKeepAspectRatio() (*depthai.ImageManipConfig* method), [287](#)
 setKeyframeFrequency() (*depthai.VideoEncoder* method), [352](#)
 setLayer() (*depthai.NNData* method), [301](#)
 setLeftRightCheck() (*depthai.StereoDepth* method), [340](#)
 setLogLevel() (*depthai.Device* method), [279](#)
 setLogOutputLevel() (*depthai.Device* method), [280](#)
 setLossless() (*depthai.VideoEncoder* method), [352](#)
 setLumaDenoise() (*depthai.CameraControl* method), [259](#)
 setManualExposure() (*depthai.CameraControl* method), [259](#)
 setManualFocus() (*depthai.CameraControl* method), [259](#)
 setMaxDataSize() (*depthai.XLinkIn* method), [356](#)
 setMaxObjectsToTrack() (*depthai.ObjectTracker* method), [307](#)
 setMaxOutputFrameSize() (*depthai.ImageManip* method), [285](#)
 setMaxSize() (*depthai.DataInputQueue* method), [270](#)
 setMaxSize() (*depthai.DataOutputQueue* method), [271](#)
 setMedianFilter() (*depthai.StereoDepth* method), [340](#)
 setMetadataOnly() (*depthai.XLinkOut* method), [357](#)
 setNoiseReductionStrength() (*depthai.CameraControl* method), [260](#)
 setNumBframes() (*depthai.VideoEncoder* method), [352](#)
 setNumClasses() (*depthai.YoloDetectionNetwork* method), [359](#)
 setNumClasses() (*depthai.YoloSpatialDetectionNetwork* method), [360](#)
 setNumFrames() (*depthai.XLinkIn* method), [356](#)
 setNumFramesPool() (*depthai.ImageManip* method), [285](#)
 setNumFramesPool() (*depthai.VideoEncoder* method), [352](#)

setNumInferenceThreads() (*depthai.NeuralNetwork method*), 303
 setNumNCEPerInferenceThread() (*depthai.NeuralNetwork method*), 303
 setNumPoolFrames() (*depthai.NeuralNetwork method*), 303
 setOpenVINVersion() (*depthai.Pipeline method*), 313
 setOutputDepth() (*depthai.StereoDepth method*), 340
 setOutputRectified() (*depthai.StereoDepth method*), 340
 setPreviewKeepAspectRatio() (*depthai.ColorCamera method*), 265
 setPreviewSize() (*depthai.ColorCamera method*), 266
 setProfile() (*depthai.VideoEncoder method*), 352
 setQuality() (*depthai.VideoEncoder method*), 352
 setQueueSize() (*depthai.Node.Input method*), 305
 setRate() (*depthai.SystemLogger method*), 343
 setRateControlMode() (*depthai.VideoEncoder method*), 352
 setRectifyEdgeFillColor() (*depthai.StereoDepth method*), 340
 setRectifyMirrorFrame() (*depthai.StereoDepth method*), 340
 setResize() (*depthai.ImageManip method*), 285
 setResize() (*depthai.ImageManipConfig method*), 287
 setResizeThumbnail() (*depthai.ImageManip method*), 285
 setResizeThumbnail() (*depthai.ImageManipConfig method*), 287
 setResolution() (*depthai.ColorCamera method*), 266
 setResolution() (*depthai.MonoCamera method*), 298
 setReusePreviousImage() (*depthai.ImageManipConfig method*), 287
 setROIs() (*depthai.SpatialLocationCalculatorConfig method*), 336
 setRotationDegrees() (*depthai.ImageManipConfig method*), 287
 setRotationRadians() (*depthai.ImageManipConfig method*), 287
 setSaturation() (*depthai.CameraControl method*), 260
 setSceneMode() (*depthai.CameraControl method*), 260
 setSensorCrop() (*depthai.ColorCamera method*), 266
 setSequenceNum() (*depthai.ImgFrame method*), 294
 setSharpness() (*depthai.CameraControl method*), 260
 setSkipCurrentImage() (*depthai.ImageManipConfig method*), 288
 setStartStreaming() (*depthai.CameraControl method*), 260
 setStillSize() (*depthai.ColorCamera method*), 266
 setStopStreaming() (*depthai.CameraControl method*), 260
 setStreamName() (*depthai.SPIOut method*), 331
 setStreamName() (*depthai.XLinkIn method*), 356
 setStreamName() (*depthai.XLinkOut method*), 357
 setSubpixel() (*depthai.StereoDepth method*), 340
 setSystemInformationLoggingRate() (*depthai.Device method*), 280
 setTimestamp() (*depthai.ImgFrame method*), 294
 setTrackerIdAssignmentPolicy() (*depthai.ObjectTracker method*), 307
 setTrackerThreshold() (*depthai.ObjectTracker method*), 308
 setTrackerType() (*depthai.ObjectTracker method*), 308
 setType() (*depthai.ImgFrame method*), 294
 setVideoSize() (*depthai.ColorCamera method*), 266
 setWaitForConfigInput() (*depthai.ColorCamera method*), 266
 setWaitForConfigInput() (*depthai.ImageManip method*), 285
 setWaitForConfigInput() (*depthai.SpatialLocationCalculator method*), 335
 setWarpBorderFillColor() (*depthai.ImageManipConfig method*), 288
 setWarpBorderReplicatePixels() (*depthai.ImageManipConfig method*), 288
 setWarpTransformFourPoints() (*depthai.ImageManipConfig method*), 288
 setWarpTransformMatrix3x3() (*depthai.ImageManipConfig method*), 288
 setWidth() (*depthai.ImgFrame method*), 294
 SHADE (*depthai.CameraControl.AutoWhiteBalanceMode attribute*), 255
 SHADE (*depthai.RawCameraControl.AutoWhiteBalanceMode attribute*), 317
 size() (*depthai.AssetManager method*), 250
 size() (*depthai.Rect method*), 330
 size() (*depthai.RotatedRect property*), 330
 Size2f (*class in depthai*), 331
 SMALLEST_ID (*depthai.TrackerIdAssignmentPolicy attribute*), 347
 SNOW (*depthai.CameraControl.SceneMode attribute*), 258
 SNOW (*depthai.RawCameraControl.SceneMode attribute*), 320

- SOLARIZE (*depthai.CameraControl.EffectMode* attribute), 256
- SOLARIZE (*depthai.RawCameraControl.EffectMode* attribute), 318
- spatialCoordinates () (*depthai.SpatialImgDetection* property), 334
- spatialCoordinates () (*depthai.SpatialLocations* property), 338
- spatialCoordinates () (*depthai.Tracklet* property), 349
- SpatialDetectionNetwork (class in *depthai*), 331
- SpatialDetectionNetworkProperties (class in *depthai*), 333
- SpatialImgDetection (class in *depthai*), 333
- SpatialImgDetections (class in *depthai*), 334
- SpatialLocationCalculator (class in *depthai*), 334
- SpatialLocationCalculatorConfig (class in *depthai*), 335
- SpatialLocationCalculatorConfigData (class in *depthai*), 336
- SpatialLocationCalculatorConfigThresholds (class in *depthai*), 336
- SpatialLocationCalculatorData (class in *depthai*), 336
- SpatialLocationCalculatorProperties (class in *depthai*), 337
- SpatialLocations (class in *depthai*), 337
- SPIOut (class in *depthai*), 330
- SPORTS (*depthai.CameraControl.SceneMode* attribute), 258
- SPORTS (*depthai.RawCameraControl.SceneMode* attribute), 320
- srcImgDetection () (*depthai.Tracklet* property), 349
- startPipeline () (*depthai.Device* method), 280
- state () (*depthai.DeviceInfo* property), 283
- status () (*depthai.Tracklet* property), 349
- STEADYPHOTO (*depthai.CameraControl.SceneMode* attribute), 258
- STEADYPHOTO (*depthai.RawCameraControl.SceneMode* attribute), 320
- StereoDepth (class in *depthai*), 338
- StereoDepthProperties (class in *depthai*), 340
- StereoDepthProperties.MedianFilter (class in *depthai*), 341
- still () (*depthai.ColorCamera* property), 266
- stillHeight () (*depthai.ColorCameraProperties* property), 268
- stillWidth () (*depthai.ColorCameraProperties* property), 268
- stride () (*depthai.ImgFrame.Specs* property), 290
- stride () (*depthai.RawImgFrame.Specs* property), 324
- strides () (*depthai.TensorInfo* property), 346
- SUNSET (*depthai.CameraControl.SceneMode* attribute), 258
- SUNSET (*depthai.RawCameraControl.SceneMode* attribute), 320
- syncedLeft () (*depthai.StereoDepth* property), 340
- syncedRight () (*depthai.StereoDepth* property), 340
- SystemInformation (class in *depthai*), 342
- SystemLogger (class in *depthai*), 343
- SystemLoggerProperties (class in *depthai*), 343
- ## T
- TensorInfo (class in *depthai*), 343
- TensorInfo.DataType (class in *depthai*), 344
- TensorInfo.StorageOrder (class in *depthai*), 344
- tensors () (*depthai.RawNNData* property), 328
- THE_1080_P (*depthai.ColorCameraProperties.SensorResolution* attribute), 268
- THE_12_MP (*depthai.ColorCameraProperties.SensorResolution* attribute), 268
- THE_400_P (*depthai.MonoCameraProperties.SensorResolution* attribute), 299
- THE_4_K (*depthai.ColorCameraProperties.SensorResolution* attribute), 268
- THE_720_P (*depthai.MonoCameraProperties.SensorResolution* attribute), 299
- THE_800_P (*depthai.MonoCameraProperties.SensorResolution* attribute), 299
- THEATRE (*depthai.CameraControl.SceneMode* attribute), 258
- THEATRE (*depthai.RawCameraControl.SceneMode* attribute), 320
- Timestamp (class in *depthai*), 346
- topLeft () (*depthai.Rect* method), 330
- total () (*depthai.MemoryInfo* property), 295
- TRACE (*depthai.LogLevel* attribute), 295
- TRACKED (*depthai.Tracklet.TrackingStatus* attribute), 348
- TrackerIdAssignmentPolicy (class in *depthai*), 346
- trackerIdAssignmentPolicy () (*depthai.ObjectTrackerProperties* property), 308
- trackerThreshold () (*depthai.ObjectTrackerProperties* property), 308
- TrackerType (class in *depthai*), 347
- trackerType () (*depthai.ObjectTrackerProperties* property), 308
- Tracklet (class in *depthai*), 347
- Tracklet.TrackingStatus (class in *depthai*), 348
- Tracklets (class in *depthai*), 349
- tracklets () (*depthai.RawTracklets* property), 329
- tracklets () (*depthai.Tracklets* property), 349

- tryGet() (*depthai.DataOutputQueue* method), 271
- tryGetAll() (*depthai.DataOutputQueue* method), 271
- ts() (*depthai.RawImgFrame* property), 327
- TWILIGHT (*depthai.CameraControl.AutoWhiteBalanceMode* attribute), 255
- TWILIGHT (*depthai.RawCameraControl.AutoWhiteBalanceMode* attribute), 317
- type() (*depthai.ImgFrame.Specs* property), 290
- type() (*depthai.RawImageManipConfig.FormatConfig* property), 322
- type() (*depthai.RawImgFrame.Specs* property), 324
- ## U
- U8F (*depthai.TensorInfo.DataType* attribute), 344
- UNIQUE_ID (*depthai.TrackerIdAssignmentPolicy* attribute), 347
- unlink() (*depthai.Node.Output* method), 305
- unlink() (*depthai.Pipeline* method), 313
- UNSUPPORTED (*depthai.CameraControl.SceneMode* attribute), 258
- UNSUPPORTED (*depthai.RawCameraControl.SceneMode* attribute), 320
- upa() (*depthai.ChipTemperature* property), 261
- upperThreshold() (*depthai.SpatialLocationCalculatorConfig* property), 336
- used() (*depthai.MemoryInfo* property), 295
- ## V
- value() (*depthai.CameraBoardSocket* property), 252
- value() (*depthai.CameraControl.AntiBandingMode* property), 253
- value() (*depthai.CameraControl.AutoFocusMode* property), 254
- value() (*depthai.CameraControl.AutoWhiteBalanceMode* property), 255
- value() (*depthai.CameraControl.EffectMode* property), 256
- value() (*depthai.CameraControl.SceneMode* property), 258
- value() (*depthai.CameraImageOrientation* property), 261
- value() (*depthai.ColorCameraProperties.ColorOrder* property), 267
- value() (*depthai.ColorCameraProperties.SensorResolution* property), 268
- value() (*depthai.ImgFrame.Type* property), 293
- value() (*depthai.LogLevel* property), 295
- value() (*depthai.MonoCameraProperties.SensorResolution* property), 299
- value() (*depthai.OpenVINO.Version* property), 310
- value() (*depthai.RawCameraControl.AntiBandingMode* property), 315
- value() (*depthai.RawCameraControl.AutoFocusMode* property), 316
- value() (*depthai.RawCameraControl.AutoWhiteBalanceMode* property), 317
- value() (*depthai.RawCameraControl.EffectMode* property), 318
- value() (*depthai.RawCameraControl.SceneMode* property), 320
- value() (*depthai.RawImgFrame.Type* property), 327
- value() (*depthai.StereoDepthProperties.MedianFilter* property), 342
- value() (*depthai.TensorInfo.DataType* property), 344
- value() (*depthai.TensorInfo.StorageOrder* property), 346
- value() (*depthai.TrackerIdAssignmentPolicy* property), 347
- value() (*depthai.TrackerType* property), 347
- value() (*depthai.Tracklet.TrackingStatus* property), 348
- value() (*depthai.VideoEncoderProperties.Profile* property), 354
- value() (*depthai.VideoEncoderProperties.RateControlMode* property), 354
- value() (*depthai.XLinkDeviceState* property), 356
- value() (*depthai.XLinkPlatform* property), 358
- value() (*depthai.XLinkProtocol* property), 359
- VBR (*depthai.VideoEncoderProperties.RateControlMode* attribute), 354
- VERSION_2020_1 (*depthai.OpenVINO* attribute), 309
- VERSION_2020_1 (*depthai.OpenVINO.Version* attribute), 310
- VERSION_2020_2 (*depthai.OpenVINO* attribute), 309
- VERSION_2020_2 (*depthai.OpenVINO.Version* attribute), 310
- VERSION_2020_3 (*depthai.OpenVINO* attribute), 309
- VERSION_2020_3 (*depthai.OpenVINO.Version* attribute), 310
- VERSION_2020_4 (*depthai.OpenVINO* attribute), 309
- VERSION_2020_4 (*depthai.OpenVINO.Version* attribute), 310
- VERSION_2021_1 (*depthai.OpenVINO* attribute), 309
- VERSION_2021_1 (*depthai.OpenVINO.Version* attribute), 310
- VERSION_2021_2 (*depthai.OpenVINO* attribute), 309
- VERSION_2021_2 (*depthai.OpenVINO.Version* attribute), 310
- VERSION_2021_3 (*depthai.OpenVINO* attribute), 309
- VERSION_2021_3 (*depthai.OpenVINO.Version* attribute), 310
- VERTICAL_FLIP (*depthai.CameraImageOrientation* attribute), 261
- video() (*depthai.ColorCamera* property), 266
- VideoEncoder (class in *depthai*), 349
- VideoEncoderProperties (class in *depthai*), 352

VideoEncoderProperties.Profile (class in *depthai*), 353
 VideoEncoderProperties.RateControlMode (class in *depthai*), 354
 videoHeight() (*depthai.ColorCameraProperties* property), 268
 videoWidth() (*depthai.ColorCameraProperties* property), 268

W

W (*depthai.TensorInfo.StorageOrder* attribute), 346
 WARM_FLUORESCENT (*depthai.CameraControl.AutoWhiteBalanceMode* attribute), 255
 WARM_FLUORESCENT (*depthai.RawCameraControl.AutoWhiteBalanceMode* attribute), 317
 WARN (*depthai.LogLevel* attribute), 295
 warpBorderReplicate() (*depthai.RawImageManipConfig.ResizeConfig* property), 323
 warpFourPoints() (*depthai.RawImageManipConfig.ResizeConfig* property), 323
 warpMatrix3x3() (*depthai.RawImageManipConfig.ResizeConfig* property), 323
 WCH (*depthai.TensorInfo.StorageOrder* attribute), 346
 WHC (*depthai.TensorInfo.StorageOrder* attribute), 346
 WHITEBOARD (*depthai.CameraControl.EffectMode* attribute), 256
 WHITEBOARD (*depthai.RawCameraControl.EffectMode* attribute), 318
 width() (*depthai.ImgFrame.Specs* property), 290
 width() (*depthai.RawImageManipConfig.ResizeConfig* property), 323
 width() (*depthai.RawImgFrame.Specs* property), 324
 width() (*depthai.Rect* property), 330
 width() (*depthai.Size2f* property), 331
 width() (*depthai.StereoDepthProperties* property), 342
 width() (*depthai.VideoEncoderProperties* property), 354
 widthHeightAspectRatio() (*depthai.RawImageManipConfig.CropConfig* property), 321

X

x() (*depthai.Point2f* property), 314
 x() (*depthai.Point3f* property), 314
 x() (*depthai.Rect* property), 330
 X_LINK_ANY_PLATFORM (*depthai.XLinkPlatform* attribute), 358
 X_LINK_ANY_PROTOCOL (*depthai.XLinkProtocol* attribute), 358
 X_LINK_ANY_STATE (*depthai.XLinkDeviceState* attribute), 355
 X_LINK_BOOTED (*depthai.XLinkDeviceState* attribute), 355

X_LINK_BOOTLOADER (*depthai.XLinkDeviceState* attribute), 355
 X_LINK_IPC (*depthai.XLinkProtocol* attribute), 358
 X_LINK_MYRIAD_2 (*depthai.XLinkPlatform* attribute), 358
 X_LINK_MYRIAD_X (*depthai.XLinkPlatform* attribute), 358
 X_LINK_NMB_OF_PROTOCOLS (*depthai.XLinkProtocol* attribute), 358
 X_LINK_PCIE (*depthai.XLinkProtocol* attribute), 359
 X_LINK_UNBOOTED (*depthai.XLinkDeviceState* attribute), 355
 X_LINK_USB_CDC (*depthai.XLinkProtocol* attribute), 359
 X_LINK_USB_VSC (*depthai.XLinkProtocol* attribute), 359
 XLinkConnection (class in *depthai*), 355
 XLinkDeviceState (class in *depthai*), 355
 XLinkIn (class in *depthai*), 356
 XLinkOut (class in *depthai*), 356
 XLinkPlatform (class in *depthai*), 357
 XLinkProtocol (class in *depthai*), 358
 xmax() (*depthai.ImgDetection* property), 288
 xmax() (*depthai.RawImageManipConfig.CropRect* property), 321
 xmin() (*depthai.ImgDetection* property), 289
 xmin() (*depthai.RawImageManipConfig.CropRect* property), 321

Y

y() (*depthai.Point2f* property), 314
 y() (*depthai.Point3f* property), 314
 y() (*depthai.Rect* property), 330
 ymax() (*depthai.ImgDetection* property), 289
 ymax() (*depthai.RawImageManipConfig.CropRect* property), 321
 ymin() (*depthai.ImgDetection* property), 289
 ymin() (*depthai.RawImageManipConfig.CropRect* property), 321
 YoloDetectionNetwork (class in *depthai*), 359
 YoloSpatialDetectionNetwork (class in *depthai*), 359
 YUV400p (*depthai.ImgFrame.Type* attribute), 293
 YUV400p (*depthai.RawImgFrame.Type* attribute), 327
 YUV420p (*depthai.ImgFrame.Type* attribute), 293
 YUV420p (*depthai.RawImgFrame.Type* attribute), 327
 YUV422i (*depthai.ImgFrame.Type* attribute), 293
 YUV422i (*depthai.RawImgFrame.Type* attribute), 327
 YUV422p (*depthai.ImgFrame.Type* attribute), 293
 YUV422p (*depthai.RawImgFrame.Type* attribute), 327
 YUV444i (*depthai.ImgFrame.Type* attribute), 293
 YUV444i (*depthai.RawImgFrame.Type* attribute), 327
 YUV444p (*depthai.ImgFrame.Type* attribute), 293
 YUV444p (*depthai.RawImgFrame.Type* attribute), 327

Z

`z ()` (*depthai.Point3f* property), [314](#)

`ZERO_TERM_COLOR_HISTOGRAM`

(*depthai.TrackerType* attribute), [347](#)

`ZERO_TERM_IMAGELESS` (*depthai.TrackerType*
attribute), [347](#)