DepthAI API Docs

Release 2.2.1.0.dev+d99fa42233320b513fe8d4a4f87b8d7f5e63ab29

Luxonis

Apr 27, 2021

CONTENT:

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*

CHAPTER ONE

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.

CHAPTER

TWO

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.

CHAPTER

THREE

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, it's 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 backgound 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 ${ID_VENDOR}." >&2
virsh "${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 decribed 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. And 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

```
            DepthAI device (eg. LUX-D)                                  Host (eg.␣
↪RaspberryPi)
───────────────────────────────────────────────                  ␣
↪─────────────────────────
│                                               │                   │               ␣
↪         │
│       Node                            Node    │                   │ # Your python␣
↪code that │
│  ─────────────                  ──────────                    │ # runs on the host ␣
↪     │
│  │             │                   │          │                   │               ␣
↪         │
│  │             │preview      input│          │   XLink protocol  │ # Get the frame␣
↪         │
│  │ ColorCamera ├───────────────────┤ XLinkOut ├──────────────────┤ data=q_preview.
↪get()     │
│  │             │     ImgFrame      │          │(USB/Ethernet/PCIe)│ frame=data.
↪getCvFrame() │
│  │             │     Message       │          │                   │ # Show the␣
↪frame        │
│  └─────────────┘                   └──────────┘                  │ cv2.imshow("rgb",
↪frame) │
│ inputControl                                  │                   │               ␣
↪        │
│              │                                │                   │               ␣
↪         │
│              │                      Node      │                   │               ␣
↪         │
│              │                   ──────────                      │ # Control the␣
↪camera     │
│              │                    │           │                   │ cc=dai.
↪CameraControl() │
│              │                out │           │   XLink protocol  │ cc.
↪setManualFocus(100) │
│              └────────────────────┤ XLinkIn  │├─────────────────── q_cam_control.
↪send(cc)   │
│             CameraControl         │           │(USB/Ethernet/PCIe)│               ␣
↪         │
│               Message             │           │                   │               ␣
↪         │
│                                   └──────────┘                   │               ␣
↪        │
│                                               │                   │               ␣
↪         │
```

(continues on next page)

(continued from previous page)

```
└─────────────────────────────────────────────────────────────────────────────────── ␣
↪ └─────────────────────────────
                                        A simple pipeline visualzied
```

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 recieve 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 recieve 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 it's 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

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) → 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) → 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 transfered 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 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> &*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 *embeddedMvcmd*, bool *usb2Mode*, **const** std::string &*pathToMvcmd*)

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**

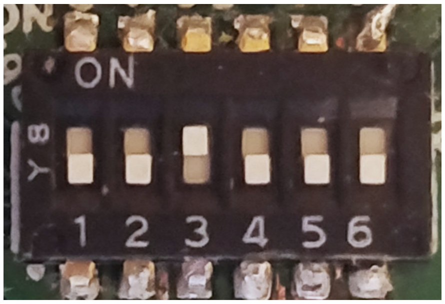
```
import depthai as dai
(f, bl) = dai.DeviceBootloader.getFirstAvailableDevice()
bootloader = dai.DeviceBootloader(bl)
progress = lambda p : print(f'Flashing progress: {p*100:.1f}%')
bootloader.flashBootloader(progress)
```

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

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

C++

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

Inputs and Outputs

```
               ┌───────────────────┐  still
               │                   ├───────────►
inputConfig    │                   │
──────────────►│                   │  preview
               │    ColorCamera    ├───────────►
inputControl   │                   │
──────────────►│                   │  video
               │                   ├───────────►
               └───────────────────┘
```

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

```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++

```cpp
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

```
              ┌───────────────────┐
inputImage    │                   │
──────────────►                   │
              │                   │        out
              │    ImageManip     ├───────────
inputConfig   │                   │
──────────────►                   │
              │                   │
              └───────────────────┘
```

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 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 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

```
                  ┌───────────────────┐
                  │                   │       out
                  │                   ├───────────
                  │     MobileNet     │
                  │     Detection     │
input             │     Network       │ passthrough
──────────────────►-------------------├───────────
                  │                   │
                  └───────────────────┘
```

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 avilable 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 *SpatialLocationCalculator*.

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

```
                ┌───────────────────┐
input           │                   │       passthrough
───────────────►│-------------------├─────────────────►
                │     MobileNet     │               out
                │     Spatial       ├─────────────────►
                │     Detection     │boundingBoxMapping
                │     Network       ├─────────────────►
inputDepth      │                   │  passthroughDepth
───────────────►│-------------------├─────────────────►
                └───────────────────┘
```

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 ingore 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 ingore 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 milimeters 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 milimeters 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 avilable 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

```
               ┌──────────────────┐
               │                  │
inputControl   │                  │       out
──────────────►│    MonoCamera    ├───────────►
               │                  │
               │                  │
               └──────────────────┘
```

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

```
                 ___________________
                |                   |        out
                |                   |______________
                |   NeuralNetwork   |
input           |                   |  passthrough
───────────────├-------------------├──────────────
                |                   |
                |___________________|
```

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

```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++

```cpp
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 infered 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 avilable 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 avilable 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 infered 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

```
               ┌───────────────────┐
               │                   │
inputConfig    │                   │       out
──────────────►│      Spatial      ├───────────►
               │      location     │
               │     calculator    │
inputDepth     │                   │passthroughDepth
──────────────►│-------------------├───────────►
               └───────────────────┘
```

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 identificator. 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*::*SpatialLocationCalculatorCo*
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

```
              ┌───────────────────┐
              │                   │  SPI (to MCU)
input         │                   │
─────────────►│      SPIOut       ├------------>
              │                   │
              │                   │
              └───────────────────┘
```

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 dispartiy/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

```
                ┌───────────────────┐
                │                   │rectifiedLeft
                │                   ├─────────────
left            │                   │   syncedLeft
───────────────►│                   ├─────────────►
                │                   │        depth
                │                   ├─────────────►
                │    StereoDepth    │    disparity
                │                   ├─────────────►
right           │                   │rectifiedRight
───────────────►│                   ├─────────────►
                │                   │  syncedRight
                │                   ├─────────────►
                └───────────────────┘
```

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 identificator. 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 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.

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

```
┌──────────────┐
│              │      out
│ SystemLogger │──────────
│              │
└──────────────┘
```

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 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 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

```
              ┌──────────────┐
              │              │bitstream
  input       │              │
─────────────►│ VideoEncoder ├───────────►
              │              │
              │              │
              └──────────────┘
```

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

```
              ______________
(From host)  |              |       out
-----------> |   XLinkIn    |-----------
             |              |
             |______________|
```

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

```
           ┌───────────┐
input      │           │     (to host)
───────────►│ XLinkOut  ├------------
           │           │
           └───────────┘
```

Message types

- `input` - `Any`

Usage

Python

```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++

```cpp
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 identificator. 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

```
              ┌───────────────────┐
              │                   │       out
              │                   ├───────────
              │     Yolo          │
              │     Detection     │
input         │     Network       │ passthrough
──────────────┤-------------------├───────────
              │                   │
              └───────────────────┘
```

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 coordianate 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 avilable 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 coordianate 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

```
                  ┌───────────────────┐
input             │                   │       passthrough
─────────────────►│-------------------├─────────────────►
                  │      Yolo         │               out
                  │      Spatial      ├─────────────────►
                  │      Detection    │boundingBoxMapping
                  │      Network      ├─────────────────►
inputDepth        │                   │   passthroughDepth
─────────────────►│-------------------├─────────────────►
                  └───────────────────┘
```

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 coordianate size

setDepthLowerThreshold (*self: depthai.SpatialDetectionNetwork*, *lowerThreshold: int*) → None
Specifies lower threshold in milimeters 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 milimeters 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 avilable 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 coordianate 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, confifguring 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 is 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 streched 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 suppling 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 suppling 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 streched 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
Retrievies 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
Retrievies image height in pixels

getInstanceNum(*self:* depthai.ImgFrame) → int
Retrievies instance number

getRaw(*self:* depthai.ADatatype) → *depthai.RawBuffer*

getSequenceNum(*self:* depthai.ImgFrame) → int
Retrievies image sequence number

getTimestamp(*self:* depthai.ImgFrame) → datetime.timedelta
Retrievies image timestamp related to steady_clock / time.monotonic

getType(*self:* depthai.ImgFrame) → *depthai.RawImgFrame.Type*
Retrieves image type

getWidth(*self:* depthai.ImgFrame) → int
Retrievies 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**

Retrievies image timestamp related to steady_clock / time.monotonic

unsigned int **getInstanceNum**() **const**

Retrievies instance number

unsigned int **getCategory**() **const**

Retrievies image category

unsigned int **getSequenceNum**() **const**

Retrievies image sequence number

unsigned int **getWidth**() **const**

Retrievies image width in pixels

unsigned int **getHeight**() **const**

Retrievies 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 recieve 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 *Neural-Network* 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]

Convinience function to retrieve float values from first layers FP16 tensor

Returns Float data

getFirstLayerInt32(*self*: depthai.NNData) → List[int]

Convinience function to retrieve INT32 values from first layers tensor

Returns INT32 data

getFirstLayerUInt8(*self*: depthai.NNData) → List[int]

Convinience 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 infromation 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]
Convinience 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]
Convinience 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]
Convinience 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 infromation 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**
Convinience 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**
Convinience 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**
Convinience function to retrieve INT32 values from layers tensor

Return INT32 data

Parameters

- `name`: Name of the layer

std::vector<std::uint8_t> **getFirstLayerUInt8**() **const**
Convinience function to retrieve U8 data from first layer

Return U8 binary data

std::vector<float> **getFirstLayerFp16**() **const**
Convinience function to retrieve float values from first layers FP16 tensor

Return Float data

std::vector<std::int32_t> **getFirstLayerInt32**() **const**
Convinience 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`:

```python
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_form`, 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

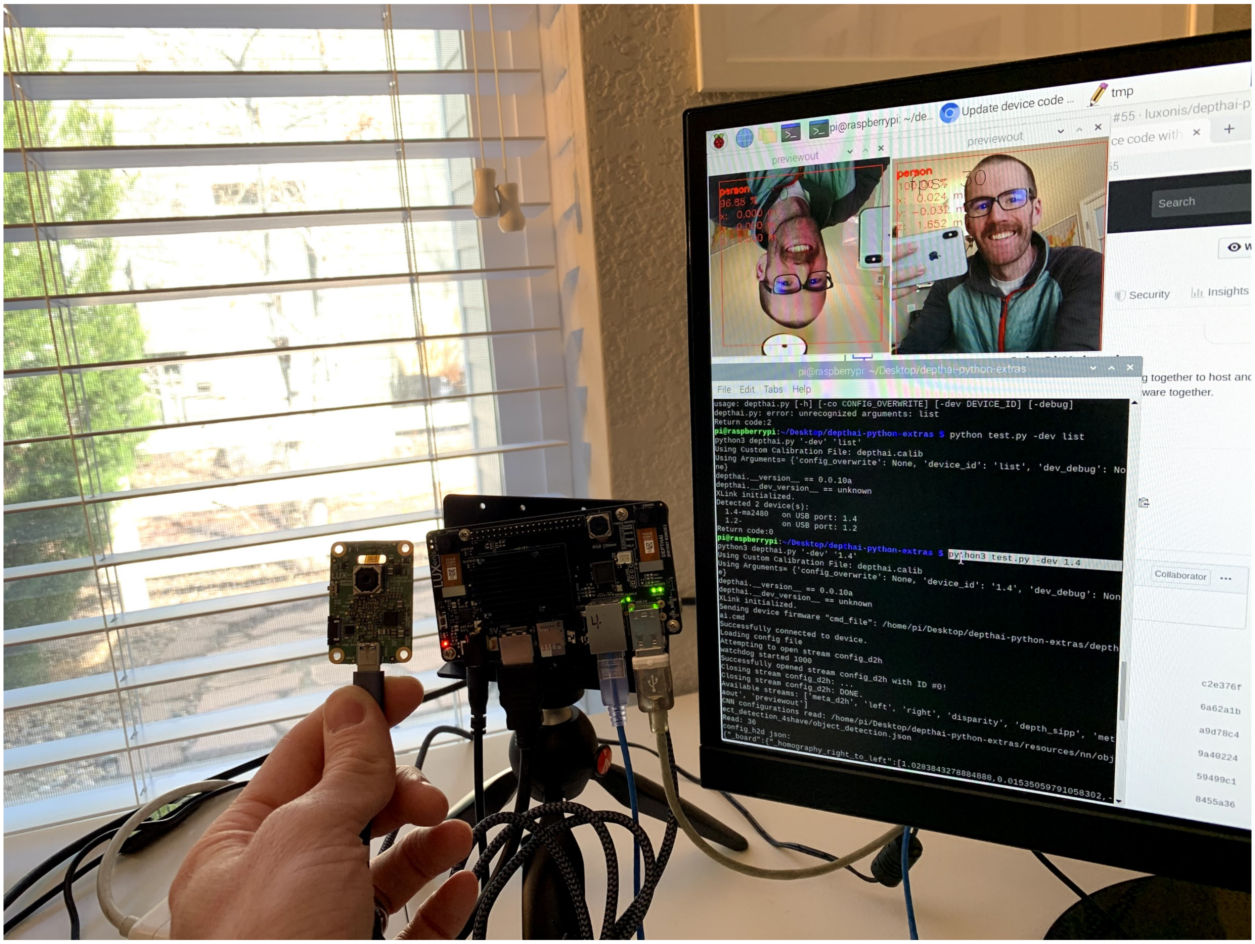
```
python3 hello_world.py
```

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.

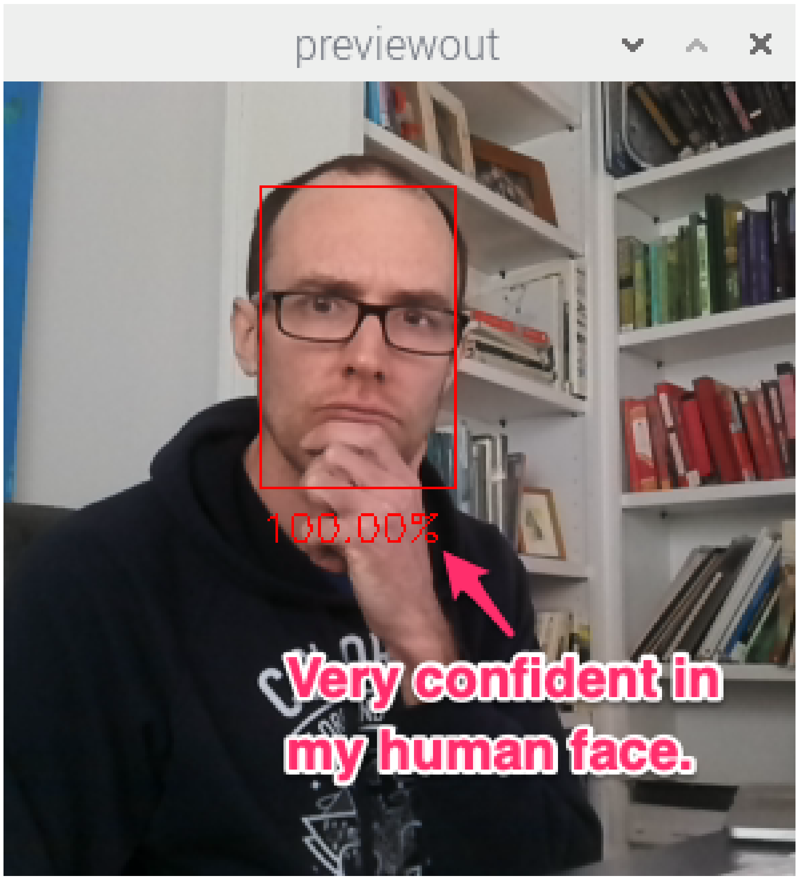
Now use as many DepthAI devices as you need!

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 wee 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.

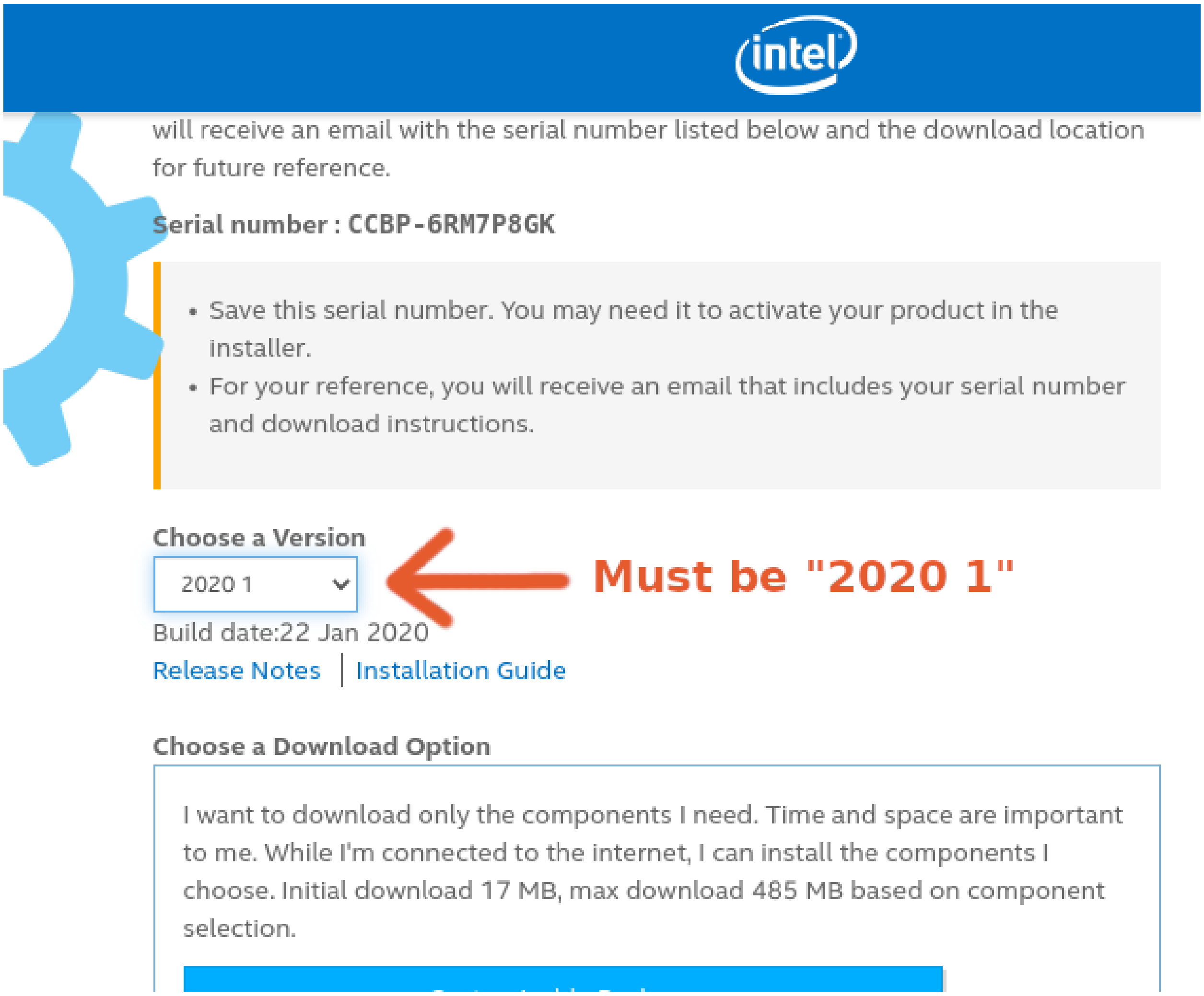
Let's verify that a correct version is installed on your host. Check your version by running the following from a terminal session:

```
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:



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

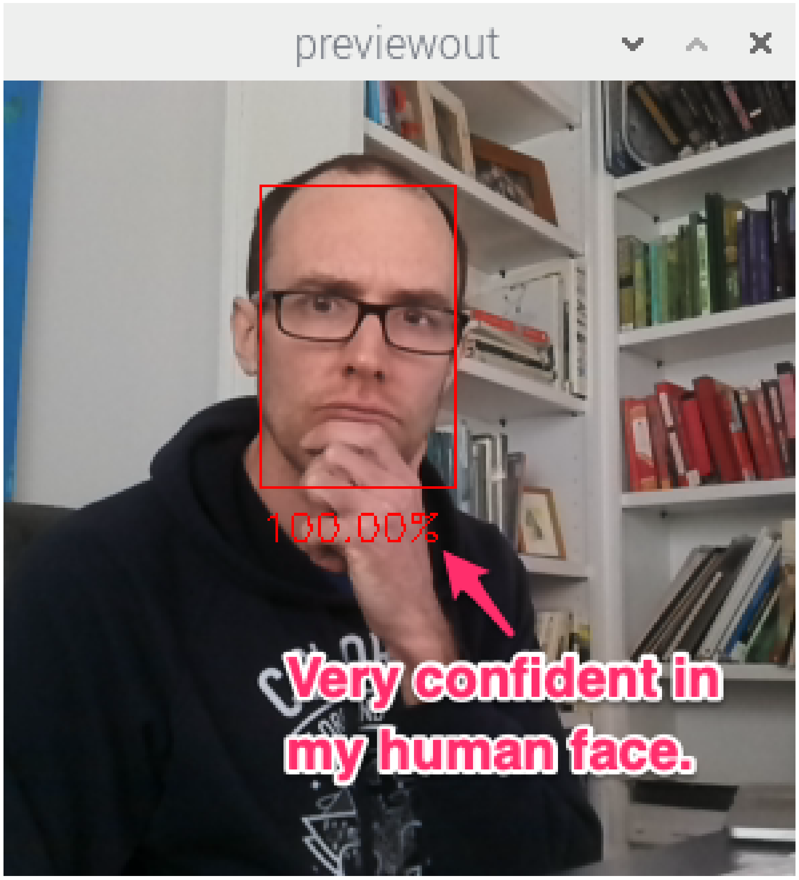
With neural network `blob` in place, we're ready to roll! To verify that the model is running correctly, let's modify a bit the program we've created in *Hello World* tutorial

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 outpus 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

```python
#!/usr/bin/env python3

import cv2
import depthai as dai

# Start defining a pipeline
pipeline = dai.Pipeline()

# Define a source - color camera
camRgb = pipeline.createColorCamera()
camRgb.setPreviewSize(300, 300)
camRgb.setBoardSocket(dai.CameraBoardSocket.RGB)
camRgb.setResolution(dai.ColorCameraProperties.SensorResolution.THE_1080_P)
camRgb.setInterleaved(False)
camRgb.setColorOrder(dai.ColorCameraProperties.ColorOrder.RGB)

# Create output
xoutRgb = pipeline.createXLinkOut()
xoutRgb.setStreamName("rgb")
camRgb.preview.link(xoutRgb.input)

# Pipeline is defined, now we can connect to the device
with dai.Device(pipeline) as device:
    # Start pipeline
    device.startPipeline()

    # Output queue will be used to get the rgb frames from the output defined above
    qRgb = device.getOutputQueue(name="rgb", maxSize=4, blocking=False)

    while True:
        inRgb = qRgb.get()  # blocking call, will wait until a new data has arrived
```

(continues on next page)

(continued from previous page)

```python

        # Retrieve 'bgr' (opencv format) frame
        cv2.imshow("bgr", inRgb.getCvFrame())

        if cv2.waitKey(1) == ord('q'):
            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

```python
#!/usr/bin/env python3

import cv2
import depthai as dai

# Start defining a pipeline
pipeline = dai.Pipeline()

# Define a source - two mono (grayscale) cameras
camLeft = pipeline.createMonoCamera()
camLeft.setBoardSocket(dai.CameraBoardSocket.LEFT)
camLeft.setResolution(dai.MonoCameraProperties.SensorResolution.THE_720_P)

camRight = pipeline.createMonoCamera()
camRight.setBoardSocket(dai.CameraBoardSocket.RIGHT)
camRight.setResolution(dai.MonoCameraProperties.SensorResolution.THE_720_P)

# Create outputs
xoutLeft = pipeline.createXLinkOut()
xoutLeft.setStreamName('left')
camLeft.out.link(xoutLeft.input)

xoutRight = pipeline.createXLinkOut()
```

(continues on next page)

(continued from previous page)

```
xoutRight.setStreamName('right')
camRight.out.link(xoutRight.input)

# Pipeline is defined, now we can connect to the device
with dai.Device(pipeline) as device:
    # Start pipeline
    device.startPipeline()

    # Output queues will be used to get the grayscale frames from the outputs defined above
    qLeft = device.getOutputQueue(name="left", maxSize=4, blocking=False)
    qRight = device.getOutputQueue(name="right", maxSize=4, blocking=False)

    frameLeft = None
    frameRight = None

    while True:
        # Instead of get (blocking), we use tryGet (nonblocking) which will return the available data or None otherwise
        inLeft = qLeft.tryGet()
        inRight = qRight.tryGet()

        if inLeft is not None:
            frameLeft = inLeft.getCvFrame()

        if inRight is not None:
            frameRight = inRight.getCvFrame()

        # show the frames if available
        if frameLeft is not None:
            cv2.imshow("left", frameLeft)
        if frameRight is not None:
            cv2.imshow("right", frameRight)

        if cv2.waitKey(1) == ord('q'):
            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

```
#!/usr/bin/env python3

import cv2
import depthai as dai
import numpy as np


'''
If one or more of the additional depth modes (lrcheck, extended, subpixel)
are enabled, then:
 - depth output is FP16. TODO enable U16.
 - median filtering is disabled on device. TODO enable.
 - with subpixel, either depth or disparity has valid data.
Otherwise, depth output is U16 (mm) and median is functional.
But like on Gen1, either depth or disparity has valid data. TODO enable both.
'''

# Closer-in minimum depth, disparity range is doubled (from 95 to 190):
extended_disparity = False
# Better accuracy for longer distance, fractional disparity 32-levels:
subpixel = False
# Better handling for occlusions:
lr_check = False

# Start defining a pipeline
pipeline = dai.Pipeline()

# Define a source - two mono (grayscale) cameras
left = pipeline.createMonoCamera()
left.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
left.setBoardSocket(dai.CameraBoardSocket.LEFT)

right = pipeline.createMonoCamera()
right.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
right.setBoardSocket(dai.CameraBoardSocket.RIGHT)

# 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)

```python
depth = pipeline.createStereoDepth()
depth.setConfidenceThreshold(200)
depth.setOutputDepth(False)
# Options: MEDIAN_OFF, KERNEL_3x3, KERNEL_5x5, KERNEL_7x7 (default)
median = dai.StereoDepthProperties.MedianFilter.KERNEL_7x7 # For depth filtering
depth.setMedianFilter(median)

depth.setLeftRightCheck(lr_check)

# Normal disparity values range from 0..95, will be used for normalization
max_disparity = 95

if extended_disparity: max_disparity *= 2 # Double the range
depth.setExtendedDisparity(extended_disparity)

if subpixel: max_disparity *= 32 # 5 fractional bits, x32
depth.setSubpixel(subpixel)

# When we get disparity to the host, we will multiply all values with the multiplier
# for better visualization
multiplier = 255 / max_disparity

left.out.link(depth.left)
right.out.link(depth.right)

# Create output
xout = pipeline.createXLinkOut()
xout.setStreamName("disparity")
depth.disparity.link(xout.input)

# Pipeline is defined, now we can connect to the device
with dai.Device(pipeline) as device:
    # Start pipeline
    device.startPipeline()

    # Output queue will be used to get the disparity frames from the outputs defined above
    q = device.getOutputQueue(name="disparity", maxSize=4, blocking=False)
    while True:
        inDepth = q.get()  # blocking call, will wait until a new data has arrived
        frame = inDepth.getFrame()
        frame = (frame*multiplier).astype(np.uint8)
        # Available color maps: https://docs.opencv.org/3.4/d3/d50/group__imgproc__colormap.html
        frame = cv2.applyColorMap(frame, cv2.COLORMAP_JET)

        # frame is ready to be shown
        cv2.imshow("disparity", frame)

        if cv2.waitKey(1) == ord('q'):
            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 resolultion 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

```
#!/usr/bin/env python3

import depthai as dai

# Start defining a pipeline
pipeline = dai.Pipeline()

# Define a source - color camera
cam = pipeline.createColorCamera()
cam.setBoardSocket(dai.CameraBoardSocket.RGB)
cam.setResolution(dai.ColorCameraProperties.SensorResolution.THE_4_K)

# Create an encoder, consuming the frames and encoding them using H.265 encoding
videoEncoder = pipeline.createVideoEncoder()
videoEncoder.setDefaultProfilePreset(3840, 2160, 30, dai.VideoEncoderProperties.
↪Profile.H265_MAIN)
cam.video.link(videoEncoder.input)

# Create output
videoOut = pipeline.createXLinkOut()
videoOut.setStreamName('h265')
videoEncoder.bitstream.link(videoOut.input)

# Pipeline is defined, now we can connect to the device
with dai.Device(pipeline) as device:
    # Start pipeline
    device.startPipeline()
```

(continues on next page)

(continued from previous page)

```

    # Output queue will be used to get the encoded data from the output defined above
    q = device.getOutputQueue(name="h265", maxSize=30, blocking=True)

    # The .h265 file is a raw stream file (not playable yet)
    with open('video.h265', 'wb') as videoFile:
        print("Press Ctrl+C to stop encoding...")
        try:
            while True:
                h264Packet = q.get()  # Blocking call, will wait until a new data has arrived
                h264Packet.getData().tofile(videoFile)  # Appends the packet data to the opened file
        except KeyboardInterrupt:
            # Keyboard interrupt (Ctrl + C) detected
            pass

    print("To view the encoded data, convert the stream file (.h265) into a video file (.mp4) using a command below:")
    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

```python
#!/usr/bin/env python3

import depthai as dai

# Start defining a pipeline
pipeline = dai.Pipeline()

# Define a source - color and mono cameras
colorCam = pipeline.createColorCamera()
monoCam = pipeline.createMonoCamera()
monoCam.setBoardSocket(dai.CameraBoardSocket.LEFT)
monoCam2 = pipeline.createMonoCamera()
monoCam2.setBoardSocket(dai.CameraBoardSocket.RIGHT)

# Create encoders, one for each camera, consuming the frames and encoding them using H.264 / H.265 encoding
ve1 = pipeline.createVideoEncoder()
ve1.setDefaultProfilePreset(1280, 720, 30, dai.VideoEncoderProperties.Profile.H264_MAIN)
monoCam.out.link(ve1.input)

ve2 = pipeline.createVideoEncoder()
ve2.setDefaultProfilePreset(1920, 1080, 30, dai.VideoEncoderProperties.Profile.H265_MAIN)
colorCam.video.link(ve2.input)

ve3 = pipeline.createVideoEncoder()
ve3.setDefaultProfilePreset(1280, 720, 30, dai.VideoEncoderProperties.Profile.H264_MAIN)
monoCam2.out.link(ve3.input)

# Create outputs
ve1Out = pipeline.createXLinkOut()
ve1Out.setStreamName('ve1Out')
ve1.bitstream.link(ve1Out.input)

ve2Out = pipeline.createXLinkOut()
ve2Out.setStreamName('ve2Out')
ve2.bitstream.link(ve2Out.input)

ve3Out = pipeline.createXLinkOut()
ve3Out.setStreamName('ve3Out')
ve3.bitstream.link(ve3Out.input)


# Pipeline is defined, now we can connect to the device
with dai.Device(pipeline) as dev:
    # Start pipeline
    dev.startPipeline()

    # Output queues will be used to get the encoded data from the outputs defined above
    outQ1 = dev.getOutputQueue(name='ve1Out', maxSize=30, blocking=True)
    outQ2 = dev.getOutputQueue(name='ve2Out', maxSize=30, blocking=True)
```

(continues on next page)

(continued from previous page)

```
    outQ3 = dev.getOutputQueue(name='ve3Out', maxSize=30, blocking=True)

    # The .h264 / .h265 files are raw stream files (not playable yet)
    with open('mono1.h264', 'wb') as fileMono1H264, open('color.h265', 'wb') as fileColorH265, open('mono2.h264', 'wb') as fileMono2H264:
        print("Press Ctrl+C to stop encoding...")
        while True:
            try:
                # Empty each queue
                while outQ1.has():
                    outQ1.get().getData().tofile(fileMono1H264)

                while outQ2.has():
                    outQ2.get().getData().tofile(fileColorH265)

                while outQ3.has():
                    outQ3.get().getData().tofile(fileMono2H264)
            except KeyboardInterrupt:
                # Keyboard interrupt (Ctrl + C) detected
                break

    print("To view the encoded data, convert the stream file (.h264/.h265) into a video file (.mp4), using commands below:")
    cmd = "ffmpeg -framerate 30 -i {} -c copy {}"
    print(cmd.format("mono1.h264", "mono1.mp4"))
    print(cmd.format("mono2.h264", "mono2.mp4"))
    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 at 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

```
#!/usr/bin/env python3

import time
from pathlib import Path

import cv2
import depthai as dai

# Start defining a pipeline
pipeline = dai.Pipeline()

# Define a source - color camera
camRgb = pipeline.createColorCamera()
camRgb.setResolution(dai.ColorCameraProperties.SensorResolution.THE_4_K)

# Create RGB output
xoutRgb = pipeline.createXLinkOut()
xoutRgb.setStreamName("rgb")
camRgb.video.link(xoutRgb.input)

# Create encoder to produce JPEG images
videoEnc = pipeline.createVideoEncoder()
videoEnc.setDefaultProfilePreset(camRgb.getVideoSize(), camRgb.getFps(), dai.
↪VideoEncoderProperties.Profile.MJPEG)
camRgb.video.link(videoEnc.input)

# Create JPEG output
xoutJpeg = pipeline.createXLinkOut()
xoutJpeg.setStreamName("jpeg")
videoEnc.bitstream.link(xoutJpeg.input)


# Pipeline is defined, now we can connect to the device
with dai.Device(pipeline) as device:
    # Start pipeline
    device.startPipeline()

    # Output queue will be used to get the rgb frames from the output defined above
    qRgb = device.getOutputQueue(name="rgb", maxSize=30, blocking=False)
    qJpeg = device.getOutputQueue(name="jpeg", maxSize=30, blocking=True)

    # Make sure the destination path is present before starting to store the examples
    Path('06_data').mkdir(parents=True, exist_ok=True)

    while True:
        inRgb = qRgb.tryGet()  # Non-blocking call, will return a new data that has␣
↪arrived or None otherwise

        if inRgb is not None:
            cv2.imshow("rgb", inRgb.getCvFrame())

        for encFrame in qJpeg.tryGetAll():
            with open(f"06_data/{int(time.time() * 10000)}.jpeg", "wb") as f:
                f.write(bytearray(encFrame.getData()))
```

(continues on next page)

(continued from previous page)

```

        if cv2.waitKey(1) == ord('q'):
            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

```
#!/usr/bin/env python3

import time
from pathlib import Path

import cv2
import depthai as dai

# Start defining a pipeline
pipeline = dai.Pipeline()

# Define a source - mono (grayscale) camera
camRight = pipeline.createMonoCamera()
camRight.setBoardSocket(dai.CameraBoardSocket.RIGHT)
camRight.setResolution(dai.MonoCameraProperties.SensorResolution.THE_720_P)

# Create output
xoutRight = pipeline.createXLinkOut()
xoutRight.setStreamName("right")
camRight.out.link(xoutRight.input)

# Pipeline is defined, now we can connect to the device
with dai.Device(pipeline) as device:
    # Start pipeline
    device.startPipeline()

```

(continues on next page)

(continued from previous page)

```
    # Output queue will be used to get the grayscale frames from the output defined␣
↪above
    qRight = device.getOutputQueue(name="right", maxSize=4, blocking=False)

    # Make sure the destination path is present before starting to store the examples
    Path('07_data').mkdir(parents=True, exist_ok=True)

    while True:
        inRight = qRight.get()  # Blocking call, will wait until a new data has␣
↪arrived
        # Data is originally represented as a flat 1D array, it needs to be converted␣
↪into HxW form
        frameRight = inRight.getCvFrame()
        # Frame is transformed and ready to be shown
        cv2.imshow("right", frameRight)
        # After showing the frame, it's being stored inside a target directory as a␣
↪PNG image
        cv2.imwrite(f"07_data/{int(time.time() * 10000)}.png", frameRight)

        if cv2.waitKey(1) == ord('q'):
            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 MobilenetSDD blob (`mobilenet-ssd_openvino_2021.2_6shave.blob` file) to work - you can download it from here

Source code

Also available on GitHub

```
#!/usr/bin/env python3

from pathlib import Path
import cv2
import depthai as dai
import numpy as np
import time
import argparse

nnPathDefault = str((Path(__file__).parent / Path('models/mobilenet-ssd_openvino_2021.
↪2_6shave.blob')).resolve().absolute())
parser = argparse.ArgumentParser()
parser.add_argument('nnPath', nargs='?', help="Path to mobilenet detection network␣
↪blob", default=nnPathDefault)
parser.add_argument('-s', '--sync', action="store_true", help="Sync RGB output with␣
↪NN output", default=False)
args = parser.parse_args()

if not Path(nnPathDefault).exists():
    import sys
    raise FileNotFoundError(f'Required file/s not found, please run "{sys.executable}␣
↪install_requirements.py"')

# Start defining a pipeline
pipeline = dai.Pipeline()

# Define a source - color camera
camRgb = pipeline.createColorCamera()
camRgb.setPreviewSize(300, 300)
camRgb.setInterleaved(False)
camRgb.setFps(40)

# Define a neural network that will make predictions based on the source frames
nn = pipeline.createMobileNetDetectionNetwork()
nn.setConfidenceThreshold(0.5)
nn.setBlobPath(args.nnPath)
nn.setNumInferenceThreads(2)
nn.input.setBlocking(False)
camRgb.preview.link(nn.input)

# Create outputs
xoutRgb = pipeline.createXLinkOut()
xoutRgb.setStreamName("rgb")
if args.sync:
    nn.passthrough.link(xoutRgb.input)
else:
    camRgb.preview.link(xoutRgb.input)

nnOut = pipeline.createXLinkOut()
nnOut.setStreamName("nn")
nn.out.link(nnOut.input)

# MobilenetSSD label texts
labelMap = ["background", "aeroplane", "bicycle", "bird", "boat", "bottle", "bus",
↪"car", "cat", "chair", "cow",
```

(continues on next page)

(continued from previous page)

```
            "diningtable", "dog", "horse", "motorbike", "person", "pottedplant", "sheep", "sofa", "train", "tvmonitor"]


# Pipeline is defined, now we can connect to the device
with dai.Device(pipeline) as device:
    # Start pipeline
    device.startPipeline()

    # Output queues will be used to get the rgb frames and nn data from the outputs defined above
    qRgb = device.getOutputQueue(name="rgb", maxSize=4, blocking=False)
    qDet = device.getOutputQueue(name="nn", maxSize=4, blocking=False)

    startTime = time.monotonic()
    counter = 0
    detections = []
    frame = None

    # nn data (bounding box locations) are in <0..1> range - they need to be normalized with frame width/height
    def frameNorm(frame, bbox):
        normVals = np.full(len(bbox), frame.shape[0])
        normVals[::2] = frame.shape[1]
        return (np.clip(np.array(bbox), 0, 1) * normVals).astype(int)

    def displayFrame(name, frame):
        for detection in detections:
            bbox = frameNorm(frame, (detection.xmin, detection.ymin, detection.xmax, detection.ymax))
            cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (255, 0, 0), 2)
            cv2.putText(frame, labelMap[detection.label], (bbox[0] + 10, bbox[1] + 20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
            cv2.putText(frame, f"{int(detection.confidence * 100)}%", (bbox[0] + 10, bbox[1] + 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
        cv2.imshow(name, frame)


    while True:
        if args.sync:
            # Use blocking get() call to catch frame and inference result synced
            inRgb = qRgb.get()
            inDet = qDet.get()
        else:
            # Instead of get (blocking), we use tryGet (nonblocking) which will return the available data or None otherwise
            inRgb = qRgb.tryGet()
            inDet = qDet.tryGet()

        if inRgb is not None:
            frame = inRgb.getCvFrame()
            cv2.putText(frame, "NN fps: {:.2f}".format(counter / (time.monotonic() - startTime)),
                        (2, frame.shape[0] - 4), cv2.FONT_HERSHEY_TRIPLEX, 0.4, color=(255, 255, 255))

```

(continues on next page)

(continued from previous page)

```
        if inDet is not None:
            detections = inDet.detections
            counter += 1

        # If the frame is available, draw bounding boxes on it and show the frame
        if frame is not None:
            displayFrame("rgb", frame)

        if cv2.waitKey(1) == ord('q'):
            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 MobileNetv2SSD 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 MobilenetSDD blob (`mobilenet-ssd_openvino_2021.2_6shave.blob` file) to work - you can download it from here

Source code

Also available on GitHub

```
#!/usr/bin/env python3

from pathlib import Path
import sys
import cv2
import depthai as dai
import numpy as np

# Get argument first
nnPath = str((Path(__file__).parent / Path('models/mobilenet-ssd_openvino_2021.2_
↪6shave.blob')).resolve().absolute())
if len(sys.argv) > 1:
    nnPath = sys.argv[1]

if not Path(nnPath).exists():
    import sys
```

(continues on next page)

(continued from previous page)

```python
    raise FileNotFoundError(f'Required file/s not found, please run "{sys.executable} install_requirements.py"')

# Start defining a pipeline
pipeline = dai.Pipeline()

# Define a source - mono (grayscale) camera
camRight = pipeline.createMonoCamera()
camRight.setBoardSocket(dai.CameraBoardSocket.RIGHT)
camRight.setResolution(dai.MonoCameraProperties.SensorResolution.THE_720_P)

# Define a neural network that will make predictions based on the source frames
nn = pipeline.createMobileNetDetectionNetwork()
nn.setConfidenceThreshold(0.5)
nn.setBlobPath(nnPath)
nn.setNumInferenceThreads(2)
nn.input.setBlocking(False)

# Create a node to convert the grayscale frame into the nn-acceptable form
manip = pipeline.createImageManip()
manip.initialConfig.setResize(300, 300)
# The NN model expects BGR input. By default ImageManip output type would be same as input (gray in this case)
manip.initialConfig.setFrameType(dai.ImgFrame.Type.BGR888p)
camRight.out.link(manip.inputImage)
manip.out.link(nn.input)

# Create outputs
manipOut = pipeline.createXLinkOut()
manipOut.setStreamName("right")
manip.out.link(manipOut.input)

nnOut = pipeline.createXLinkOut()
nnOut.setStreamName("nn")
nn.out.link(nnOut.input)

# MobilenetSSD label texts
labelMap = ["background", "aeroplane", "bicycle", "bird", "boat", "bottle", "bus", "car", "cat", "chair", "cow",
            "diningtable", "dog", "horse", "motorbike", "person", "pottedplant", "sheep", "sofa", "train", "tvmonitor"]

# Pipeline is defined, now we can connect to the device
with dai.Device(pipeline) as device:
    # Start pipeline
    device.startPipeline()

    # Output queues will be used to get the grayscale frames and nn data from the outputs defined above
    qRight = device.getOutputQueue("right", maxSize=4, blocking=False)
    qDet = device.getOutputQueue("nn", maxSize=4, blocking=False)

    frame = None
    detections = []

    # 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)

```
    def frameNorm(frame, bbox):
        normVals = np.full(len(bbox), frame.shape[0])
        normVals[::2] = frame.shape[1]
        return (np.clip(np.array(bbox), 0, 1) * normVals).astype(int)

    def displayFrame(name, frame):
        for detection in detections:
            bbox = frameNorm(frame, (detection.xmin, detection.ymin, detection.xmax, detection.ymax))
            cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (255, 0, 0), 2)
            cv2.putText(frame, labelMap[detection.label], (bbox[0] + 10, bbox[1] + 20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
            cv2.putText(frame, f"{int(detection.confidence * 100)}%", (bbox[0] + 10, bbox[1] + 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
        cv2.imshow(name, frame)


    while True:
        # Instead of get (blocking), we use tryGet (nonblocking) which will return the available data or None otherwise
        inRight = qRight.tryGet()
        inDet = qDet.tryGet()

        if inRight is not None:
            frame = inRight.getCvFrame()

        if inDet is not None:
            detections = inDet.detections

        if frame is not None:
            displayFrame("right", frame)

        if cv2.waitKey(1) == ord('q'):
            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 MobilenetSDD blob (`mobilenet-ssd_openvino_2021.2_6shave.blob` file) to work - you can download it from here

Source code

Also available on GitHub

```
#!/usr/bin/env python3

from pathlib import Path
import sys
import cv2
import depthai as dai
import numpy as np


flipRectified = True

# Get argument first
nnPath = str((Path(__file__).parent / Path('models/mobilenet-ssd_openvino_2021.2_6shave.blob')).resolve().absolute())
if len(sys.argv) > 1:
    nnPath = sys.argv[1]

if not Path(nnPath).exists():
    import sys
    raise FileNotFoundError(f'Required file/s not found, please run "{sys.executable} install_requirements.py"')

# Start defining a pipeline
pipeline = dai.Pipeline()

# Define a source - mono (grayscale) cameras
left = pipeline.createMonoCamera()
left.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
left.setBoardSocket(dai.CameraBoardSocket.LEFT)

right = pipeline.createMonoCamera()
right.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
right.setBoardSocket(dai.CameraBoardSocket.RIGHT)

# Create a node that will produce the depth map (using disparity output as it's easier to visualize depth this way)
stereo = pipeline.createStereoDepth()
stereo.setOutputRectified(True)  # The rectified streams are horizontally mirrored by default
stereo.setConfidenceThreshold(255)
stereo.setRectifyEdgeFillColor(0)  # Black, to better see the cutout from rectification (black stripe on the edges)

left.out.link(stereo.left)
right.out.link(stereo.right)

# Create a node to convert the grayscale frame into the nn-acceptable form
manip = pipeline.createImageManip()
```

(continues on next page)

(continued from previous page)

```
manip.initialConfig.setResize(300, 300)
# The NN model expects BGR input. By default ImageManip output type would be same as input (gray in this case)
manip.initialConfig.setFrameType(dai.ImgFrame.Type.BGR888p)
stereo.rectifiedRight.link(manip.inputImage)

# Define a neural network that will make predictions based on the source frames
nn = pipeline.createMobileNetDetectionNetwork()
nn.setConfidenceThreshold(0.5)
nn.setBlobPath(nnPath)
nn.setNumInferenceThreads(2)
nn.input.setBlocking(False)
manip.out.link(nn.input)

# Create outputs
disparityOut = pipeline.createXLinkOut()
disparityOut.setStreamName("disparity")
stereo.disparity.link(disparityOut.input)

xoutRight = pipeline.createXLinkOut()
xoutRight.setStreamName("rectifiedRight")
manip.out.link(xoutRight.input)

nnOut = pipeline.createXLinkOut()
nnOut.setStreamName("nn")
nn.out.link(nnOut.input)

# MobilenetSSD label nnLabels
labelMap = ["background", "aeroplane", "bicycle", "bird", "boat", "bottle", "bus", "car", "cat", "chair", "cow",
            "diningtable", "dog", "horse", "motorbike", "person", "pottedplant", "sheep", "sofa", "train", "tvmonitor"]

# Pipeline is defined, now we can connect to the device
with dai.Device(pipeline) as device:
    # Start pipeline
    device.startPipeline()

    # Output queues will be used to get the grayscale / depth frames and nn data from the outputs defined above
    qRight = device.getOutputQueue("rectifiedRight", maxSize=4, blocking=False)
    qDisparity = device.getOutputQueue("disparity", maxSize=4, blocking=False)
    qDet = device.getOutputQueue("nn", maxSize=4, blocking=False)

    rightFrame = None
    depthFrame = None
    detections = []
    offsetX = (right.getResolutionWidth() - right.getResolutionHeight()) // 2
    croppedFrame = np.zeros((right.getResolutionHeight(), right.getResolutionHeight()))

    # nn data, being the bounding box locations, are in <0..1> range - they need to be normalized with frame width/height
    def frameNorm(frame, bbox):
        normVals = np.full(len(bbox), frame.shape[0])
        normVals[::2] = frame.shape[1]
        return (np.clip(np.array(bbox), 0, 1) * normVals).astype(int)
```

(continues on next page)

(continued from previous page)

```python

    # Add bounding boxes and text to the frame and show it to the user
    def show(name, frame):
        for detection in detections:
            bbox = frameNorm(frame, (detection.xmin, detection.ymin, detection.xmax, detection.ymax))
            cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (255, 0, 0), 2)
            cv2.putText(frame, labelMap[detection.label], (bbox[0] + 10, bbox[1] + 20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
            cv2.putText(frame, f"{int(detection.confidence * 100)}%", (bbox[0] + 10, bbox[1] + 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
        # Show the frame
        cv2.imshow(name, frame)

    disparity_multiplier = 255 / 95 # Disparity range is 0..95
    while True:
        # Instead of get (blocking), we use tryGet (nonblocking) which will return the available data or None otherwise
        inRight = qRight.tryGet()
        inDet = qDet.tryGet()
        inDisparity = qDisparity.tryGet()

        if inRight is not None:
            rightFrame = inRight.getCvFrame()
            if flipRectified:
                rightFrame = cv2.flip(rightFrame, 1)


        if inDet is not None:
            detections = inDet.detections
            if flipRectified:
                for detection in detections:
                    swap = detection.xmin
                    detection.xmin = 1 - detection.xmax
                    detection.xmax = 1 - swap

        if inDisparity is not None:
            # Frame is transformed, normalized, and color map will be applied to highlight the depth info
            disparityFrame = inDisparity.getFrame()
            disparityFrame = (disparityFrame*disparity_multiplier).astype(np.uint8)
            # Available color maps: https://docs.opencv.org/3.4/d3/d50/group__imgproc__colormap.html
            disparityFrame = cv2.applyColorMap(disparityFrame, cv2.COLORMAP_JET)
            show("disparity", disparityFrame)

        if rightFrame is not None:
            show("rectified right", rightFrame)

        detections = []

        if cv2.waitKey(1) == ord('q'):
            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 MobilenetSDD blob (`mobilenet-ssd_openvino_2021.2_6shave.blob` file) to work - you can download it from here

Source code

Also available on GitHub

```
#!/usr/bin/env python3

from pathlib import Path
import sys
import cv2
import depthai as dai
import numpy as np

# Get argument first
nnPath = str((Path(__file__).parent / Path('models/mobilenet-ssd_openvino_2021.2_6shave.blob')).resolve().absolute())
if len(sys.argv) > 1:
    nnPath = sys.argv[1]

if not Path(nnPath).exists():
    import sys
    raise FileNotFoundError(f'Required file/s not found, please run "{sys.executable} install_requirements.py"')

pipeline = dai.Pipeline()

cam = pipeline.createColorCamera()
cam.setBoardSocket(dai.CameraBoardSocket.RGB)
cam.setResolution(dai.ColorCameraProperties.SensorResolution.THE_1080_P)

```

(continues on next page)

(continued from previous page)

```
videoEncoder = pipeline.createVideoEncoder()
videoEncoder.setDefaultProfilePreset(1920, 1080, 30, dai.VideoEncoderProperties.Profile.H265_MAIN)
cam.video.link(videoEncoder.input)

videoOut = pipeline.createXLinkOut()
videoOut.setStreamName('h265')
videoEncoder.bitstream.link(videoOut.input)

camRight = pipeline.createMonoCamera()
camRight.setBoardSocket(dai.CameraBoardSocket.RIGHT)
camRight.setResolution(dai.MonoCameraProperties.SensorResolution.THE_720_P)

nn = pipeline.createMobileNetDetectionNetwork()
nn.setConfidenceThreshold(0.5)
nn.setBlobPath(nnPath)
nn.setNumInferenceThreads(2)
nn.input.setBlocking(False)

manip = pipeline.createImageManip()
manip.initialConfig.setResize(300, 300)
# The NN model expects BGR input. By default ImageManip output type would be same as input (gray in this case)
manip.initialConfig.setFrameType(dai.ImgFrame.Type.BGR888p)
camRight.out.link(manip.inputImage)
manip.out.link(nn.input)

xoutRight = pipeline.createXLinkOut()
xoutRight.setStreamName("right")
camRight.out.link(xoutRight.input)

manipOut = pipeline.createXLinkOut()
manipOut.setStreamName("manip")
manip.out.link(manipOut.input)

nnOut = pipeline.createXLinkOut()
nnOut.setStreamName("nn")
nn.out.link(nnOut.input)

# MobilenetSSD label texts
labelMap = ["background", "aeroplane", "bicycle", "bird", "boat", "bottle", "bus", "car", "cat", "chair", "cow",
            "diningtable", "dog", "horse", "motorbike", "person", "pottedplant", "sheep", "sofa", "train", "tvmonitor"]


# Pipeline is defined, now we can connect to the device
with dai.Device(pipeline) as device:
    # Start pipeline
    device.startPipeline()

    queue_size = 8
    qRight = device.getOutputQueue("right", queue_size)
    qManip = device.getOutputQueue("manip", queue_size)
    qDet = device.getOutputQueue("nn", queue_size)
    qRgbEnc = device.getOutputQueue('h265', maxSize=30, blocking=True)

```

(continues on next page)

(continued from previous page)

```
    frame = None
    frameManip = None
    detections = []
    offsetX = (camRight.getResolutionWidth() - camRight.getResolutionHeight()) // 2
    croppedFrame = np.zeros((camRight.getResolutionHeight(), camRight.getResolutionHeight()))

    def frameNorm(frame, bbox):
        normVals = np.full(len(bbox), frame.shape[0])
        normVals[::2] = frame.shape[1]
        return (np.clip(np.array(bbox), 0, 1) * normVals).astype(int)

    videoFile = open('video.h265', 'wb')
    cv2.namedWindow("right", cv2.WINDOW_NORMAL)
    cv2.namedWindow("manip", cv2.WINDOW_NORMAL)

    while True:
        inRight = qRight.tryGet()
        inManip = qManip.tryGet()
        inDet = qDet.tryGet()

        while qRgbEnc.has():
            qRgbEnc.get().getData().tofile(videoFile)

        if inRight is not None:
            frame = inRight.getCvFrame()

        if inManip is not None:
            frameManip = inManip.getCvFrame()

        if inDet is not None:
            detections = inDet.detections

        if frame is not None:
            for detection in detections:
                bbox = frameNorm(croppedFrame, (detection.xmin, detection.ymin, detection.xmax, detection.ymax))
                bbox[::2] += offsetX
                cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (255, 0, 0), 2)
                cv2.putText(frame, labelMap[detection.label], (bbox[0] + 10, bbox[1] + 20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
                cv2.putText(frame, f"{int(detection.confidence * 100)}%", (bbox[0] + 10, bbox[1] + 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
            cv2.imshow("right", frame)

        if frameManip is not None:
            for detection in detections:
                bbox = frameNorm(frameManip, (detection.xmin, detection.ymin, detection.xmax, detection.ymax))
                cv2.rectangle(frameManip, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (255, 0, 0), 2)
                cv2.putText(frameManip, labelMap[detection.label], (bbox[0] + 10, bbox[1] + 20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
                cv2.putText(frameManip, f"{int(detection.confidence * 100)}%", (bbox[0] + 10, bbox[1] + 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
            cv2.imshow("manip", frameManip)
```

(continues on next page)

(continued from previous page)

```

        if cv2.waitKey(1) == ord('q'):
            break

    videoFile.close()

    print("To view the encoded data, convert the stream file (.h265) into a video␣
↪file (.mp4) using a command below:")
    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 MobilenetSDD blob (`mobilenet-ssd_openvino_2021.2_6shave.blob` file) to work - you can download it from here

Source code

Also available on GitHub

```
#!/usr/bin/env python3

from pathlib import Path
import sys
import cv2
import depthai as dai
import numpy as np

```

(continues on next page)

(continued from previous page)

```
flipRectified = True

# Get argument first
nnPath = str((Path(__file__).parent / Path('models/mobilenet-ssd_openvino_2021.2_
↪6shave.blob')).resolve().absolute())
if len(sys.argv) > 1:
    nnPath = sys.argv[1]

if not Path(nnPath).exists():
    import sys
    raise FileNotFoundError(f'Required file/s not found, please run "{sys.executable}␣
↪install_requirements.py"')

pipeline = dai.Pipeline()

cam = pipeline.createColorCamera()
cam.setBoardSocket(dai.CameraBoardSocket.RGB)
cam.setResolution(dai.ColorCameraProperties.SensorResolution.THE_1080_P)

videoEncoder = pipeline.createVideoEncoder()
videoEncoder.setDefaultProfilePreset(1920, 1080, 30, dai.VideoEncoderProperties.
↪Profile.H265_MAIN)
cam.video.link(videoEncoder.input)

videoOut = pipeline.createXLinkOut()
videoOut.setStreamName('h265')
videoEncoder.bitstream.link(videoOut.input)
camLeft = pipeline.createMonoCamera()
camLeft.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
camLeft.setBoardSocket(dai.CameraBoardSocket.LEFT)

camRight = pipeline.createMonoCamera()
camRight.setBoardSocket(dai.CameraBoardSocket.RIGHT)
camRight.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)

depth = pipeline.createStereoDepth()
depth.setConfidenceThreshold(255)
# Note: the rectified streams are horizontally mirrored by default
depth.setOutputRectified(True)
depth.setRectifyMirrorFrame(False)
depth.setRectifyEdgeFillColor(0)  # Black, to better see the cutout
camLeft.out.link(depth.left)
camRight.out.link(depth.right)
# Disparity range is 0..95, used for normalization
disparity_multiplier = 255 / 95

disparityOut = pipeline.createXLinkOut()
disparityOut.setStreamName("disparity")
depth.disparity.link(disparityOut.input)

nn = pipeline.createMobileNetDetectionNetwork()
nn.setConfidenceThreshold(0.5)
nn.setBlobPath(nnPath)
nn.setNumInferenceThreads(2)
nn.input.setBlocking(False)

manip = pipeline.createImageManip()
```

(continues on next page)

(continued from previous page)

```
manip.initialConfig.setResize(300, 300)
# The NN model expects BGR input. By default ImageManip output type would be same as input (gray in this case)
manip.initialConfig.setFrameType(dai.ImgFrame.Type.BGR888p)
depth.rectifiedRight.link(manip.inputImage)
manip.out.link(nn.input)

xoutRight = pipeline.createXLinkOut()
xoutRight.setStreamName("right")
camRight.out.link(xoutRight.input)

manipOut = pipeline.createXLinkOut()
manipOut.setStreamName("manip")
manip.out.link(manipOut.input)

nnOut = pipeline.createXLinkOut()
nnOut.setStreamName("nn")
nn.out.link(nnOut.input)

# MobilenetSSD label texts
labelMap = ["background", "aeroplane", "bicycle", "bird", "boat", "bottle", "bus", "car", "cat", "chair", "cow",
            "diningtable", "dog", "horse", "motorbike", "person", "pottedplant", "sheep", "sofa", "train", "tvmonitor"]


# Pipeline is defined, now we can connect to the device
with dai.Device(pipeline) as device:
    # Start pipeline
    device.startPipeline()

    queueSize = 8
    qRight = device.getOutputQueue("right", queueSize)
    qDisparity = device.getOutputQueue("disparity", queueSize)
    qManip = device.getOutputQueue("manip", queueSize)
    qDet = device.getOutputQueue("nn", queueSize)
    qRgbEnc = device.getOutputQueue('h265', maxSize=30, blocking=True)

    frame = None
    frameManip = None
    frameDisparity = None
    detections = []
    offsetX = (camRight.getResolutionWidth() - camRight.getResolutionHeight()) // 2
    croppedFrame = np.zeros((camRight.getResolutionHeight(), camRight.getResolutionHeight()))

    def frameNorm(frame, bbox):
        normVals = np.full(len(bbox), frame.shape[0])
        normVals[::2] = frame.shape[1]
        return (np.clip(np.array(bbox), 0, 1) * normVals).astype(int)

    videoFile = open('video.h265', 'wb')
    cv2.namedWindow("right", cv2.WINDOW_NORMAL)
    cv2.namedWindow("manip", cv2.WINDOW_NORMAL)
    cv2.namedWindow("disparity", cv2.WINDOW_NORMAL)

    while True:
```

(continues on next page)

(continued from previous page)

```
        inRight = qRight.tryGet()
        inManip = qManip.tryGet()
        inDet = qDet.tryGet()
        inDisparity = qDisparity.tryGet()

        while qRgbEnc.has():
            qRgbEnc.get().getData().tofile(videoFile)

        if inRight is not None:
            frame = cv2.flip(inRight.getCvFrame(), 1)
            if flipRectified:
                frame = cv2.flip(frame, 1)

        if inManip is not None:
            frameManip = inManip.getCvFrame()

        if inDisparity is not None:
            # Flip disparity frame, normalize it and apply color map for better visualization
            frameDisparity = inDisparity.getFrame()
            if flipRectified:
                frameDisparity = cv2.flip(frameDisparity, 1)
            frameDisparity = (frameDisparity*disparity_multiplier).astype(np.uint8)
            frameDisparity = cv2.applyColorMap(frameDisparity, cv2.COLORMAP_JET)

        if inDet is not None:
            detections = inDet.detections

        if frame is not None:
            for detection in detections:
                bbox = frameNorm(croppedFrame, (detection.xmin, detection.ymin, detection.xmax, detection.ymax))
                bbox[::2] += offsetX
                cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (255, 0, 0), 2)
                cv2.putText(frame, labelMap[detection.label], (bbox[0] + 10, bbox[1] + 20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
                cv2.putText(frame, f"{int(detection.confidence * 100)}%", (bbox[0] + 10, bbox[1] + 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
            cv2.imshow("right", frame)

        if frameDisparity is not None:
            for detection in detections:
                bbox = frameNorm(croppedFrame, (detection.xmin, detection.ymin, detection.xmax, detection.ymax))
                bbox[::2] += offsetX
                cv2.rectangle(frameDisparity, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (255, 0, 0), 2)
                cv2.putText(frameDisparity, labelMap[detection.label], (bbox[0] + 10, bbox[1] + 20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
                cv2.putText(frameDisparity, f"{int(detection.confidence * 100)}%", (bbox[0] + 10, bbox[1] + 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
            cv2.imshow("disparity", frameDisparity)

        if frameManip is not None:
            for detection in detections:
                bbox = frameNorm(frameManip, (detection.xmin, detection.ymin, detection.xmax, detection.ymax))
```

(continues on next page)

(continued from previous page)

```
                cv2.rectangle(frameManip, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (255, 0, 0), 2)
                cv2.putText(frameManip, labelMap[detection.label], (bbox[0] + 10, bbox[1] + 20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
                cv2.putText(frameManip, f"{int(detection.confidence * 100)}%", (bbox[0] + 10, bbox[1] + 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
            cv2.imshow("manip", frameManip)

        if cv2.waitKey(1) == ord('q'):
            break

    videoFile.close()

    print("To view the encoded data, convert the stream file (.h265) into a video file (.mp4) using a command below:")
    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

```
#!/usr/bin/env python3

import depthai as dai

pipeline = dai.Pipeline()

# Nodes
colorCam = pipeline.createColorCamera()
colorCam.setResolution(dai.ColorCameraProperties.SensorResolution.THE_4_K)
monoCam = pipeline.createMonoCamera()
monoCam2 = pipeline.createMonoCamera()
ve1 = pipeline.createVideoEncoder()
ve2 = pipeline.createVideoEncoder()
ve3 = pipeline.createVideoEncoder()

ve1Out = pipeline.createXLinkOut()
ve2Out = pipeline.createXLinkOut()
ve3Out = pipeline.createXLinkOut()

# Properties
monoCam.setBoardSocket(dai.CameraBoardSocket.LEFT)
monoCam2.setBoardSocket(dai.CameraBoardSocket.RIGHT)
ve1Out.setStreamName('ve1Out')
ve2Out.setStreamName('ve2Out')
ve3Out.setStreamName('ve3Out')

# Setting to 26fps will trigger error
ve1.setDefaultProfilePreset(1280, 720, 25, dai.VideoEncoderProperties.Profile.H264_
↪MAIN)
ve2.setDefaultProfilePreset(3840, 2160, 25, dai.VideoEncoderProperties.Profile.H265_
↪MAIN)
ve3.setDefaultProfilePreset(1280, 720, 25, dai.VideoEncoderProperties.Profile.H264_
↪MAIN)

# Link nodes
monoCam.out.link(ve1.input)
colorCam.video.link(ve2.input)
monoCam2.out.link(ve3.input)

ve1.bitstream.link(ve1Out.input)
ve2.bitstream.link(ve2Out.input)
ve3.bitstream.link(ve3Out.input)


# Pipeline is defined, now we can connect to the device
with dai.Device(pipeline) as dev:

    # Prepare data queues
    outQ1 = dev.getOutputQueue('ve1Out', maxSize=30, blocking=True)
    outQ2 = dev.getOutputQueue('ve2Out', maxSize=30, blocking=True)
    outQ3 = dev.getOutputQueue('ve3Out', maxSize=30, blocking=True)

    # Start the pipeline
    dev.startPipeline()
```

(continues on next page)

(continued from previous page)

```

    # Processing loop
    with open('mono1.h264', 'wb') as fileMono1H264, open('color.h265', 'wb') as fileColorH265, open('mono2.h264', 'wb') as fileMono2H264:
        print("Press Ctrl+C to stop encoding...")
        while True:
            try:
                # Empty each queue
                while outQ1.has():
                    outQ1.get().getData().tofile(fileMono1H264)

                while outQ2.has():
                    outQ2.get().getData().tofile(fileColorH265)

                while outQ3.has():
                    outQ3.get().getData().tofile(fileMono2H264)
            except KeyboardInterrupt:
                break

    print("To view the encoded data, convert the stream file (.h264/.h265) into a video file (.mp4), using commands below:")
    cmd = "ffmpeg -framerate 25 -i {} -c copy {}"
    print(cmd.format("mono1.h264", "mono1.mp4"))
    print(cmd.format("mono2.h264", "mono2.mp4"))
    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

```
#!/usr/bin/env python3

"""
This example shows usage of Camera Control message as well as ColorCamera configInput␣
↪to change crop x and y
Uses 'WASD' controls to move the crop window, 'C' to capture a still image, 'T' to␣
↪trigger autofocus, 'IOKL,.'
for manual exposure/focus:
  Control:      key[dec/inc]  min..max
  exposure time:     I   O      1..33000 [us]
  sensitivity iso:   K   L    100..1600
  focus:             ,   .      0..255 [far..near]
To go back to auto controls:
  'E' - autoexposure
  'F' - autofocus (continuous)
"""

import depthai as dai
import cv2

# Step size ('W','A','S','D' controls)
STEP_SIZE = 8
# Manual exposure/focus set step
EXP_STEP = 500  # us
ISO_STEP = 50
LENS_STEP = 3

pipeline = dai.Pipeline()

# Nodes
colorCam = pipeline.createColorCamera()
controlIn = pipeline.createXLinkIn()
configIn = pipeline.createXLinkIn()
videoEncoder = pipeline.createVideoEncoder()
stillEncoder = pipeline.createVideoEncoder()
videoMjpegOut = pipeline.createXLinkOut()
stillMjpegOut = pipeline.createXLinkOut()
previewOut = pipeline.createXLinkOut()


```

(continues on next page)

(continued from previous page)

```
# Properties
colorCam.setVideoSize(640, 360)
colorCam.setPreviewSize(300, 300)
controlIn.setStreamName('control')
configIn.setStreamName('config')
videoEncoder.setDefaultProfilePreset(colorCam.getVideoSize(), colorCam.getFps(), dai.
↪VideoEncoderProperties.Profile.MJPEG)
stillEncoder.setDefaultProfilePreset(colorCam.getStillSize(), 1, dai.
↪VideoEncoderProperties.Profile.MJPEG)
videoMjpegOut.setStreamName('video')
stillMjpegOut.setStreamName('still')
previewOut.setStreamName('preview')


# Link nodes
colorCam.video.link(videoEncoder.input)
colorCam.still.link(stillEncoder.input)
colorCam.preview.link(previewOut.input)
controlIn.out.link(colorCam.inputControl)
configIn.out.link(colorCam.inputConfig)
videoEncoder.bitstream.link(videoMjpegOut.input)
stillEncoder.bitstream.link(stillMjpegOut.input)


def clamp(num, v0, v1):
    return max(v0, min(num, v1))


# Pipeline is defined, now we can connect to the device
with dai.Device(pipeline) as dev:

    # Get data queues
    controlQueue = dev.getInputQueue('control')
    configQueue = dev.getInputQueue('config')
    previewQueue = dev.getOutputQueue('preview')
    videoQueue = dev.getOutputQueue('video')
    stillQueue = dev.getOutputQueue('still')

    # Start pipeline
    dev.startPipeline()

    # Max cropX & cropY
    maxCropX = (colorCam.getResolutionWidth() - colorCam.getVideoWidth()) / colorCam.
↪getResolutionWidth()
    maxCropY = (colorCam.getResolutionHeight() - colorCam.getVideoHeight()) /␣
↪colorCam.getResolutionHeight()

    # Default crop
    cropX = 0
    cropY = 0
    sendCamConfig = True

    # Defaults and limits for manual focus/exposure controls
    lensPos = 150
    lensMin = 0
    lensMax = 255

```

(continues on next page)

(continued from previous page)

```
    expTime = 20000
    expMin = 1
    expMax = 33000

    sensIso = 800
    sensMin = 100
    sensMax = 1600

    while True:

        previewFrames = previewQueue.tryGetAll()
        for previewFrame in previewFrames:
            cv2.imshow('preview', previewFrame.getData().reshape(previewFrame.
↪getWidth(), previewFrame.getHeight(), 3))

        videoFrames = videoQueue.tryGetAll()
        for videoFrame in videoFrames:
            # Decode JPEG
            frame = cv2.imdecode(videoFrame.getData(), cv2.IMREAD_UNCHANGED)
            # Display
            cv2.imshow('video', frame)

            # Send new cfg to camera
            if sendCamConfig:
                cfg = dai.ImageManipConfig()
                cfg.setCropRect(cropX, cropY, 0, 0)
                configQueue.send(cfg)
                print('Sending new crop - x: ', cropX, ' y: ', cropY)
                sendCamConfig = False

        stillFrames = stillQueue.tryGetAll()
        for stillFrame in stillFrames:
            # Decode JPEG
            frame = cv2.imdecode(stillFrame.getData(), cv2.IMREAD_UNCHANGED)
            # Display
            cv2.imshow('still', frame)


        # Update screen
        key = cv2.waitKey(1)
        if key == ord('q'):
            break
        elif key == ord('c'):
            ctrl = dai.CameraControl()
            ctrl.setCaptureStill(True)
            controlQueue.send(ctrl)
        elif key == ord('t'):
            print("Autofocus trigger (and disable continuous)")
            ctrl = dai.CameraControl()
            ctrl.setAutoFocusMode(dai.CameraControl.AutoFocusMode.AUTO)
            ctrl.setAutoFocusTrigger()
            controlQueue.send(ctrl)
        elif key == ord('f'):
            print("Autofocus enable, continuous")
            ctrl = dai.CameraControl()
            ctrl.setAutoFocusMode(dai.CameraControl.AutoFocusMode.CONTINUOUS_VIDEO)
            controlQueue.send(ctrl)
```

(continues on next page)

(continued from previous page)

```
        elif key == ord('e'):
            print("Autoexposure enable")
            ctrl = dai.CameraControl()
            ctrl.setAutoExposureEnable()
            controlQueue.send(ctrl)
        elif key in [ord(','), ord('.')]:
            if key == ord(','): lensPos -= LENS_STEP
            if key == ord('.'): lensPos += LENS_STEP
            lensPos = clamp(lensPos, lensMin, lensMax)
            print("Setting manual focus, lens position:", lensPos)
            ctrl = dai.CameraControl()
            ctrl.setManualFocus(lensPos)
            controlQueue.send(ctrl)
        elif key in [ord('i'), ord('o'), ord('k'), ord('l')]:
            if key == ord('i'): expTime -= EXP_STEP
            if key == ord('o'): expTime += EXP_STEP
            if key == ord('k'): sensIso -= ISO_STEP
            if key == ord('l'): sensIso += ISO_STEP
            expTime = clamp(expTime, expMin, expMax)
            sensIso = clamp(sensIso, sensMin, sensMax)
            print("Setting manual exposure, time:", expTime, "iso:", sensIso)
            ctrl = dai.CameraControl()
            ctrl.setManualExposure(expTime, sensIso)
            controlQueue.send(ctrl)
        elif key in [ord('w'), ord('a'), ord('s'), ord('d')]:
            if key == ord('a'):
                cropX = cropX - (maxCropX / colorCam.getResolutionWidth()) * STEP_SIZE
                if cropX < 0: cropX = maxCropX
            elif key == ord('d'):
                cropX = cropX + (maxCropX / colorCam.getResolutionWidth()) * STEP_SIZE
                if cropX > maxCropX: cropX = 0
            elif key == ord('w'):
                cropY = cropY - (maxCropY / colorCam.getResolutionHeight()) * STEP_
↪SIZE
                if cropY < 0: cropY = maxCropY
            elif key == ord('s'):
                cropY = cropY + (maxCropY / colorCam.getResolutionHeight()) * STEP_
↪SIZE
                if cropY > maxCropY: cropY = 0
            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

```
#!/usr/bin/env python3

"""
This example shows usage of mono camera in crop mode with the possibility to move the crop.
Uses 'WASD' controls to move the crop window, 'T' to trigger autofocus, 'IOKL,.' for manual exposure/focus:
  Control:      key[dec/inc]  min..max
  exposure time:     I   O      1..33000 [us]
  sensitivity iso:   K   L    100..1600
To go back to auto controls:
  'E' - autoexposure
"""


import cv2
import depthai as dai

# Step size ('W','A','S','D' controls)
stepSize = 0.02
# Manual exposure/focus set step
expStep = 500  # us
isoStep = 50

# Start defining a pipeline
pipeline = dai.Pipeline()

# Define a source - two mono (grayscale) camera
camRight = pipeline.createMonoCamera()
camRight.setBoardSocket(dai.CameraBoardSocket.RIGHT)
camRight.setResolution(dai.MonoCameraProperties.SensorResolution.THE_720_P)
camLeft = pipeline.createMonoCamera()
camLeft.setBoardSocket(dai.CameraBoardSocket.LEFT)
camLeft.setResolution(dai.MonoCameraProperties.SensorResolution.THE_720_P)

# Crop range
```

(continues on next page)

(continued from previous page)

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

manipRight = pipeline.createImageManip()
manipRight.initialConfig.setCropRect(topLeft.x, topLeft.y, bottomRight.x, bottomRight.y)
manipLeft = pipeline.createImageManip()
manipLeft.initialConfig.setCropRect(topLeft.x, topLeft.y, bottomRight.x, bottomRight.y)
manipRight.setMaxOutputFrameSize(camRight.getResolutionHeight()*camRight.getResolutionWidth()*3)

# Camera movement config (wasd)
configIn = pipeline.createXLinkIn()
configIn.setStreamName('config')
configIn.out.link(manipRight.inputConfig)
configIn.out.link(manipLeft.inputConfig)

# Camera control (exp, iso, focus)
controlIn = pipeline.createXLinkIn()
controlIn.setStreamName('control')
controlIn.out.link(camRight.inputControl)
controlIn.out.link(camLeft.inputControl)

# Linking with USB
camRight.out.link(manipRight.inputImage)
camLeft.out.link(manipLeft.inputImage)

# Create outputs
manipOutRight = pipeline.createXLinkOut()
manipOutRight.setStreamName("right")
manipRight.out.link(manipOutRight.input)

manipOutLeft = pipeline.createXLinkOut()
manipOutLeft.setStreamName("left")
manipLeft.out.link(manipOutLeft.input)

def clamp(num, v0, v1):
    return max(v0, min(num, v1))

# Pipeline defined, now the device is connected to
with dai.Device(pipeline) as device:
    # Start pipeline
    device.startPipeline()

    # Output queues will be used to get the grayscale frames
    qRight = device.getOutputQueue(manipOutRight.getStreamName(), maxSize=4, blocking=False)
    qLeft = device.getOutputQueue(manipOutLeft.getStreamName(), maxSize=4, blocking=False)
    configQueue = device.getInputQueue(configIn.getStreamName())
    controlQueue = device.getInputQueue(controlIn.getStreamName())

    def displayFrame(name, frame):
        cv2.imshow(name, frame)

    sendCamConfig = False
```

(continues on next page)

(continued from previous page)

```

    # Defaults and limits for manual focus/exposure controls
    expTime = 20000
    expMin = 1
    expMax = 33000

    sensIso = 800
    sensMin = 100
    sensMax = 1600

    while True:
        inRight = qRight.get()
        inLeft = qLeft.get()
        frameRight = inRight.getCvFrame()
        frameLeft = inLeft.getCvFrame()
        displayFrame("right", frameRight)
        displayFrame("left", frameLeft)

        # Update screen
        key = cv2.waitKey(1)
        if key == ord('q'):
            break
        elif key == ord('c'):
            ctrl = dai.CameraControl()
            ctrl.setCaptureStill(True)
            controlQueue.send(ctrl)
        elif key == ord('e'):
            print("Autoexposure enable")
            ctrl = dai.CameraControl()
            ctrl.setAutoExposureEnable()
            controlQueue.send(ctrl)
        elif key in [ord('i'), ord('o'), ord('k'), ord('l')]:
            if key == ord('i'): expTime -= expStep
            if key == ord('o'): expTime += expStep
            if key == ord('k'): sensIso -= isoStep
            if key == ord('l'): sensIso += isoStep
            expTime = clamp(expTime, expMin, expMax)
            sensIso = clamp(sensIso, sensMin, sensMax)
            print("Setting manual exposure, time:", expTime, "iso:", sensIso)
            ctrl = dai.CameraControl()
            ctrl.setManualExposure(expTime, sensIso)
            controlQueue.send(ctrl)
        elif key == ord('w'):
            if topLeft.y - stepSize >= 0:
                topLeft.y -= stepSize
                bottomRight.y -= stepSize
                sendCamConfig = True
        elif key == ord('a'):
            if topLeft.x - stepSize >= 0:
                topLeft.x -= stepSize
                bottomRight.x -= stepSize
                sendCamConfig = True
        elif key == ord('s'):
            if bottomRight.y + stepSize <= 1:
                topLeft.y += stepSize
                bottomRight.y += stepSize
                sendCamConfig = True
```

(continues on next page)

(continued from previous page)

```
        elif key == ord('d'):
            if bottomRight.x + stepSize <= 1:
                topLeft.x += stepSize
                bottomRight.x += stepSize
                sendCamConfig = True


        if sendCamConfig:
            cfg = dai.ImageManipConfig()
            cfg.setCropRect(topLeft.x, topLeft.y, bottomRight.x, bottomRight.y)
            configQueue.send(cfg)
            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

```
#!/usr/bin/env python3

"""
This example shows usage of depth camera in crop mode with the possibility to move␣
↪the crop.
Use 'WASD' in order to do it.
"""

import cv2
import depthai as dai
```

(continues on next page)

(continued from previous page)

```
import numpy as np

stepSize = 0.02

# Start defining a pipeline
pipeline = dai.Pipeline()

# Define a source - two mono (grayscale) cameras
left = pipeline.createMonoCamera()
left.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
left.setBoardSocket(dai.CameraBoardSocket.LEFT)

right = pipeline.createMonoCamera()
right.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
right.setBoardSocket(dai.CameraBoardSocket.RIGHT)

# Crop range
topLeft = dai.Point2f(0.4, 0.4)
bottomRight = dai.Point2f(0.6, 0.6)

manip = pipeline.createImageManip()
manip.initialConfig.setCropRect(topLeft.x, topLeft.y, bottomRight.x, bottomRight.y)
manip.setMaxOutputFrameSize(right.getResolutionHeight()*right.getResolutionWidth()*3)


# Create a node that will produce the depth map
stereo = pipeline.createStereoDepth()
stereo.setConfidenceThreshold(200)
stereo.setOutputDepth(True)

left.out.link(stereo.left)
right.out.link(stereo.right)


# Control movement
controlIn = pipeline.createXLinkIn()
controlIn.setStreamName('control')
controlIn.out.link(manip.inputConfig)

# Create outputs
xout = pipeline.createXLinkOut()
xout.setStreamName("depth")
stereo.depth.link(manip.inputImage)
manip.out.link(xout.input)

# Pipeline defined, now the device is connected to
with dai.Device(pipeline) as device:
    # Start pipeline
    device.startPipeline()

    # Output queue will be used to get the depth frames from the outputs defined above
    q = device.getOutputQueue(xout.getStreamName(), maxSize=4, blocking=False)

    sendCamConfig = False

    while True:
        inDepth = q.get()  # blocking call, will wait until a new data has arrived
```

(continues on next page)

(continued from previous page)

```
        # data is originally represented as a flat 1D array, it needs to be converted␣
↪into HxW form
        depthFrame = inDepth.getFrame()
        # frame is transformed, the color map will be applied to highlight the depth␣
↪info
        depthFrameColor = cv2.normalize(depthFrame, None, 255, 0, cv2.NORM_INF, cv2.
↪CV_8UC1)
        depthFrameColor = cv2.equalizeHist(depthFrameColor)
        depthFrameColor = cv2.applyColorMap(depthFrameColor, cv2.COLORMAP_HOT)
        controlQueue = device.getInputQueue(controlIn.getStreamName())

        # frame is ready to be shown
        cv2.imshow("depth", depthFrameColor)

        # Update screen
        key = cv2.waitKey(1)
        if key == ord('q'):
            break
        elif key == ord('w'):
            if topLeft.y - stepSize >= 0:
                topLeft.y -= stepSize
                bottomRight.y -= stepSize
                sendCamConfig = True
        elif key == ord('a'):
            if topLeft.x - stepSize >= 0:
                topLeft.x -= stepSize
                bottomRight.x -= stepSize
                sendCamConfig = True
        elif key == ord('s'):
            if bottomRight.y + stepSize <= 1:
                topLeft.y += stepSize
                bottomRight.y += stepSize
                sendCamConfig = True
        elif key == ord('d'):
            if bottomRight.x + stepSize <= 1:
                topLeft.x += stepSize
                bottomRight.x += stepSize
                sendCamConfig = True


        if sendCamConfig:
            cfg = dai.ImageManipConfig()
            cfg.setCropRect(topLeft.x, topLeft.y, bottomRight.x, bottomRight.y)
            controlQueue.send(cfg)
            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 MobilenetSDD blob (`mobilenet-ssd_openvino_2021.2_5shave.blob` file) to work - you can download it from here

Source code

Also available on GitHub

```
#!/usr/bin/env python3

from pathlib import Path
import sys
import cv2
import depthai as dai
import numpy as np

# Get argument first
nnPath = str((Path(__file__).parent / Path('models/mobilenet-ssd_openvino_2021.2_
↪5shave.blob')).resolve().absolute())
if len(sys.argv) > 1:
    nnPath = sys.argv[1]

if not Path(nnPath).exists():
    import sys
    raise FileNotFoundError(f'Required file/s not found, please run "{sys.executable}␣
↪install_requirements.py"')

# Start defining a pipeline
pipeline = dai.Pipeline()

# Define a source - color camera
camRgb = pipeline.createColorCamera()
camRgb.setPreviewSize(300, 300)    # NN input
camRgb.setResolution(dai.ColorCameraProperties.SensorResolution.THE_4_K)
camRgb.setInterleaved(False)
camRgb.setPreviewKeepAspectRatio(False)

# Define a neural network that will make predictions based on the source frames
nn = pipeline.createMobileNetDetectionNetwork()
```

(continues on next page)

(continued from previous page)

```python
nn.setConfidenceThreshold(0.5)
nn.setBlobPath(nnPath)
nn.setNumInferenceThreads(2)
nn.input.setBlocking(False)
camRgb.preview.link(nn.input)

# Create outputs
xoutVideo = pipeline.createXLinkOut()
xoutVideo.setStreamName("video")
camRgb.video.link(xoutVideo.input)

xoutPreview = pipeline.createXLinkOut()
xoutPreview.setStreamName("preview")
camRgb.preview.link(xoutPreview.input)

nnOut = pipeline.createXLinkOut()
nnOut.setStreamName("nn")
nn.out.link(nnOut.input)

# MobilenetSSD label texts
labelMap = ["background", "aeroplane", "bicycle", "bird", "boat", "bottle", "bus", "car", "cat", "chair", "cow",
            "diningtable", "dog", "horse", "motorbike", "person", "pottedplant", "sheep", "sofa", "train", "tvmonitor"]

# Pipeline is defined, now we can connect to the device
with dai.Device(pipeline) as device:
    # Start pipeline
    device.startPipeline()

    # Output queues will be used to get the frames and nn data from the outputs defined above
    qVideo = device.getOutputQueue(name="video", maxSize=4, blocking=False)
    qPreview = device.getOutputQueue(name="preview", maxSize=4, blocking=False)
    qDet = device.getOutputQueue(name="nn", maxSize=4, blocking=False)

    previewFrame = None
    videoFrame = None
    detections = []

    # nn data, being the bounding box locations, are in <0..1> range - they need to be normalized with frame width/height
    def frameNorm(frame, bbox):
        normVals = np.full(len(bbox), frame.shape[0])
        normVals[::2] = frame.shape[1]
        return (np.clip(np.array(bbox), 0, 1) * normVals).astype(int)

    def displayFrame(name, frame):
        for detection in detections:
            bbox = frameNorm(frame, (detection.xmin, detection.ymin, detection.xmax, detection.ymax))
            cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (255, 0, 0), 2)
            cv2.putText(frame, labelMap[detection.label], (bbox[0] + 10, bbox[1] + 20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
            cv2.putText(frame, f"{int(detection.confidence * 100)}%", (bbox[0] + 10, bbox[1] + 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
```

(continues on next page)

(continued from previous page)

```
        cv2.imshow(name, frame)

    cv2.namedWindow("video", cv2.WINDOW_NORMAL)
    cv2.resizeWindow("video", 1280, 720)
    print("Resize video window with mouse drag!")

    while True:
        # Instead of get (blocking), we use tryGet (nonblocking) which will return␣
↪the available data or None otherwise
        inVideo = qVideo.tryGet()
        inPreview = qPreview.tryGet()
        inDet = qDet.tryGet()

        if inVideo is not None:
            videoFrame = inVideo.getCvFrame()

        if inPreview is not None:
            previewFrame = inPreview.getCvFrame()

        if inDet is not None:
            detections = inDet.detections

        if videoFrame is not None:
            displayFrame("video", videoFrame)

        if previewFrame is not None:
            displayFrame("preview", previewFrame)

        if cv2.waitKey(1) == ord('q'):
            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

```python
#!/usr/bin/env python3

"""
 This example demonstrates use of queue events to block a thread until a message
 arrives to any (of the specified) queue
"""

import cv2
import depthai as dai

# Start defining a pipeline
pipeline = dai.Pipeline()

# Create Color and Mono cameras
camRgb = pipeline.createColorCamera()
camMono = pipeline.createMonoCamera()
# Create separate streams for them
xoutRgb = pipeline.createXLinkOut()
xoutMono = pipeline.createXLinkOut()

# Set properties
xoutRgb.setStreamName("rgb")
xoutMono.setStreamName("mono")
# Cap color camera to 5 fps
camRgb.setFps(5)
camRgb.setInterleaved(True)
camRgb.setPreviewSize(300, 300)

# Connect
camRgb.preview.link(xoutRgb.input)
camMono.out.link(xoutMono.input)


# Pipeline is defined, now we can connect to the device
with dai.Device(pipeline) as device:
    # Start pipeline
    device.startPipeline()

    # Clear queue events
    device.getQueueEvents()

    while True:
        # Block until a message arrives to any of the specified queues
        queueName = device.getQueueEvent(("rgb", "mono"))

        # Getting that message from queue with name specified by the event
        # Note: number of events doesn't necessarily match number of messages in␣
↪queues
        # because queues can be set to non-blocking (overwriting) behavior
        message = device.getOutputQueue(queueName).get()

        # Display arrived frames
        if type(message) == dai.ImgFrame:
            cv2.imshow(queueName, message.getCvFrame())
```

(continues on next page)

(continued from previous page)

```
        if cv2.waitKey(1) == ord('q'):
            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 MobilenetSDD 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

```
#!/usr/bin/env python3

from pathlib import Path
import sys
import cv2
import depthai as dai
import numpy as np
from time import monotonic

# Get argument first
parentDir = Path(__file__).parent
nnPath = str((parentDir / Path('models/mobilenet-ssd_openvino_2021.2_8shave.blob')).
↪resolve().absolute())
videoPath = str((parentDir / Path('models/construction_vest.mp4')).resolve().
↪absolute())
if len(sys.argv) > 2:
    nnPath = sys.argv[1]
    videoPath = sys.argv[2]

if not Path(nnPath).exists() or not Path(videoPath).exists():
    import sys
```

(continues on next page)

(continued from previous page)

```
    raise FileNotFoundError(f'Required file/s not found, please run "{sys.executable} install_requirements.py"')

# Start defining a pipeline
pipeline = dai.Pipeline()

# Create xLink input to which host will send frames from the video file
xinFrame = pipeline.createXLinkIn()
xinFrame.setStreamName("inFrame")

# Define a neural network that will make predictions based on the source frames
nn = pipeline.createMobileNetDetectionNetwork()
nn.setConfidenceThreshold(0.5)
nn.setBlobPath(nnPath)
nn.setNumInferenceThreads(2)
nn.input.setBlocking(False)
xinFrame.out.link(nn.input)

# Create output
nnOut = pipeline.createXLinkOut()
nnOut.setStreamName("nn")
nn.out.link(nnOut.input)

# MobilenetSSD label texts
labelMap = ["background", "aeroplane", "bicycle", "bird", "boat", "bottle", "bus", "car", "cat", "chair", "cow",
            "diningtable", "dog", "horse", "motorbike", "person", "pottedplant", "sheep", "sofa", "train", "tvmonitor"]


# Pipeline is defined, now we can connect to the device
with dai.Device(pipeline) as device:
    # Start pipeline
    device.startPipeline()

    # Input queue will be used to send video frames to the device.
    qIn = device.getInputQueue(name="inFrame")
    # Output queue will be used to get nn data from the video frames.
    qDet = device.getOutputQueue(name="nn", maxSize=4, blocking=False)

    frame = None
    detections = []

    # nn data, being the bounding box locations, are in <0..1> range - they need to be normalized with frame width/height
    def frameNorm(frame, bbox):
        normVals = np.full(len(bbox), frame.shape[0])
        normVals[::2] = frame.shape[1]
        return (np.clip(np.array(bbox), 0, 1) * normVals).astype(int)


    def to_planar(arr: np.ndarray, shape: tuple) -> np.ndarray:
        return cv2.resize(arr, shape).transpose(2, 0, 1).flatten()

    def displayFrame(name, frame):
        for detection in detections:
            bbox = frameNorm(frame, (detection.xmin, detection.ymin, detection.xmax, detection.ymax))
```

(continues on next page)

(continued from previous page)

```
            cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (255, 0, 0), 2)
            cv2.putText(frame, labelMap[detection.label], (bbox[0] + 10, bbox[1] + 20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
            cv2.putText(frame, f"{int(detection.confidence * 100)}%", (bbox[0] + 10, bbox[1] + 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
        cv2.imshow(name, frame)

    cap = cv2.VideoCapture(videoPath)
    while cap.isOpened():
        read_correctly, frame = cap.read()
        if not read_correctly:
            break

        img = dai.ImgFrame()
        img.setData(to_planar(frame, (300, 300)))
        img.setTimestamp(monotonic())
        img.setWidth(300)
        img.setHeight(300)
        qIn.send(img)

        inDet = qDet.tryGet()

        if inDet is not None:
            detections = inDet.detections

        if frame is not None:
            displayFrame("rgb", frame)

        if cv2.waitKey(1) == ord('q'):
            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

```python
#!/usr/bin/env python3

from pathlib import Path
import sys
import cv2
import depthai as dai
import numpy as np

# Get argument first
nnPath = str((Path(__file__).parent / Path('models/mobilenet-ssd_openvino_2021.2_6shave.blob')).resolve().absolute())
if len(sys.argv) > 1:
    nnPath = sys.argv[1]

if not Path(nnPath).exists():
    import sys
    raise FileNotFoundError(f'Required file/s not found, please run "{sys.executable} install_requirements.py"')

pipeline = dai.Pipeline()

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

videoEncoder = pipeline.createVideoEncoder()
videoEncoder.setDefaultProfilePreset(1920, 1080, 30, dai.VideoEncoderProperties.Profile.H265_MAIN)
cam.video.link(videoEncoder.input)

nn = pipeline.createMobileNetDetectionNetwork()
nn.setConfidenceThreshold(0.5)
nn.setBlobPath(nnPath)
nn.setNumInferenceThreads(2)
nn.input.setBlocking(False)
cam.preview.link(nn.input)
```

(continues on next page)

(continued from previous page)

```

videoOut = pipeline.createXLinkOut()
videoOut.setStreamName('h265')
videoEncoder.bitstream.link(videoOut.input)

xoutRgb = pipeline.createXLinkOut()
xoutRgb.setStreamName("rgb")
cam.preview.link(xoutRgb.input)

nnOut = pipeline.createXLinkOut()
nnOut.setStreamName("nn")
nn.out.link(nnOut.input)

# MobilenetSSD label texts
labelMap = ["background", "aeroplane", "bicycle", "bird", "boat", "bottle", "bus", "car", "cat", "chair", "cow",
            "diningtable", "dog", "horse", "motorbike", "person", "pottedplant", "sheep", "sofa", "train", "tvmonitor"]

with dai.Device(pipeline) as device, open('video.h265', 'wb') as videoFile:
    device.startPipeline()

    queue_size = 8
    qRgb = device.getOutputQueue("rgb", queue_size)
    qDet = device.getOutputQueue("nn", queue_size)
    qRgbEnc = device.getOutputQueue('h265', maxSize=30, blocking=True)

    frame = None
    detections = []


    def frameNorm(frame, bbox):
        normVals = np.full(len(bbox), frame.shape[0])
        normVals[::2] = frame.shape[1]
        return (np.clip(np.array(bbox), 0, 1) * normVals).astype(int)

    def displayFrame(name, frame):
        for detection in detections:
            bbox = frameNorm(frame, (detection.xmin, detection.ymin, detection.xmax, detection.ymax))
            cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (255, 0, 0), 2)
            cv2.putText(frame, labelMap[detection.label], (bbox[0] + 10, bbox[1] + 20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
            cv2.putText(frame, f"{int(detection.confidence * 100)}%", (bbox[0] + 10, bbox[1] + 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
        cv2.imshow(name, frame)


    while True:
        inRgb = qRgb.tryGet()
        inDet = qDet.tryGet()

        while qRgbEnc.has():
            qRgbEnc.get().getData().tofile(videoFile)

        if inRgb is not None:
```

(continues on next page)

(continued from previous page)

```
            frame = inRgb.getCvFrame()

        if inDet is not None:
            detections = inDet.detections

        if frame is not None:
            displayFrame("rgb", frame)

        if cv2.waitKey(1) == ord('q'):
            break

print("To view the encoded data, convert the stream file (.h265) into a video file (.
↪mp4) using a command below:")
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

```
#!/usr/bin/env python3

"""
Mono camera control demo:
  Control:      key[dec/inc]  min..max
  exposure time:     I   O      1..33000 [us]
  sensitivity iso:   K   L    100..1600
Back to autoexposure: 'E'
"""

import cv2
import depthai as dai

# Start defining a pipeline
pipeline = dai.Pipeline()

# Define a source - two mono (grayscale) cameras
```

(continues on next page)

(continued from previous page)

```
camLeft = pipeline.createMonoCamera()
camLeft.setBoardSocket(dai.CameraBoardSocket.LEFT)
camLeft.setResolution(dai.MonoCameraProperties.SensorResolution.THE_720_P)

camRight = pipeline.createMonoCamera()
camRight.setBoardSocket(dai.CameraBoardSocket.RIGHT)
camRight.setResolution(dai.MonoCameraProperties.SensorResolution.THE_720_P)

# Create outputs
xoutLeft = pipeline.createXLinkOut()
xoutLeft.setStreamName('left')
camLeft.out.link(xoutLeft.input)
xoutRight = pipeline.createXLinkOut()
xoutRight.setStreamName('right')
camRight.out.link(xoutRight.input)

# Create and link control input
control_in = pipeline.createXLinkIn()
control_in.setStreamName('control')
control_in.out.link(camLeft.inputControl)
control_in.out.link(camRight.inputControl)


def clamp(num, v0, v1): return max(v0, min(num, v1))


# Pipeline is defined, now we can connect to the device
with dai.Device(pipeline) as device:
    # Start pipeline
    device.startPipeline()

    # Output queues will be used to get the grayscale frames from the outputs defined above
    qLeft = device.getOutputQueue(name="left", maxSize=4, blocking=False)
    qRight = device.getOutputQueue(name="right", maxSize=4, blocking=False)
    # Input queue for control commands
    controlQueue = device.getInputQueue('control')

    # Manual exposure: key-press step, defaults and limits
    EXP_STEP = 500  # us
    ISO_STEP = 50

    exp_time = 20000
    expMin = 1
    expMax = 33000

    sensIso = 800
    sensMin = 100
    sensMax = 1600

    frameLeft = None
    frameRight = None

    while True:
        # Instead of get (blocking), we use tryGet (nonblocking) which will return the available data or None otherwise
        inLeft = qLeft.tryGet()
```

(continues on next page)

(continued from previous page)

```
        inRight = qRight.tryGet()

        if inLeft is not None:
            frameLeft = inLeft.getCvFrame()

        if inRight is not None:
            frameRight = inRight.getCvFrame()

        # show the frames if available
        if frameLeft is not None:
            cv2.imshow("left", frameLeft)
        if frameRight is not None:
            cv2.imshow("right", frameRight)

        key = cv2.waitKey(1)
        if key == ord('q'):
            break
        elif key in [ord('i'), ord('o'), ord('k'), ord('l')]:
            if key == ord('i'):
                exp_time -= EXP_STEP
            if key == ord('o'):
                exp_time += EXP_STEP
            if key == ord('k'):
                sensIso -= ISO_STEP
            if key == ord('l'):
                sensIso += ISO_STEP
            exp_time = clamp(exp_time, expMin, expMax)
            sensIso = clamp(sensIso, sensMin, sensMax)
            print("Setting manual exposure, time:", exp_time, "iso:", sensIso)
            ctrl = dai.CameraControl()
            ctrl.setManualExposure(exp_time, sensIso)
            controlQueue.send(ctrl)
        elif key == ord('e'):
            print("Autoexposure enable")
            ctrl = dai.CameraControl()
            ctrl.setAutoExposureEnable()
            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

```
#!/usr/bin/env python3

"""
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.
"""

import depthai as dai
import cv2
import numpy as np

keyRotateDecr = 'z'
keyRotateIncr = 'x'
keyResizeInc = 'v'
keyWarpTestCycle = 'c'


def printControls():
    print("=== Controls:")
    print(keyRotateDecr, "-rotated rectangle crop, decrease rate")
    print(keyRotateIncr, "-rotated rectangle crop, increase rate")
    print(keyWarpTestCycle, "-warp 4-point transform, cycle through modes")
    print(keyResizeInc, "-resize cropped region, or disable resize")
    print("h -print controls (help)")


rotateRateMax = 5.0
rotateRateInc = 0.1

resizeMaxW = 800
resizeMaxH = 600
resizeFactorMax = 5

'''
The crop points are specified in clockwise order,
with first point mapped to output top-left, as:
    P0  ->  P1
     ^       v
    P3  <-  P2
'''
P0 = [0, 0]  # top-left
P1 = [1, 0]  # top-right
P2 = [1, 1]  # bottom-right
```

(continues on next page)

(continued from previous page)

```
P3 = [0, 1]  # bottom-left
warpTestList = [
    # points order, normalized cordinates, description
    # [[[0, 0], [1, 0], [1, 1], [0, 1]], True, "passthrough"],
    # [[[0, 0], [639, 0], [639, 479], [0, 479]], False, "passthrough (pixels)"],
    [[P0, P1, P2, P3], True, "1.passthrough"],
    [[P3, P0, P1, P2], True, "2.rotate 90"],
    [[P2, P3, P0, P1], True, "3.rotate 180"],
    [[P1, P2, P3, P0], True, "4.rotate 270"],
    [[P1, P0, P3, P2], True, "5.horizontal mirror"],
    [[P3, P2, P1, P0], True, "6.vertical flip"],
    [[[-0.1, -0.1], [1.1, -0.1], [1.1, 1.1], [-0.1, 1.1]], True, "7.add black borders
↪"],
    [[[-0.3, 0], [1, 0], [1.3, 1], [0, 1]], True, "8.parallelogram transform"],
    [[[-0.2, 0], [1.8, 0], [1, 1], [0, 1]], True, "9.trapezoid transform"],
]

pipeline = dai.Pipeline()

cam = pipeline.createColorCamera()
cam.setPreviewSize(640, 480)
cam.setInterleaved(False)
camOut = pipeline.createXLinkOut()
camOut.setStreamName("preview")

manip = pipeline.createImageManip()
manip.setMaxOutputFrameSize(2000*1500*3)
manipOut = pipeline.createXLinkOut()
manipOut.setStreamName("manip")
manipCfg = pipeline.createXLinkIn()
manipCfg.setStreamName("manipCfg")

cam.preview.link(camOut.input)
cam.preview.link(manip.inputImage)
manip.out.link(manipOut.input)
manipCfg.out.link(manip.inputConfig)

with dai.Device(pipeline) as device:
    device.startPipeline()

    qPreview = device.getOutputQueue(name="preview", maxSize=4)
    qManip = device.getOutputQueue(name="manip", maxSize=4)
    qManipCfg = device.getInputQueue(name="manipCfg")

    key = -1
    angleDeg = 0
    rotateRate = 1.0
    resizeFactor = 0
    resizeX = 0
    resizeY = 0
    testFourPt = False
    warpIdx = -1

    printControls()

    while key != ord('q'):
        if key > 0:
```

(continues on next page)

(continued from previous page)

```
            if key == ord(keyRotateDecr) or key == ord(keyRotateIncr):
                if key == ord(keyRotateDecr):
                    if rotateRate > -rotateRateMax:
                        rotateRate -= rotateRateInc
                if key == ord(keyRotateIncr):
                    if rotateRate < rotateRateMax:
                        rotateRate += rotateRateInc
                testFourPt = False
                print("Crop rotated rectangle, rate per frame: {:.1f} degrees.".format(rotateRate))
            elif key == ord(keyResizeInc):
                resizeFactor += 1
                if resizeFactor > resizeFactorMax:
                    resizeFactor = 0
                    print("Crop region not resized")
                else:
                    resizeX = resizeMaxW // resizeFactor
                    resizeY = resizeMaxH // resizeFactor
                    print("Crop region resized to", resizeX, 'x', resizeY)
            elif key == ord(keyWarpTestCycle):
                # Disable resizing initially
                resizeFactor = 0
                warpIdx = (warpIdx + 1) % len(warpTestList)
                testFourPt = True
                testDescription = warpTestList[warpIdx][2]
                print("Warp 4-point transform:", testDescription)
            elif key == ord('h'):
                printControls()

        # Send an updated config with continuous rotate, or after a key press
        if key >= 0 or (not testFourPt and abs(rotateRate) > 0.0001):
            cfg = dai.ImageManipConfig()
            if testFourPt:
                test = warpTestList[warpIdx]
                points, normalized = test[0], test[1]
                # TODO: improve this, should avoid this conversion
                point2fList = []
                for p in points:
                    pt = dai.Point2f()
                    pt.x, pt.y = p[0], p[1]
                    point2fList.append(pt)
                cfg.setWarpTransformFourPoints(point2fList, normalized)
            else:
                angleDeg += rotateRate
                rotatedRect = ((320, 240), (400, 400), angleDeg)
                rr = dai.RotatedRect()
                rr.center.x, rr.center.y = rotatedRect[0]
                rr.size.width, rr.size.height = rotatedRect[1]
                rr.angle = rotatedRect[2]
                cfg.setCropRotatedRect(rr, False)
            if resizeFactor > 0:
                cfg.setResize(resizeX, resizeY)
            # cfg.setWarpBorderFillColor(0, 0, 255)
            # cfg.setWarpBorderReplicatePixels()
            qManipCfg.send(cfg)

        for q in [qPreview, qManip]:
```

(continues on next page)

(continued from previous page)

```
            pkt = q.get()
            name = q.getName()
            shape = (3, pkt.getHeight(), pkt.getWidth())
            frame = pkt.getCvFrame()
            if name == "preview" and not testFourPt:
                # Draw RotatedRect cropped area on input frame
                points = np.int0(cv2.boxPoints(rotatedRect))
                cv2.drawContours(frame, [points], 0, (255, 0, 0), 1)
                # Mark top-left corner
                cv2.circle(frame, tuple(points[1]), 10, (255, 0, 0), 2)
            cv2.imshow(name, frame)
        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

```
#!/usr/bin/env python3

"""
Tiny-yolo-v3 device side decoding demo
  YOLO v3 Tiny is a real-time object detection model implemented with Keras* from
  this repository <https://github.com/david8862/keras-YOLOv3-model-set> and converted
  to TensorFlow* framework. This model was pretrained on COCO* dataset with 80 classes.
"""
```

(continues on next page)

(continued from previous page)

```

from pathlib import Path
import sys
import cv2
import depthai as dai
import numpy as np
import time

# Tiny yolo v3 label texts
labelMap = [
    "person",         "bicycle",    "car",           "motorbike",     "aeroplane",
↪"bus",           "train",
    "truck",          "boat",       "traffic light", "fire hydrant",  "stop sign",
↪"parking meter", "bench",
    "bird",           "cat",        "dog",           "horse",         "sheep",
↪"cow",           "elephant",
    "bear",           "zebra",      "giraffe",       "backpack",      "umbrella",
↪"handbag",       "tie",
    "suitcase",       "frisbee",    "skis",          "snowboard",     "sports ball",
↪"kite",          "baseball bat",
    "baseball glove", "skateboard", "surfboard",     "tennis racket", "bottle",
↪"wine glass",    "cup",
    "fork",           "knife",      "spoon",         "bowl",          "banana",
↪"apple",         "sandwich",
    "orange",         "broccoli",   "carrot",        "hot dog",       "pizza",
↪"donut",         "cake",
    "chair",          "sofa",       "pottedplant",   "bed",           "diningtable",
↪"toilet",        "tvmonitor",
    "laptop",         "mouse",      "remote",        "keyboard",      "cell phone",
↪"microwave",     "oven",
    "toaster",        "sink",       "refrigerator",  "book",          "clock",
↪"vase",          "scissors",
    "teddy bear",     "hair drier", "toothbrush"
]


syncNN = True

# Get argument first
nnPath = str((Path(__file__).parent / Path('models/tiny-yolo-v3_openvino_2021.2_
↪6shave.blob')).resolve().absolute())
if len(sys.argv) > 1:
    nnPath = sys.argv[1]

if not Path(nnPath).exists():
    import sys
    raise FileNotFoundError(f'Required file/s not found, please run "{sys.executable}␣
↪install_requirements.py"')

# Start defining a pipeline
pipeline = dai.Pipeline()

# Define a source - color camera
camRgb = pipeline.createColorCamera()
camRgb.setPreviewSize(416, 416)
camRgb.setInterleaved(False)
camRgb.setFps(40)
```

(continues on next page)

(continued from previous page)

```python

# network specific settings
detectionNetwork = pipeline.createYoloDetectionNetwork()
detectionNetwork.setConfidenceThreshold(0.5)
detectionNetwork.setNumClasses(80)
detectionNetwork.setCoordinateSize(4)
detectionNetwork.setAnchors(np.array([10,14, 23,27, 37,58, 81,82, 135,169, 344,319]))
detectionNetwork.setAnchorMasks({"side26": np.array([1, 2, 3]), "side13": np.array([3, 4, 5])})
detectionNetwork.setIouThreshold(0.5)

detectionNetwork.setBlobPath(nnPath)
detectionNetwork.setNumInferenceThreads(2)
detectionNetwork.input.setBlocking(False)

camRgb.preview.link(detectionNetwork.input)

# Create outputs
xoutRgb = pipeline.createXLinkOut()
xoutRgb.setStreamName("rgb")
if syncNN:
    detectionNetwork.passthrough.link(xoutRgb.input)
else:
    camRgb.preview.link(xoutRgb.input)

nnOut = pipeline.createXLinkOut()
nnOut.setStreamName("detections")
detectionNetwork.out.link(nnOut.input)


# Pipeline is defined, now we can connect to the device
with dai.Device(pipeline) as device:
    # Start pipeline
    device.startPipeline()

    # Output queues will be used to get the rgb frames and nn data from the outputs defined above
    qRgb = device.getOutputQueue(name="rgb", maxSize=4, blocking=False)
    qDet = device.getOutputQueue(name="detections", maxSize=4, blocking=False)

    frame = None
    detections = []

    # nn data, being the bounding box locations, are in <0..1> range - they need to be normalized with frame width/height
    def frameNorm(frame, bbox):
        normVals = np.full(len(bbox), frame.shape[0])
        normVals[::2] = frame.shape[1]
        return (np.clip(np.array(bbox), 0, 1) * normVals).astype(int)

    def displayFrame(name, frame):
        for detection in detections:
            bbox = frameNorm(frame, (detection.xmin, detection.ymin, detection.xmax, detection.ymax))
            cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (255, 0, 0), 2)
            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)

```
            cv2.putText(frame, f"{int(detection.confidence * 100)}%", (bbox[0] + 10, bbox[1] + 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
        cv2.imshow(name, frame)

    startTime = time.monotonic()
    counter = 0

    while True:
        if syncNN:
            inRgb = qRgb.get()
            inDet = qDet.get()
        else:
            inRgb = qRgb.tryGet()
            inDet = qDet.tryGet()

        if inRgb is not None:
            frame = inRgb.getCvFrame()
            cv2.putText(frame, "NN fps: {:.2f}".format(counter / (time.monotonic() - startTime)),
                        (2, frame.shape[0] - 4), cv2.FONT_HERSHEY_TRIPLEX, 0.4, color=(255, 255, 255))

        if inDet is not None:
            detections = inDet.detections
            counter += 1

        if frame is not None:
            displayFrame("rgb", frame)

        if cv2.waitKey(1) == ord('q'):
            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

```
#!/usr/bin/env python3

"""
Tiny-yolo-v4 device side decoding demo
The code is the same as for Tiny-yolo-V3, the only difference is the blob file.
The blob was compiled following this tutorial: https://github.com/TNTWEN/OpenVINO-YOLOV4
"""

from pathlib import Path
import sys
import cv2
import depthai as dai
import numpy as np
import time

# tiny yolo v4 label texts
labelMap = [
    "person",         "bicycle",    "car",           "motorbike",     "aeroplane",   "bus",           "train",
    "truck",          "boat",       "traffic light", "fire hydrant",  "stop sign",   "parking meter", "bench",
    "bird",           "cat",        "dog",           "horse",         "sheep",       "cow",           "elephant",
    "bear",           "zebra",      "giraffe",       "backpack",      "umbrella",    "handbag",       "tie",
    "suitcase",       "frisbee",    "skis",          "snowboard",     "sports ball", "kite",          "baseball bat",
    "baseball glove", "skateboard", "surfboard",     "tennis racket", "bottle",      "wine glass",    "cup",
    "fork",           "knife",      "spoon",         "bowl",          "banana",      "apple",         "sandwich",
    "orange",         "broccoli",   "carrot",        "hot dog",       "pizza",       "donut",         "cake",
    "chair",          "sofa",       "pottedplant",   "bed",           "diningtable", "toilet",        "tvmonitor",
    "laptop",         "mouse",      "remote",        "keyboard",      "cell phone",  "microwave",     "oven",
    "toaster",        "sink",       "refrigerator",  "book",          "clock",       "vase",          "scissors",
    "teddy bear",     "hair drier", "toothbrush"
]


syncNN = True

# Get argument first
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)

```
if len(sys.argv) > 1:
    nnPath = sys.argv[1]

if not Path(nnPath).exists():
    import sys
    raise FileNotFoundError(f'Required file/s not found, please run "{sys.executable} install_requirements.py"')

# Start defining a pipeline
pipeline = dai.Pipeline()

# Define a source - color camera
camRgb = pipeline.createColorCamera()
camRgb.setPreviewSize(416, 416)
camRgb.setInterleaved(False)
camRgb.setFps(40)

# Network specific settings
detectionNetwork = pipeline.createYoloDetectionNetwork()
detectionNetwork.setConfidenceThreshold(0.5)
detectionNetwork.setNumClasses(80)
detectionNetwork.setCoordinateSize(4)
detectionNetwork.setAnchors(np.array([10,14, 23,27, 37,58, 81,82, 135,169, 344,319]))
detectionNetwork.setAnchorMasks({"side26": np.array([1, 2, 3]), "side13": np.array([3, 4, 5])})
detectionNetwork.setIouThreshold(0.5)

detectionNetwork.setBlobPath(nnPath)
detectionNetwork.setNumInferenceThreads(2)
detectionNetwork.input.setBlocking(False)

camRgb.preview.link(detectionNetwork.input)

# Create outputs
xoutRgb = pipeline.createXLinkOut()
xoutRgb.setStreamName("rgb")
if syncNN:
    detectionNetwork.passthrough.link(xoutRgb.input)
else:
    camRgb.preview.link(xoutRgb.input)

nnOut = pipeline.createXLinkOut()
nnOut.setStreamName("detections")
detectionNetwork.out.link(nnOut.input)


# Pipeline is defined, now we can connect to the device
with dai.Device(pipeline) as device:
    # Start pipeline
    device.startPipeline()

    # Output queues will be used to get the rgb frames and nn data from the outputs defined above
    qRgb = device.getOutputQueue(name="rgb", maxSize=4, blocking=False)
    qDet = device.getOutputQueue(name="detections", maxSize=4, blocking=False)

    frame = None
```

(continues on next page)

(continued from previous page)

```
    detections = []

    # nn data, being the bounding box locations, are in <0..1> range - they need to be normalized with frame width/height
    def frameNorm(frame, bbox):
        normVals = np.full(len(bbox), frame.shape[0])
        normVals[::2] = frame.shape[1]
        return (np.clip(np.array(bbox), 0, 1) * normVals).astype(int)

    def displayFrame(name, frame):
        for detection in detections:
            bbox = frameNorm(frame, (detection.xmin, detection.ymin, detection.xmax, detection.ymax))
            cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (255, 0, 0), 2)
            cv2.putText(frame, labelMap[detection.label], (bbox[0] + 10, bbox[1] + 20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
            cv2.putText(frame, f"{int(detection.confidence * 100)}%", (bbox[0] + 10, bbox[1] + 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
        cv2.imshow(name, frame)

    startTime = time.monotonic()
    counter = 0

    while True:
        if syncNN:
            inRgb = qRgb.get()
            inDet = qDet.get()
        else:
            inRgb = qRgb.tryGet()
            inDet = qDet.tryGet()

        if inRgb is not None:
            frame = inRgb.getCvFrame()
            cv2.putText(frame, "NN fps: {:.2f}".format(counter / (time.monotonic() - startTime)),
                        (2, frame.shape[0] - 4), cv2.FONT_HERSHEY_TRIPLEX, 0.4, color=(255, 255, 255))

        if inDet is not None:
            detections = inDet.detections
            counter += 1

        if frame is not None:
            displayFrame("rgb", frame)

        if cv2.waitKey(1) == ord('q'):
            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

```
#!/usr/bin/env python3

from pathlib import Path
import sys
import cv2
import depthai as dai
import numpy as np

# Press WASD to move a manual ROI window for auto-exposure control.
# Press N to go back to the region controlled by the NN detections.

# Get argument first
nnPath = str((Path(__file__).parent / Path('models/mobilenet-ssd_openvino_2021.2_
↪6shave.blob')).resolve().absolute())
if len(sys.argv) > 1:
    nnPath = sys.argv[1]

```

(continues on next page)

(continued from previous page)

```
if not Path(nnPath).exists():
    import sys
    raise FileNotFoundError(f'Required file/s not found, please run "{sys.executable}␣
↪install_requirements.py"')

previewSize = (300, 300)

# Start defining a pipeline
pipeline = dai.Pipeline()

# Define a source - color camera
camRgb = pipeline.createColorCamera()
camRgb.setPreviewSize(*previewSize)
camRgb.setInterleaved(False)

camControlIn = pipeline.createXLinkIn()
camControlIn.setStreamName('camControl')
camControlIn.out.link(camRgb.inputControl)

# Define a neural network that will make predictions based on the source frames
nn = pipeline.createMobileNetDetectionNetwork()
nn.setConfidenceThreshold(0.5)
nn.setBlobPath(nnPath)
nn.setNumInferenceThreads(2)
nn.input.setBlocking(False)
camRgb.preview.link(nn.input)

# Create outputs
xoutRgb = pipeline.createXLinkOut()
xoutRgb.setStreamName("rgb")
camRgb.preview.link(xoutRgb.input)

nnOut = pipeline.createXLinkOut()
nnOut.setStreamName("nn")
nn.out.link(nnOut.input)

# MobilenetSSD label texts
labelMap = ["background", "aeroplane", "bicycle", "bird", "boat", "bottle", "bus",
↪"car", "cat", "chair", "cow",
            "diningtable", "dog", "horse", "motorbike", "person", "pottedplant",
↪"sheep", "sofa", "train", "tvmonitor"]


def clamp(num, v0, v1):
    return max(v0, min(num, v1))


def asControl(roi):
    camControl = dai.CameraControl()
    camControl.setAutoExposureRegion(*roi)
    return camControl


class AutoExposureRegion:
    step = 10
    position = (0, 0)
    size = (100, 100)
```

(continues on next page)

(continued from previous page)

```
    resolution = camRgb.getResolutionSize()
    maxDims = previewSize[0], previewSize[1]

    def grow(self, x=0, y=0):
        self.size = (
            clamp(x + self.size[0], 1, self.maxDims[0]),
            clamp(y + self.size[1], 1, self.maxDims[1])
        )

    def move(self, x=0, y=0):
        self.position = (
            clamp(x + self.position[0], 0, self.maxDims[0]),
            clamp(y + self.position[1], 0, self.maxDims[1])
        )

    def endPosition(self):
        return (
            clamp(self.position[0] + self.size[0], 0, self.maxDims[0]),
            clamp(self.position[1] + self.size[1], 0, self.maxDims[1]),
        )

    def toRoi(self):
        roi = np.array([*self.position, *self.size])
        # Convert to absolute camera coordinates
        roi = roi * self.resolution[1] // 300
        roi[0] += (self.resolution[0] - self.resolution[1]) // 2  # x offset for device crop
        return roi

    @staticmethod
    def bboxToRoi(bbox):
        startX, startY = bbox[:2]
        width, height = bbox[2] - startX, bbox[3] - startY
        roi = frameNorm(np.empty(camRgb.getResolutionSize()), (startX, startY, width, height))
        return roi


# Pipeline is defined, now we can connect to the device
with dai.Device(pipeline) as device:
    # Start pipeline
    device.startPipeline()

    # Output queues will be used to get the rgb frames and nn data from the outputs defined above
    qControl = device.getInputQueue(name="camControl")
    qRgb = device.getOutputQueue(name="rgb", maxSize=4, blocking=False)
    qDet = device.getOutputQueue(name="nn", maxSize=4, blocking=False)
    frame = None
    detections = []

    nnRegion = True
    region = AutoExposureRegion()

    # nn data (bounding box locations) are in <0..1> range - they need to be normalized with frame width/height
    def frameNorm(frame, bbox):
```

(continues on next page)

(continued from previous page)

```
        normVals = np.full(len(bbox), frame.shape[0])
        normVals[::2] = frame.shape[1]
        return (np.clip(np.array(bbox), 0, 1) * normVals).astype(int)

    def displayFrame(name, frame):
        for detection in detections:
            bbox = frameNorm(frame, (detection.xmin, detection.ymin, detection.xmax, detection.ymax))
            cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (255, 0, 0), 2)
            cv2.putText(frame, labelMap[detection.label], (bbox[0] + 10, bbox[1] + 20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
            cv2.putText(frame, f"{int(detection.confidence * 100)}%", (bbox[0] + 10, bbox[1] + 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
        if not nnRegion:
            cv2.rectangle(frame, region.position, region.endPosition(), (0, 255, 0), 2)
        cv2.imshow(name, frame)

    while True:
        # Instead of get (blocking), we use tryGet (nonblocking) which will return the available data or None otherwise
        inRgb = qRgb.tryGet()
        inDet = qDet.tryGet()

        if inRgb is not None:
            frame = inRgb.getCvFrame()

        if inDet is not None:
            detections = inDet.detections

            if nnRegion and len(detections) > 0:
                bbox = (detections[0].xmin, detections[0].ymin, detections[0].xmax, detections[0].ymax)
                qControl.send(asControl(AutoExposureRegion.bboxToRoi(bbox)))

        if frame is not None:
            displayFrame("rgb", frame)

        key = cv2.waitKey(1)
        if key == ord('n'):
            print("AE ROI controlled by NN")
            nnRegion = True
        elif key in [ord('w'), ord('a'), ord('s'), ord('d'), ord('+'), ord('-')]:
            nnRegion = False
            if key == ord('a'):
                region.move(x=-region.step)
            if key == ord('d'):
                region.move(x=region.step)
            if key == ord('w'):
                region.move(y=-region.step)
            if key == ord('s'):
                region.move(y=region.step)
            if key == ord('+'):
                region.grow(x=10, y=10)
                region.step = region.step + 1
            if key == ord('-'):
```

(continues on next page)

(continued from previous page)

```
                region.grow(x=-10, y=-10)
                region.step = max(region.step - 1, 1)
            print(f"Setting static AE ROI: {region.toRoi()} (on frame: {[*region.
↪position, *region.endPosition()]})")
            qControl.send(asControl(region.toRoi()))
        elif key == ord('q'):
            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 eaiser 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

```
#!/usr/bin/env python3

import cv2
import depthai as dai

# Start defining a pipeline
pipeline = dai.Pipeline()

# Define a source - color camera
camRgb = pipeline.createColorCamera()
camRgb.setPreviewSize(300, 300)
camRgb.setBoardSocket(dai.CameraBoardSocket.RGB)
camRgb.setResolution(dai.ColorCameraProperties.SensorResolution.THE_1080_P)
camRgb.setInterleaved(True)
camRgb.setColorOrder(dai.ColorCameraProperties.ColorOrder.BGR)

# Create output
xoutVideo = pipeline.createXLinkOut()
xoutVideo.setStreamName("video")
xoutPreview = pipeline.createXLinkOut()
xoutPreview.setStreamName("preview")
```

(continues on next page)

(continued from previous page)

```

camRgb.preview.link(xoutPreview.input)
camRgb.video.link(xoutVideo.input)

# Pipeline is defined, now we can connect to the device
with dai.Device(pipeline) as device:
    # Start pipeline
    device.startPipeline()

    while True:
        # Get preview and video frames
        preview = device.getOutputQueue('preview').get()
        video = device.getOutputQueue('video').get()

        # Show 'preview' frame as is (already in correct format, no copy is made)
        cv2.imshow("preview", preview.getFrame())
        # Get BGR frame from NV12 encoded video frame to show with opencv
        cv2.imshow("video", video.getCvFrame())

        if cv2.waitKey(1) == ord('q'):
            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

```python
#!/usr/bin/env python3

import cv2
import depthai as dai


def print_sys_info(info):
    m = 1024 * 1024 # MiB
    print(f"Drr used / total - {info.ddrMemoryUsage.used / m:.2f} / {info.ddrMemoryUsage.total / m:.2f} MiB")
    print(f"Cmx used / total - {info.cmxMemoryUsage.used / m:.2f} / {info.cmxMemoryUsage.total / m:.2f} MiB")
    print(f"LeonCss heap used / total - {info.leonCssMemoryUsage.used / m:.2f} / {info.leonCssMemoryUsage.total / m:.2f} MiB")
    print(f"LeonMss heap used / total - {info.leonMssMemoryUsage.used / m:.2f} / {info.leonMssMemoryUsage.total / m:.2f} MiB")
    t = info.chipTemperature
    print(f"Chip temperature - average: {t.average:.2f}, css: {t.css:.2f}, mss: {t.mss:.2f}, upa0: {t.upa:.2f}, upa1: {t.dss:.2f}")
    print(f"Cpu usage - Leon OS: {info.leonCssCpuUsage.average * 100:.2f}%, Leon RT: {info.leonMssCpuUsage.average * 100:.2f} %")
    print("----------------------------------------")


# Start defining a pipeline
pipeline = dai.Pipeline()

sys_logger = pipeline.createSystemLogger()
sys_logger.setRate(1)  # 1 Hz

# Create output
linkOut = pipeline.createXLinkOut()
linkOut.setStreamName("sysinfo")
```

(continues on next page)

(continued from previous page)

```
sys_logger.out.link(linkOut.input)

# Pipeline is defined, now we can connect to the device
with dai.Device(pipeline) as device:
    # Start pipeline
    device.startPipeline()

    # Output queue will be used to get the system info
    q_sysinfo = device.getOutputQueue(name="sysinfo", maxSize=4, blocking=False)

    while True:
        info = q_sysinfo.get()  # Blocking call, will wait until a new data has␣
↪arrived
        print_sys_info(info)

        if cv2.waitKey(1) == ord('q'):
            break

```

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 MobilenetSDD blob (`mobilenet-ssd_openvino_2021.2_6shave.blob` file) to work - you can download it from here

Source code

Also available on GitHub

```python
#!/usr/bin/env python3

from pathlib import Path
import sys
import cv2
import depthai as dai
import numpy as np
import time

'''
Spatial detection network demo.
    Performs inference on RGB camera and retrieves spatial location coordinates: x,y,z relative to the center of depth map.
'''

# MobilenetSSD label texts
labelMap = ["background", "aeroplane", "bicycle", "bird", "boat", "bottle", "bus", "car", "cat", "chair", "cow",
            "diningtable", "dog", "horse", "motorbike", "person", "pottedplant", "sheep", "sofa", "train", "tvmonitor"]

syncNN = True

# Get argument first
nnBlobPath = str((Path(__file__).parent / Path('models/mobilenet-ssd_openvino_2021.2_6shave.blob')).resolve().absolute())
if len(sys.argv) > 1:
    nnBlobPath = sys.argv[1]

if not Path(nnBlobPath).exists():
    import sys
    raise FileNotFoundError(f'Required file/s not found, please run "{sys.executable} install_requirements.py"')

# Start defining a pipeline
pipeline = dai.Pipeline()

# Define a source - color camera
colorCam = pipeline.createColorCamera()
spatialDetectionNetwork = pipeline.createMobileNetSpatialDetectionNetwork()
monoLeft = pipeline.createMonoCamera()
monoRight = pipeline.createMonoCamera()
stereo = pipeline.createStereoDepth()

xoutRgb = pipeline.createXLinkOut()
xoutNN = pipeline.createXLinkOut()
xoutBoundingBoxDepthMapping = pipeline.createXLinkOut()
xoutDepth = pipeline.createXLinkOut()

xoutRgb.setStreamName("rgb")
xoutNN.setStreamName("detections")
xoutBoundingBoxDepthMapping.setStreamName("boundingBoxDepthMapping")
xoutDepth.setStreamName("depth")

```

(continues on next page)

(continued from previous page)

```

colorCam.setPreviewSize(300, 300)
colorCam.setResolution(dai.ColorCameraProperties.SensorResolution.THE_1080_P)
colorCam.setInterleaved(False)
colorCam.setColorOrder(dai.ColorCameraProperties.ColorOrder.BGR)

monoLeft.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
monoLeft.setBoardSocket(dai.CameraBoardSocket.LEFT)
monoRight.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
monoRight.setBoardSocket(dai.CameraBoardSocket.RIGHT)

# Setting node configs
stereo.setOutputDepth(True)
stereo.setConfidenceThreshold(255)

spatialDetectionNetwork.setBlobPath(nnBlobPath)
spatialDetectionNetwork.setConfidenceThreshold(0.5)
spatialDetectionNetwork.input.setBlocking(False)
spatialDetectionNetwork.setBoundingBoxScaleFactor(0.5)
spatialDetectionNetwork.setDepthLowerThreshold(100)
spatialDetectionNetwork.setDepthUpperThreshold(5000)

# Create outputs

monoLeft.out.link(stereo.left)
monoRight.out.link(stereo.right)

colorCam.preview.link(spatialDetectionNetwork.input)
if syncNN:
    spatialDetectionNetwork.passthrough.link(xoutRgb.input)
else:
    colorCam.preview.link(xoutRgb.input)

spatialDetectionNetwork.out.link(xoutNN.input)
spatialDetectionNetwork.boundingBoxMapping.link(xoutBoundingBoxDepthMapping.input)

stereo.depth.link(spatialDetectionNetwork.inputDepth)
spatialDetectionNetwork.passthroughDepth.link(xoutDepth.input)

# Pipeline is defined, now we can connect to the device
with dai.Device(pipeline) as device:
    # Start pipeline
    device.startPipeline()

    # Output queues will be used to get the rgb frames and nn data from the outputs␣
↪defined above
    previewQueue = device.getOutputQueue(name="rgb", maxSize=4, blocking=False)
    detectionNNQueue = device.getOutputQueue(name="detections", maxSize=4,␣
↪blocking=False)
    xoutBoundingBoxDepthMapping = device.getOutputQueue(name="boundingBoxDepthMapping
↪", maxSize=4, blocking=False)
    depthQueue = device.getOutputQueue(name="depth", maxSize=4, blocking=False)

    frame = None
    detections = []

    startTime = time.monotonic()
```

(continues on next page)

(continued from previous page)

```
    counter = 0
    fps = 0
    color = (255, 255, 255)

    while True:
        inPreview = previewQueue.get()
        inNN = detectionNNQueue.get()
        depth = depthQueue.get()

        counter+=1
        current_time = time.monotonic()
        if (current_time - startTime) > 1 :
            fps = counter / (current_time - startTime)
            counter = 0
            startTime = current_time

        frame = inPreview.getCvFrame()
        depthFrame = depth.getFrame()

        depthFrameColor = cv2.normalize(depthFrame, None, 255, 0, cv2.NORM_INF, cv2.CV_8UC1)
        depthFrameColor = cv2.equalizeHist(depthFrameColor)
        depthFrameColor = cv2.applyColorMap(depthFrameColor, cv2.COLORMAP_HOT)
        detections = inNN.detections
        if len(detections) != 0:
            boundingBoxMapping = xoutBoundingBoxDepthMapping.get()
            roiDatas = boundingBoxMapping.getConfigData()

            for roiData in roiDatas:
                roi = roiData.roi
                roi = roi.denormalize(depthFrameColor.shape[1], depthFrameColor.shape[0])
                topLeft = roi.topLeft()
                bottomRight = roi.bottomRight()
                xmin = int(topLeft.x)
                ymin = int(topLeft.y)
                xmax = int(bottomRight.x)
                ymax = int(bottomRight.y)

                cv2.rectangle(depthFrameColor, (xmin, ymin), (xmax, ymax), color, cv2.FONT_HERSHEY_SCRIPT_SIMPLEX)


        # If the frame is available, draw bounding boxes on it and show the frame
        height = frame.shape[0]
        width  = frame.shape[1]
        for detection in detections:
            # Denormalize bounding box
            x1 = int(detection.xmin * width)
            x2 = int(detection.xmax * width)
            y1 = int(detection.ymin * height)
            y2 = int(detection.ymax * height)
            try:
                label = labelMap[detection.label]
            except:
                label = detection.label
            cv2.putText(frame, str(label), (x1 + 10, y1 + 20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
```

(continues on next page)

(continued from previous page)

```
            cv2.putText(frame, "{:.2f}".format(detection.confidence*100), (x1 + 10, y1 + 35), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
            cv2.putText(frame, f"X: {int(detection.spatialCoordinates.x)} mm", (x1 + 10, y1 + 50), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
            cv2.putText(frame, f"Y: {int(detection.spatialCoordinates.y)} mm", (x1 + 10, y1 + 65), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
            cv2.putText(frame, f"Z: {int(detection.spatialCoordinates.z)} mm", (x1 + 10, y1 + 80), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)

            cv2.rectangle(frame, (x1, y1), (x2, y2), color, cv2.FONT_HERSHEY_SIMPLEX)

        cv2.putText(frame, "NN fps: {:.2f}".format(fps), (2, frame.shape[0] - 4), cv2.FONT_HERSHEY_TRIPLEX, 0.4, color)
        cv2.imshow("depth", depthFrameColor)
        cv2.imshow("rgb", frame)

        if cv2.waitKey(1) == ord('q'):
            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 MobilenetSDD blob (`mobilenet-ssd_openvino_2021.2_6shave.blob` file) to work - you can download it from here

Source code

Also available on GitHub

```python
#!/usr/bin/env python3

from pathlib import Path
import sys
import cv2
import depthai as dai
import numpy as np
import time

'''
Mobilenet SSD device side decoding demo
  The "mobilenet-ssd" model is a Single-Shot multibox Detection (SSD) network intended
  to perform object detection. This model is implemented using the Caffe* framework.
  For details about this model, check out the repository <https://github.com/chuanqi305/MobileNet-SSD>.
'''

# MobilenetSSD label texts
labelMap = ["background", "aeroplane", "bicycle", "bird", "boat", "bottle", "bus", "car", "cat", "chair", "cow",
            "diningtable", "dog", "horse", "motorbike", "person", "pottedplant", "sheep", "sofa", "train", "tvmonitor"]

syncNN = True
flipRectified = True

# Get argument first
nnPath = str((Path(__file__).parent / Path('models/mobilenet-ssd_openvino_2021.2_6shave.blob')).resolve().absolute())
if len(sys.argv) > 1:
    nnPath = sys.argv[1]

if not Path(nnPath).exists():
    import sys
    raise FileNotFoundError(f'Required file/s not found, please run "{sys.executable} install_requirements.py"')

# Start defining a pipeline
pipeline = dai.Pipeline()


manip = pipeline.createImageManip()
manip.initialConfig.setResize(300, 300)
# The NN model expects BGR input. By default ImageManip output type would be same as input (gray in this case)
manip.initialConfig.setFrameType(dai.ImgFrame.Type.BGR888p)
# manip.setKeepAspectRatio(False)

# Define a neural network that will make predictions based on the source frames
spatialDetectionNetwork = pipeline.createMobileNetSpatialDetectionNetwork()
spatialDetectionNetwork.setConfidenceThreshold(0.5)
spatialDetectionNetwork.setBlobPath(nnPath)
spatialDetectionNetwork.input.setBlocking(False)
spatialDetectionNetwork.setBoundingBoxScaleFactor(0.5)
```

(continues on next page)

(continued from previous page)

```
spatialDetectionNetwork.setDepthLowerThreshold(100)
spatialDetectionNetwork.setDepthUpperThreshold(5000)

manip.out.link(spatialDetectionNetwork.input)

# Create outputs
xoutManip = pipeline.createXLinkOut()
xoutManip.setStreamName("right")
if syncNN:
    spatialDetectionNetwork.passthrough.link(xoutManip.input)
else:
    manip.out.link(xoutManip.input)

depthRoiMap = pipeline.createXLinkOut()
depthRoiMap.setStreamName("boundingBoxDepthMapping")

xoutDepth = pipeline.createXLinkOut()
xoutDepth.setStreamName("depth")

nnOut = pipeline.createXLinkOut()
nnOut.setStreamName("detections")
spatialDetectionNetwork.out.link(nnOut.input)
spatialDetectionNetwork.boundingBoxMapping.link(depthRoiMap.input)

monoLeft = pipeline.createMonoCamera()
monoRight = pipeline.createMonoCamera()
stereo = pipeline.createStereoDepth()
monoLeft.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
monoLeft.setBoardSocket(dai.CameraBoardSocket.LEFT)
monoRight.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
monoRight.setBoardSocket(dai.CameraBoardSocket.RIGHT)
stereo.setOutputDepth(True)
stereo.setConfidenceThreshold(255)
stereo.setOutputRectified(True)

stereo.rectifiedRight.link(manip.inputImage)

monoLeft.out.link(stereo.left)
monoRight.out.link(stereo.right)

stereo.depth.link(spatialDetectionNetwork.inputDepth)
spatialDetectionNetwork.passthroughDepth.link(xoutDepth.input)

# Pipeline is defined, now we can connect to the device
with dai.Device(pipeline) as device:
    # Start pipeline
    device.startPipeline()

    # Output queues will be used to get the rgb frames and nn data from the outputs␣
↪defined above
    previewQueue = device.getOutputQueue(name="right", maxSize=4, blocking=False)
    detectionNNQueue = device.getOutputQueue(name="detections", maxSize=4,␣
↪blocking=False)
    depthRoiMap = device.getOutputQueue(name="boundingBoxDepthMapping", maxSize=4,␣
↪blocking=False)
    depthQueue = device.getOutputQueue(name="depth", maxSize=4, blocking=False)

```

(continues on next page)

(continued from previous page)

```
    rectifiedRight = None
    detections = []

    startTime = time.monotonic()
    counter = 0
    fps = 0
    color = (255, 255, 255)

    while True:
        inRectified = previewQueue.get()
        det = detectionNNQueue.get()
        depth = depthQueue.get()

        counter += 1
        currentTime = time.monotonic()
        if (currentTime - startTime) > 1:
            fps = counter / (currentTime - startTime)
            counter = 0
            startTime = currentTime

        rectifiedRight = inRectified.getCvFrame()

        depthFrame = depth.getFrame()

        depthFrameColor = cv2.normalize(depthFrame, None, 255, 0, cv2.NORM_INF, cv2.
↪CV_8UC1)
        depthFrameColor = cv2.equalizeHist(depthFrameColor)
        depthFrameColor = cv2.applyColorMap(depthFrameColor, cv2.COLORMAP_HOT)
        detections = det.detections
        if len(detections) != 0:
            boundingBoxMapping = depthRoiMap.get()
            roiDatas = boundingBoxMapping.getConfigData()

            for roiData in roiDatas:
                roi = roiData.roi
                roi = roi.denormalize(depthFrameColor.shape[1], depthFrameColor.
↪shape[0])
                topLeft = roi.topLeft()
                bottomRight = roi.bottomRight()
                xmin = int(topLeft.x)
                ymin = int(topLeft.y)
                xmax = int(bottomRight.x)
                ymax = int(bottomRight.y)
                cv2.rectangle(depthFrameColor, (xmin, ymin), (xmax, ymax), color, cv2.
↪FONT_HERSHEY_SCRIPT_SIMPLEX)

        if flipRectified:
            rectifiedRight = cv2.flip(rectifiedRight, 1)

        # If the rectifiedRight is available, draw bounding boxes on it and show the␣
↪rectifiedRight
        height = rectifiedRight.shape[0]
        width = rectifiedRight.shape[1]
        for detection in detections:
            if flipRectified:
                swap = detection.xmin
                detection.xmin = 1 - detection.xmax
```

(continues on next page)

(continued from previous page)

```
                detection.xmax = 1 - swap
            # Denormalize bounding box
            x1 = int(detection.xmin * width)
            x2 = int(detection.xmax * width)
            y1 = int(detection.ymin * height)
            y2 = int(detection.ymax * height)

            try:
                label = labelMap[detection.label]
            except:
                label = detection.label

            cv2.putText(rectifiedRight, str(label), (x1 + 10, y1 + 20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
            cv2.putText(rectifiedRight, "{:.2f}".format(detection.confidence*100), (x1 + 10, y1 + 35), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
            cv2.putText(rectifiedRight, f"X: {int(detection.spatialCoordinates.x)} mm", (x1 + 10, y1 + 50), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
            cv2.putText(rectifiedRight, f"Y: {int(detection.spatialCoordinates.y)} mm", (x1 + 10, y1 + 65), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
            cv2.putText(rectifiedRight, f"Z: {int(detection.spatialCoordinates.z)} mm", (x1 + 10, y1 + 80), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)

            cv2.rectangle(rectifiedRight, (x1, y1), (x2, y2), color, cv2.FONT_HERSHEY_SIMPLEX)

        cv2.putText(rectifiedRight, "NN fps: {:.2f}".format(fps), (2, rectifiedRight.shape[0] - 4), cv2.FONT_HERSHEY_TRIPLEX, 0.4, color)
        cv2.imshow("depth", depthFrameColor)
        cv2.imshow("rectified right", rectifiedRight)

        if cv2.waitKey(1) == ord('q'):
            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

```python
#!/usr/bin/env python3

from pathlib import Path
import sys
import cv2
import depthai as dai
import numpy as np
import time

'''
Spatial Tiny-yolo example
  Performs inference on RGB camera and retrieves spatial location coordinates: x,y,z relative to the center of depth map.
  Can be used for tiny-yolo-v3 or tiny-yolo-v4 networks
'''

# Tiny yolo v3/4 label texts
labelMap = [
    "person",         "bicycle",    "car",           "motorbike",     "aeroplane",   "bus",           "train",
    "truck",          "boat",       "traffic light", "fire hydrant",  "stop sign",   "parking meter", "bench",
    "bird",           "cat",        "dog",           "horse",         "sheep",       "cow",           "elephant",
    "bear",           "zebra",      "giraffe",       "backpack",      "umbrella",    "handbag",       "tie",
    "suitcase",       "frisbee",    "skis",          "snowboard",     "sports ball", "kite",          "baseball bat",
    "baseball glove", "skateboard", "surfboard",     "tennis racket", "bottle",      "wine glass",    "cup",
    "fork",           "knife",      "spoon",         "bowl",          "banana",      "apple",         "sandwich",
    "orange",         "broccoli",   "carrot",        "hot dog",       "pizza",       "donut",         "cake",
    "chair",          "sofa",       "pottedplant",   "bed",           "diningtable", "toilet",        "tvmonitor",
```

(continues on next page)

(continued from previous page)

```python
    "laptop",         "mouse",      "remote",        "keyboard",      "cell phone",      "microwave",      "oven",
    "toaster",        "sink",       "refrigerator",  "book",          "clock",      "vase",           "scissors",
    "teddy bear",     "hair drier", "toothbrush"
]

syncNN = True

# Get argument first
nnBlobPath = str((Path(__file__).parent / Path('models/tiny-yolo-v4_openvino_2021.2_6shave.blob')).resolve().absolute())
if len(sys.argv) > 1:
    nnBlobPath = sys.argv[1]

if not Path(nnBlobPath).exists():
    import sys
    raise FileNotFoundError(f'Required file/s not found, please run "{sys.executable} install_requirements.py"')

# Start defining a pipeline
pipeline = dai.Pipeline()

# Define a source - color camera
colorCam = pipeline.createColorCamera()
spatialDetectionNetwork = pipeline.createYoloSpatialDetectionNetwork()
monoLeft = pipeline.createMonoCamera()
monoRight = pipeline.createMonoCamera()
stereo = pipeline.createStereoDepth()

xoutRgb = pipeline.createXLinkOut()
xoutNN = pipeline.createXLinkOut()
xoutBoundingBoxDepthMapping = pipeline.createXLinkOut()
xoutDepth = pipeline.createXLinkOut()

xoutRgb.setStreamName("rgb")
xoutNN.setStreamName("detections")
xoutBoundingBoxDepthMapping.setStreamName("boundingBoxDepthMapping")
xoutDepth.setStreamName("depth")


colorCam.setPreviewSize(416, 416)
colorCam.setResolution(dai.ColorCameraProperties.SensorResolution.THE_1080_P)
colorCam.setInterleaved(False)
colorCam.setColorOrder(dai.ColorCameraProperties.ColorOrder.BGR)

monoLeft.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
monoLeft.setBoardSocket(dai.CameraBoardSocket.LEFT)
monoRight.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
monoRight.setBoardSocket(dai.CameraBoardSocket.RIGHT)

# setting node configs
stereo.setOutputDepth(True)
stereo.setConfidenceThreshold(255)

spatialDetectionNetwork.setBlobPath(nnBlobPath)
spatialDetectionNetwork.setConfidenceThreshold(0.5)
```

(continues on next page)

(continued from previous page)

```
spatialDetectionNetwork.input.setBlocking(False)
spatialDetectionNetwork.setBoundingBoxScaleFactor(0.5)
spatialDetectionNetwork.setDepthLowerThreshold(100)
spatialDetectionNetwork.setDepthUpperThreshold(5000)
# Yolo specific parameters
spatialDetectionNetwork.setNumClasses(80)
spatialDetectionNetwork.setCoordinateSize(4)
spatialDetectionNetwork.setAnchors(np.array([10,14, 23,27, 37,58, 81,82, 135,169, 344, 319]))
spatialDetectionNetwork.setAnchorMasks({ "side26": np.array([1,2,3]), "side13": np.array([3,4,5]) })
spatialDetectionNetwork.setIouThreshold(0.5)

# Create outputs

monoLeft.out.link(stereo.left)
monoRight.out.link(stereo.right)

colorCam.preview.link(spatialDetectionNetwork.input)
if syncNN:
    spatialDetectionNetwork.passthrough.link(xoutRgb.input)
else:
    colorCam.preview.link(xoutRgb.input)

spatialDetectionNetwork.out.link(xoutNN.input)
spatialDetectionNetwork.boundingBoxMapping.link(xoutBoundingBoxDepthMapping.input)

stereo.depth.link(spatialDetectionNetwork.inputDepth)
spatialDetectionNetwork.passthroughDepth.link(xoutDepth.input)

# Pipeline is defined, now we can connect to the device
with dai.Device(pipeline) as device:
    # Start pipeline
    device.startPipeline()

    # Output queues will be used to get the rgb frames and nn data from the outputs defined above
    previewQueue = device.getOutputQueue(name="rgb", maxSize=4, blocking=False)
    detectionNNQueue = device.getOutputQueue(name="detections", maxSize=4, blocking=False)
    xoutBoundingBoxDepthMapping = device.getOutputQueue(name="boundingBoxDepthMapping", maxSize=4, blocking=False)
    depthQueue = device.getOutputQueue(name="depth", maxSize=4, blocking=False)

    frame = None
    detections = []

    startTime = time.monotonic()
    counter = 0
    fps = 0
    color = (255, 255, 255)

    while True:
        inPreview = previewQueue.get()
        inNN = detectionNNQueue.get()
        depth = depthQueue.get()

```

(continues on next page)

(continued from previous page)

```
        counter+=1
        current_time = time.monotonic()
        if (current_time - startTime) > 1 :
            fps = counter / (current_time - startTime)
            counter = 0
            startTime = current_time

        frame = inPreview.getCvFrame()
        depthFrame = depth.getFrame()

        depthFrameColor = cv2.normalize(depthFrame, None, 255, 0, cv2.NORM_INF, cv2.
↪CV_8UC1)
        depthFrameColor = cv2.equalizeHist(depthFrameColor)
        depthFrameColor = cv2.applyColorMap(depthFrameColor, cv2.COLORMAP_HOT)
        detections = inNN.detections
        if len(detections) != 0:
            boundingBoxMapping = xoutBoundingBoxDepthMapping.get()
            roiDatas = boundingBoxMapping.getConfigData()

            for roiData in roiDatas:
                roi = roiData.roi
                roi = roi.denormalize(depthFrameColor.shape[1], depthFrameColor.
↪shape[0])
                topLeft = roi.topLeft()
                bottomRight = roi.bottomRight()
                xmin = int(topLeft.x)
                ymin = int(topLeft.y)
                xmax = int(bottomRight.x)
                ymax = int(bottomRight.y)

                cv2.rectangle(depthFrameColor, (xmin, ymin), (xmax, ymax), color, cv2.
↪FONT_HERSHEY_SCRIPT_SIMPLEX)


        # If the frame is available, draw bounding boxes on it and show the frame
        height = frame.shape[0]
        width  = frame.shape[1]
        for detection in detections:
            # Denormalize bounding box
            x1 = int(detection.xmin * width)
            x2 = int(detection.xmax * width)
            y1 = int(detection.ymin * height)
            y2 = int(detection.ymax * height)
            try:
                label = labelMap[detection.label]
            except:
                label = detection.label
            cv2.putText(frame, str(label), (x1 + 10, y1 + 20), cv2.FONT_HERSHEY_
↪TRIPLEX, 0.5, color)
            cv2.putText(frame, "{:.2f}".format(detection.confidence*100), (x1 + 10,␣
↪y1 + 35), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
            cv2.putText(frame, f"X: {int(detection.spatialCoordinates.x)} mm", (x1 +␣
↪10, y1 + 50), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
            cv2.putText(frame, f"Y: {int(detection.spatialCoordinates.y)} mm", (x1 +␣
↪10, y1 + 65), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
            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)

```

            cv2.rectangle(frame, (x1, y1), (x2, y2), color, cv2.FONT_HERSHEY_SIMPLEX)

        cv2.putText(frame, "NN fps: {:.2f}".format(fps), (2, frame.shape[0] - 4), cv2.
↪FONT_HERSHEY_TRIPLEX, 0.4, color)
        cv2.imshow("depth", depthFrameColor)
        cv2.imshow("rgb", frame)

        if cv2.waitKey(1) == ord('q'):
            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

```
#!/usr/bin/env python3

import cv2
import depthai as dai

stepSize = 0.05

# Start defining a pipeline
pipeline = dai.Pipeline()

# Define a source - two mono (grayscale) cameras
monoLeft = pipeline.createMonoCamera()
monoRight = pipeline.createMonoCamera()
stereo = pipeline.createStereoDepth()
spatialLocationCalculator = pipeline.createSpatialLocationCalculator()

xoutDepth = pipeline.createXLinkOut()
```

(continues on next page)

(continued from previous page)

```
xoutSpatialData = pipeline.createXLinkOut()
xinSpatialCalcConfig = pipeline.createXLinkIn()

xoutDepth.setStreamName("depth")
xoutSpatialData.setStreamName("spatialData")
xinSpatialCalcConfig.setStreamName("spatialCalcConfig")

# MonoCamera
monoLeft.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
monoLeft.setBoardSocket(dai.CameraBoardSocket.LEFT)
monoRight.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
monoRight.setBoardSocket(dai.CameraBoardSocket.RIGHT)

outputDepth = True
outputRectified = False
lrcheck = False
subpixel = False

# StereoDepth
stereo.setOutputDepth(outputDepth)
stereo.setOutputRectified(outputRectified)
stereo.setConfidenceThreshold(255)

stereo.setLeftRightCheck(lrcheck)
stereo.setSubpixel(subpixel)

monoLeft.out.link(stereo.left)
monoRight.out.link(stereo.right)

spatialLocationCalculator.passthroughDepth.link(xoutDepth.input)
stereo.depth.link(spatialLocationCalculator.inputDepth)

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

spatialLocationCalculator.setWaitForConfigInput(False)
config = dai.SpatialLocationCalculatorConfigData()
config.depthThresholds.lowerThreshold = 100
config.depthThresholds.upperThreshold = 10000
config.roi = dai.Rect(topLeft, bottomRight)
spatialLocationCalculator.initialConfig.addROI(config)
spatialLocationCalculator.out.link(xoutSpatialData.input)
xinSpatialCalcConfig.out.link(spatialLocationCalculator.inputConfig)

# Pipeline is defined, now we can connect to the device
with dai.Device(pipeline) as device:
    device.startPipeline()

    # Output queue will be used to get the depth frames from the outputs defined above
    depthQueue = device.getOutputQueue(name="depth", maxSize=4, blocking=False)
    spatialCalcQueue = device.getOutputQueue(name="spatialData", maxSize=4, blocking=False)
    spatialCalcConfigInQueue = device.getInputQueue("spatialCalcConfig")

    color = (255, 255, 255)

    print("Use WASD keys to move ROI!")
```

(continues on next page)

(continued from previous page)

```

    while True:
        inDepth = depthQueue.get() # Blocking call, will wait until a new data has arrived
        inDepthAvg = spatialCalcQueue.get() # Blocking call, will wait until a new data has arrived

        depthFrame = inDepth.getFrame()
        depthFrameColor = cv2.normalize(depthFrame, None, 255, 0, cv2.NORM_INF, cv2.CV_8UC1)
        depthFrameColor = cv2.equalizeHist(depthFrameColor)
        depthFrameColor = cv2.applyColorMap(depthFrameColor, cv2.COLORMAP_HOT)

        spatialData = inDepthAvg.getSpatialLocations()
        for depthData in spatialData:
            roi = depthData.config.roi
            roi = roi.denormalize(width=depthFrameColor.shape[1], height=depthFrameColor.shape[0])
            xmin = int(roi.topLeft().x)
            ymin = int(roi.topLeft().y)
            xmax = int(roi.bottomRight().x)
            ymax = int(roi.bottomRight().y)

            fontType = cv2.FONT_HERSHEY_TRIPLEX
            cv2.rectangle(depthFrameColor, (xmin, ymin), (xmax, ymax), color, cv2.FONT_HERSHEY_SCRIPT_SIMPLEX)
            cv2.putText(depthFrameColor, f"X: {int(depthData.spatialCoordinates.x)} mm", (xmin + 10, ymin + 20), fontType, 0.5, color)
            cv2.putText(depthFrameColor, f"Y: {int(depthData.spatialCoordinates.y)} mm", (xmin + 10, ymin + 35), fontType, 0.5, color)
            cv2.putText(depthFrameColor, f"Z: {int(depthData.spatialCoordinates.z)} mm", (xmin + 10, ymin + 50), fontType, 0.5, color)


        cv2.imshow("depth", depthFrameColor)

        newConfig = False
        key = cv2.waitKey(1)
        if key == ord('q'):
            break
        elif key == ord('w'):
            if topLeft.y - stepSize >= 0:
                topLeft.y -= stepSize
                bottomRight.y -= stepSize
                newConfig = True
        elif key == ord('a'):
            if topLeft.x - stepSize >= 0:
                topLeft.x -= stepSize
                bottomRight.x -= stepSize
                newConfig = True
        elif key == ord('s'):
            if bottomRight.y + stepSize <= 1:
                topLeft.y += stepSize
                bottomRight.y += stepSize
                newConfig = True
        elif key == ord('d'):
            if bottomRight.x + stepSize <= 1:
```

(continues on next page)

(continued from previous page)

```
                topLeft.x += stepSize
                bottomRight.x += stepSize
                newConfig = True

        if newConfig:
            config.roi = dai.Rect(topLeft, bottomRight)
            cfg = dai.SpatialLocationCalculatorConfig()
            cfg.addROI(config)
            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 resoulution (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

```
#!/usr/bin/env python3

import cv2
import depthai as dai
import numpy as np

# Start defining a pipeline
pipeline = dai.Pipeline()

# Define a source - color camera
colorCam = pipeline.createColorCamera()
colorCam.setBoardSocket(dai.CameraBoardSocket.RGB)
colorCam.setResolution(dai.ColorCameraProperties.SensorResolution.THE_1080_P)
colorCam.setVideoSize(1920, 1080)

# Create output
xoutVideo = pipeline.createXLinkOut()
xoutVideo.setStreamName("video")
xoutVideo.input.setBlocking(False)
```

(continues on next page)

(continued from previous page)

```
xoutVideo.input.setQueueSize(1)

colorCam.video.link(xoutVideo.input)

# Pipeline is defined, now we can connect to the device
with dai.Device(pipeline) as device:
    # Start pipeline
    device.startPipeline()
    video = device.getOutputQueue(name="video", maxSize=1, blocking=False)

    while True:
        # Get preview and video frames
        videoIn = video.get()

        # Get BGR frame from NV12 encoded video frame to show with opencv
        # Visualizing the frame on slower hosts might have overhead
        cv2.imshow("video", videoIn.getCvFrame())

        if cv2.waitKey(1) == ord('q'):
            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

```
#!/usr/bin/env python3

from pathlib import Path
import cv2
import depthai as dai
import numpy as np
import time
import argparse
```

(continues on next page)

(continued from previous page)

```python

labelMap = ["background", "aeroplane", "bicycle", "bird", "boat", "bottle", "bus", "car", "cat", "chair", "cow",
            "diningtable", "dog", "horse", "motorbike", "person", "pottedplant", "sheep", "sofa", "train", "tvmonitor"]

nnPathDefault = str((Path(__file__).parent / Path('models/mobilenet-ssd_openvino_2021.2_6shave.blob')).resolve().absolute())
parser = argparse.ArgumentParser()
parser.add_argument('nnPath', nargs='?', help="Path to mobilenet detection network blob", default=nnPathDefault)
parser.add_argument('-ff', '--full_frame', action="store_true", help="Perform tracking on full RGB frame", default=False)

args = parser.parse_args()


fullFrameTracking = args.full_frame

# Start defining a pipeline
pipeline = dai.Pipeline()

colorCam = pipeline.createColorCamera()
detectionNetwork = pipeline.createMobileNetDetectionNetwork()
objectTracker = pipeline.createObjectTracker()
trackerOut = pipeline.createXLinkOut()

xlinkOut = pipeline.createXLinkOut()

xlinkOut.setStreamName("preview")
trackerOut.setStreamName("tracklets")

colorCam.setPreviewSize(300, 300)
colorCam.setResolution(dai.ColorCameraProperties.SensorResolution.THE_1080_P)
colorCam.setInterleaved(False)
colorCam.setColorOrder(dai.ColorCameraProperties.ColorOrder.BGR)
colorCam.setFps(40)

# setting node configs
detectionNetwork.setBlobPath(args.nnPath)
detectionNetwork.setConfidenceThreshold(0.5)
detectionNetwork.input.setBlocking(False)

# Link plugins CAM . NN . XLINK
colorCam.preview.link(detectionNetwork.input)
objectTracker.passthroughTrackerFrame.link(xlinkOut.input)


objectTracker.setDetectionLabelsToTrack([15])  # track only person
# possible tracking types: ZERO_TERM_COLOR_HISTOGRAM, ZERO_TERM_IMAGELESS
objectTracker.setTrackerType(dai.TrackerType.ZERO_TERM_COLOR_HISTOGRAM)
# take the smallest ID when new object is tracked, possible options: SMALLEST_ID, UNIQUE_ID
objectTracker.setTrackerIdAssigmentPolicy(dai.TrackerIdAssigmentPolicy.SMALLEST_ID)

if fullFrameTracking:
    colorCam.video.link(objectTracker.inputTrackerFrame)
```

(continues on next page)

(continued from previous page)

```
else:
    detectionNetwork.passthrough.link(objectTracker.inputTrackerFrame)

detectionNetwork.passthrough.link(objectTracker.inputDetectionFrame)
detectionNetwork.out.link(objectTracker.inputDetections)
objectTracker.out.link(trackerOut.input)


# Pipeline defined, now the device is connected to
with dai.Device(pipeline) as device:

    # Start the pipeline
    device.startPipeline()

    preview = device.getOutputQueue("preview", 4, False)
    tracklets = device.getOutputQueue("tracklets", 4, False)

    startTime = time.monotonic()
    counter = 0
    fps = 0
    frame = None

    while(True):
        imgFrame = preview.get()
        track = tracklets.get()

        counter+=1
        current_time = time.monotonic()
        if (current_time - startTime) > 1 :
            fps = counter / (current_time - startTime)
            counter = 0
            startTime = current_time

        color = (255, 0, 0)
        frame = imgFrame.getCvFrame()
        trackletsData = track.tracklets
        for t in trackletsData:
            roi = t.roi.denormalize(frame.shape[1], frame.shape[0])
            x1 = int(roi.topLeft().x)
            y1 = int(roi.topLeft().y)
            x2 = int(roi.bottomRight().x)
            y2 = int(roi.bottomRight().y)

            try:
                label = labelMap[t.label]
            except:
                label = t.label

            statusMap = {dai.Tracklet.TrackingStatus.NEW : "NEW", dai.Tracklet.
↪TrackingStatus.TRACKED : "TRACKED", dai.Tracklet.TrackingStatus.LOST : "LOST"}
            cv2.putText(frame, str(label), (x1 + 10, y1 + 20), cv2.FONT_HERSHEY_
↪TRIPLEX, 0.5, color)
            cv2.putText(frame, f"ID: {[t.id]}", (x1 + 10, y1 + 35), cv2.FONT_HERSHEY_
↪TRIPLEX, 0.5, color)
            cv2.putText(frame, statusMap[t.status], (x1 + 10, y1 + 50), cv2.FONT_
↪HERSHEY_TRIPLEX, 0.5, color)
            cv2.rectangle(frame, (x1, y1), (x2, y2), color, cv2.FONT_HERSHEY_SIMPLEX)
```

(continues on next page)

(continued from previous page)

```

        cv2.putText(frame, "NN fps: {:.2f}".format(fps), (2, frame.shape[0] - 4), cv2.FONT_HERSHEY_TRIPLEX, 0.4, color)

        cv2.imshow("tracker", frame)

        if cv2.waitKey(1) == ord('q'):
            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

```
#!/usr/bin/env python3

from pathlib import Path
import cv2
import depthai as dai
import numpy as np
import time
import argparse

labelMap = ["background", "aeroplane", "bicycle", "bird", "boat", "bottle", "bus", "car", "cat", "chair", "cow",
            "diningtable", "dog", "horse", "motorbike", "person", "pottedplant", "sheep", "sofa", "train", "tvmonitor"]

nnPathDefault = str((Path(__file__).parent / Path('models/mobilenet-ssd_openvino_2021.2_6shave.blob')).resolve().absolute())
parser = argparse.ArgumentParser()
parser.add_argument('nnPath', nargs='?', help="Path to mobilenet detection network blob", default=nnPathDefault)
```

(continues on next page)

(continued from previous page)

```
parser.add_argument('-ff', '--full_frame', action="store_true", help="Perform␣
↪tracking on full RGB frame", default=False)

args = parser.parse_args()


fullFrameTracking = args.full_frame

# Start defining a pipeline
pipeline = dai.Pipeline()

colorCam = pipeline.createColorCamera()
spatialDetectionNetwork = pipeline.createMobileNetSpatialDetectionNetwork()
monoLeft = pipeline.createMonoCamera()
monoRight = pipeline.createMonoCamera()
stereo = pipeline.createStereoDepth()
objectTracker = pipeline.createObjectTracker()

xoutRgb = pipeline.createXLinkOut()
trackerOut = pipeline.createXLinkOut()

xoutRgb.setStreamName("preview")
trackerOut.setStreamName("tracklets")

colorCam.setPreviewSize(300, 300)
colorCam.setResolution(dai.ColorCameraProperties.SensorResolution.THE_1080_P)
colorCam.setInterleaved(False)
colorCam.setColorOrder(dai.ColorCameraProperties.ColorOrder.BGR)

monoLeft.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
monoLeft.setBoardSocket(dai.CameraBoardSocket.LEFT)
monoRight.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
monoRight.setBoardSocket(dai.CameraBoardSocket.RIGHT)

# setting node configs
stereo.setOutputDepth(True)
stereo.setConfidenceThreshold(255)

spatialDetectionNetwork.setBlobPath(args.nnPath)
spatialDetectionNetwork.setConfidenceThreshold(0.5)
spatialDetectionNetwork.input.setBlocking(False)
spatialDetectionNetwork.setBoundingBoxScaleFactor(0.5)
spatialDetectionNetwork.setDepthLowerThreshold(100)
spatialDetectionNetwork.setDepthUpperThreshold(5000)

# Create outputs

monoLeft.out.link(stereo.left)
monoRight.out.link(stereo.right)

# Link plugins CAM . NN . XLINK
colorCam.preview.link(spatialDetectionNetwork.input)
objectTracker.passthroughTrackerFrame.link(xoutRgb.input)


objectTracker.setDetectionLabelsToTrack([15])  # track only person
# possible tracking types: ZERO_TERM_COLOR_HISTOGRAM, ZERO_TERM_IMAGELESS
```

(continues on next page)

(continued from previous page)

```
objectTracker.setTrackerType(dai.TrackerType.ZERO_TERM_COLOR_HISTOGRAM)
# take the smallest ID when new object is tracked, possible options: SMALLEST_ID, UNIQUE_ID
objectTracker.setTrackerIdAssigmentPolicy(dai.TrackerIdAssigmentPolicy.SMALLEST_ID)

objectTracker.out.link(trackerOut.input)
if fullFrameTracking:
    colorCam.setPreviewKeepAspectRatio(False)
    colorCam.video.link(objectTracker.inputTrackerFrame)
    objectTracker.inputTrackerFrame.setBlocking(False)
    # do not block the pipeline if it's too slow on full frame
    objectTracker.inputTrackerFrame.setQueueSize(2)
else:
    spatialDetectionNetwork.passthrough.link(objectTracker.inputTrackerFrame)

spatialDetectionNetwork.passthrough.link(objectTracker.inputDetectionFrame)
spatialDetectionNetwork.out.link(objectTracker.inputDetections)

stereo.depth.link(spatialDetectionNetwork.inputDepth)


# Pipeline defined, now the device is connected to
with dai.Device(pipeline) as device:

    # Start the pipeline
    device.startPipeline()

    preview = device.getOutputQueue("preview", 4, False)
    tracklets = device.getOutputQueue("tracklets", 4, False)

    startTime = time.monotonic()
    counter = 0
    fps = 0
    frame = None

    while(True):
        imgFrame = preview.get()
        track = tracklets.get()

        counter+=1
        current_time = time.monotonic()
        if (current_time - startTime) > 1 :
            fps = counter / (current_time - startTime)
            counter = 0
            startTime = current_time

        color = (255, 0, 0)
        frame = imgFrame.getCvFrame()
        trackletsData = track.tracklets
        for t in trackletsData:
            roi = t.roi.denormalize(frame.shape[1], frame.shape[0])
            x1 = int(roi.topLeft().x)
            y1 = int(roi.topLeft().y)
            x2 = int(roi.bottomRight().x)
            y2 = int(roi.bottomRight().y)

            try:
```

(continues on next page)

(continued from previous page)

```python
                label = labelMap[t.label]
            except:
                label = t.label
            statusMap = {dai.Tracklet.TrackingStatus.NEW : "NEW", dai.Tracklet.TrackingStatus.TRACKED : "TRACKED", dai.Tracklet.TrackingStatus.LOST : "LOST"}
            cv2.putText(frame, str(label), (x1 + 10, y1 + 20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
            cv2.putText(frame, f"ID: {[t.id]}", (x1 + 10, y1 + 35), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
            cv2.putText(frame, statusMap[t.status], (x1 + 10, y1 + 50), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
            cv2.rectangle(frame, (x1, y1), (x2, y2), color, cv2.FONT_HERSHEY_SIMPLEX)

            cv2.putText(frame, f"X: {int(t.spatialCoordinates.x)} mm", (x1 + 10, y1 + 65), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
            cv2.putText(frame, f"Y: {int(t.spatialCoordinates.y)} mm", (x1 + 10, y1 + 80), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
            cv2.putText(frame, f"Z: {int(t.spatialCoordinates.z)} mm", (x1 + 10, y1 + 95), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)

        cv2.putText(frame, "NN fps: {:.2f}".format(fps), (2, frame.shape[0] - 4), cv2.FONT_HERSHEY_TRIPLEX, 0.4, color)

        cv2.imshow("tracker", frame)

        if cv2.waitKey(1) == ord('q'):
            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

```python
#!/usr/bin/env python3

from pathlib import Path
import cv2
import depthai as dai
import numpy as np
import time
import argparse

labelMap = ["person", ""]

nnPathDefault = str((Path(__file__).parent / Path('models/person-detection-0201_openvino_2021.3_7shave.blob')).resolve().absolute())
videoPathDefault = str((Path(__file__).parent / Path('models/construction_vest.mp4')).resolve().absolute())
parser = argparse.ArgumentParser()
parser.add_argument('-nnPath', help="Path to mobilenet detection network blob", default=nnPathDefault)
parser.add_argument('-v', '--videoPath', help="Path to video frame", default=videoPathDefault)

args = parser.parse_args()

# Start defining a pipeline
pipeline = dai.Pipeline()

# Create neural network input
xinFrame = pipeline.createXLinkIn()
xinFrame.setStreamName("inFrame")
xinFrame.setMaxDataSize(1920*1080*3)

detectionNetwork = pipeline.createMobileNetDetectionNetwork()
objectTracker = pipeline.createObjectTracker()
trackerOut = pipeline.createXLinkOut()

xlinkOut = pipeline.createXLinkOut()

xlinkOut.setStreamName("trackerFrame")
trackerOut.setStreamName("tracklets")

# Create a node to convert the grayscale frame into the nn-acceptable form
manip = pipeline.createImageManip()
manip.initialConfig.setResizeThumbnail(384, 384)
# manip.initialConfig.setResize(384, 384)
# manip.initialConfig.setKeepAspectRatio(False) #squash the image to not lose FOV
# The NN model expects BGR input. By default ImageManip output type would be same as input (gray in this case)
manip.initialConfig.setFrameType(dai.ImgFrame.Type.BGR888p)
xinFrame.out.link(manip.inputImage)
manip.inputImage.setBlocking(True)

manipOut = pipeline.createXLinkOut()
manipOut.setStreamName("manip")
manip.out.link(manipOut.input)
```

(continues on next page)

(continued from previous page)

```

nnOut = pipeline.createXLinkOut()
nnOut.setStreamName("nn")
detectionNetwork.out.link(nnOut.input)


# setting node configs
detectionNetwork.setBlobPath(args.nnPath)
detectionNetwork.setConfidenceThreshold(0.5)

manip.out.link(detectionNetwork.input)
detectionNetwork.input.setBlocking(True)
objectTracker.passthroughTrackerFrame.link(xlinkOut.input)


objectTracker.setDetectionLabelsToTrack([0])  # track only person
# possible tracking types: ZERO_TERM_COLOR_HISTOGRAM, ZERO_TERM_IMAGELESS
objectTracker.setTrackerType(dai.TrackerType.ZERO_TERM_COLOR_HISTOGRAM)
# take the smallest ID when new object is tracked, possible options: SMALLEST_ID, UNIQUE_ID
objectTracker.setTrackerIdAssigmentPolicy(dai.TrackerIdAssigmentPolicy.SMALLEST_ID)

xinFrame.out.link(objectTracker.inputTrackerFrame)
objectTracker.inputTrackerFrame.setBlocking(True)
detectionNetwork.passthrough.link(objectTracker.inputDetectionFrame)
objectTracker.inputDetectionFrame.setBlocking(True)
detectionNetwork.out.link(objectTracker.inputDetections)
objectTracker.inputDetections.setBlocking(True)
objectTracker.out.link(trackerOut.input)


# Pipeline defined, now the device is connected to
with dai.Device(pipeline) as device:

    # Start the pipeline
    device.startPipeline()

    qIn = device.getInputQueue(name="inFrame")
    trackerFrameQ = device.getOutputQueue("trackerFrame", 4)
    tracklets = device.getOutputQueue("tracklets", 4)
    qManip = device.getOutputQueue("manip", maxSize=4)
    qDet = device.getOutputQueue("nn", maxSize=4)

    startTime = time.monotonic()
    counter = 0
    fps = 0
    detections = []
    frame = None

    def to_planar(arr: np.ndarray, shape: tuple) -> np.ndarray:
        return cv2.resize(arr, shape).transpose(2, 0, 1).flatten()

    # nn data, being the bounding box locations, are in <0..1> range - they need to be normalized with frame width/height
    def frameNorm(frame, bbox):
        normVals = np.full(len(bbox), frame.shape[0])
        normVals[::2] = frame.shape[1]
```

(continues on next page)

(continued from previous page)

```
        return (np.clip(np.array(bbox), 0, 1) * normVals).astype(int)

    def displayFrame(name, frame):
        for detection in detections:
            bbox = frameNorm(frame, (detection.xmin, detection.ymin, detection.xmax, detection.ymax))
            cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (255, 0, 0), 2)
            cv2.putText(frame, labelMap[detection.label], (bbox[0] + 10, bbox[1] + 20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
            cv2.putText(frame, f"{int(detection.confidence * 100)}%", (bbox[0] + 10, bbox[1] + 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
        cv2.imshow(name, frame)

    cap = cv2.VideoCapture(args.videoPath)
    baseTs = time.monotonic()
    simulatedFps = 30
    inputFrameShape = (1280, 720)

    while cap.isOpened():
        read_correctly, frame = cap.read()
        if not read_correctly:
            break

        img = dai.ImgFrame()
        img.setType(dai.ImgFrame.Type.BGR888p)
        img.setData(to_planar(frame, inputFrameShape))
        img.setTimestamp(baseTs)
        baseTs += 1/simulatedFps

        img.setWidth(inputFrameShape[0])
        img.setHeight(inputFrameShape[1])
        qIn.send(img)

        trackFrame = trackerFrameQ.tryGet()
        if trackFrame is None:
            continue

        track = tracklets.get()
        manip = qManip.get()
        inDet = qDet.get()

        counter+=1
        current_time = time.monotonic()
        if (current_time - startTime) > 1 :
            fps = counter / (current_time - startTime)
            counter = 0
            startTime = current_time

        detections = inDet.detections
        manipFrame = manip.getCvFrame()
        displayFrame("nn", manipFrame)

        color = (255, 0, 0)
        trackerFrame = trackFrame.getCvFrame()
        trackletsData = track.tracklets
        for t in trackletsData:
```

(continues on next page)

(continued from previous page)

```
            roi = t.roi.denormalize(trackerFrame.shape[1], trackerFrame.shape[0])
            x1 = int(roi.topLeft().x)
            y1 = int(roi.topLeft().y)
            x2 = int(roi.bottomRight().x)
            y2 = int(roi.bottomRight().y)

            try:
                label = labelMap[t.label]
            except:
                label = t.label

            statusMap = {dai.Tracklet.TrackingStatus.NEW : "NEW", dai.Tracklet.TrackingStatus.TRACKED : "TRACKED", dai.Tracklet.TrackingStatus.LOST : "LOST"}
            cv2.putText(trackerFrame, str(label), (x1 + 10, y1 + 20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
            cv2.putText(trackerFrame, f"ID: {[t.id]}", (x1 + 10, y1 + 35), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
            cv2.putText(trackerFrame, statusMap[t.status], (x1 + 10, y1 + 50), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
            cv2.rectangle(trackerFrame, (x1, y1), (x2, y2), color, cv2.FONT_HERSHEY_SIMPLEX)

        cv2.putText(trackerFrame, "Fps: {:.2f}".format(fps), (2, trackerFrame.shape[0] - 4), cv2.FONT_HERSHEY_TRIPLEX, 0.4, color)

        cv2.imshow("tracker", trackerFrame)

        if cv2.waitKey(1) == ord('q'):
            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

```python
#!/usr/bin/env python3

import cv2
import numpy as np
import depthai as dai
from time import sleep
import datetime
import argparse
from pathlib import Path

datasetDefault = str((Path(__file__).parent / Path('models/dataset')).resolve().absolute())
parser = argparse.ArgumentParser()
parser.add_argument('-dataset', nargs='?', help="Path to recorded frames", default=datasetDefault)
args = parser.parse_args()

if not Path(datasetDefault).exists():
    import sys
    raise FileNotFoundError(f'Required file/s not found, please run "{sys.executable} install_requirements.py"')


# StereoDepth config options.
out_depth = False  # Disparity by default
out_rectified = True   # Output and display rectified streams
lrcheck = True   # Better handling for occlusions
extended = False  # Closer-in minimum depth, disparity range is doubled
subpixel = True   # Better accuracy for longer distance, fractional disparity 32-levels
median = dai.StereoDepthProperties.MedianFilter.KERNEL_7x7

# Sanitize some incompatible options
if lrcheck or extended or subpixel:
    median = dai.StereoDepthProperties.MedianFilter.MEDIAN_OFF

print("StereoDepth config options: ")
print("Left-Right check: ", lrcheck)
print("Extended disparity: ", extended)
print("Subpixel: ", subpixel)
print("Median filtering: ", median)

right_intrinsic = [[860.0, 0.0, 640.0], [0.0, 860.0, 360.0], [0.0, 0.0, 1.0]]


def create_stereo_depth_pipeline():
    print("Creating Stereo Depth pipeline: ", end='')

    print("XLINK IN -> STEREO -> XLINK OUT")
    pipeline = dai.Pipeline()

    camLeft = pipeline.createXLinkIn()
    camRight = pipeline.createXLinkIn()
    stereo = pipeline.createStereoDepth()
```

(continues on next page)

(continued from previous page)

```
    xoutLeft = pipeline.createXLinkOut()
    xoutRight = pipeline.createXLinkOut()
    xoutDepth = pipeline.createXLinkOut()
    xoutDisparity = pipeline.createXLinkOut()
    xoutRectifLeft = pipeline.createXLinkOut()
    xoutRectifRight = pipeline.createXLinkOut()

    camLeft.setStreamName('in_left')
    camRight.setStreamName('in_right')

    stereo.setOutputDepth(out_depth)
    stereo.setOutputRectified(out_rectified)
    stereo.setConfidenceThreshold(200)
    stereo.setRectifyEdgeFillColor(0) # Black, to better see the cutout
    stereo.setMedianFilter(median) # KERNEL_7x7 default
    stereo.setLeftRightCheck(lrcheck)
    stereo.setExtendedDisparity(extended)
    stereo.setSubpixel(subpixel)

    stereo.setEmptyCalibration() # Set if the input frames are already rectified
    stereo.setInputResolution(1280, 720)

    xoutLeft.setStreamName('left')
    xoutRight.setStreamName('right')
    xoutDepth.setStreamName('depth')
    xoutDisparity.setStreamName('disparity')
    xoutRectifLeft.setStreamName('rectified_left')
    xoutRectifRight.setStreamName('rectified_right')

    camLeft.out.link(stereo.left)
    camRight.out.link(stereo.right)
    stereo.syncedLeft.link(xoutLeft.input)
    stereo.syncedRight.link(xoutRight.input)
    stereo.depth.link(xoutDepth.input)
    stereo.disparity.link(xoutDisparity.input)
    stereo.rectifiedLeft.link(xoutRectifLeft.input)
    stereo.rectifiedRight.link(xoutRectifRight.input)

    streams = ['left', 'right']
    if out_rectified:
        streams.extend(['rectified_left', 'rectified_right'])
    streams.extend(['disparity', 'depth'])

    return pipeline, streams


def convert_to_cv2_frame(name, image):
    baseline = 75 #mm
    focal = right_intrinsic[0][0]
    max_disp = 96
    disp_type = np.uint8
    disp_levels = 1
    if (extended):
        max_disp *= 2
    if (subpixel):
        max_disp *= 32
        disp_type = np.uint16
```

(continues on next page)

(continued from previous page)

```
        disp_levels = 32

    data, w, h = image.getData(), image.getWidth(), image.getHeight()
    if name == 'depth':
        # this contains FP16 with (lrcheck or extended or subpixel)
        frame = np.array(data).astype(np.uint8).view(np.uint16).reshape((h, w))
    elif name == 'disparity':
        disp = np.array(data).astype(np.uint8).view(disp_type).reshape((h, w))

        # Compute depth from disparity
        with np.errstate(divide='ignore'):
            depth = (disp_levels * baseline * focal / disp).astype(np.uint16)

        if 1: # Optionally, extend disparity range to better visualize it
            frame = (disp * 255. / max_disp).astype(np.uint8)

        if 1: # Optionally, apply a color map
            frame = cv2.applyColorMap(frame, cv2.COLORMAP_HOT)

    else: # mono streams / single channel
        frame = np.array(data).reshape((h, w)).astype(np.uint8)
        if name.startswith('rectified_'):
            frame = cv2.flip(frame, 1)
        if name == 'rectified_right':
            last_rectif_right = frame
    return frame


pipeline, streams = create_stereo_depth_pipeline()

print("Creating DepthAI device")
with dai.Device(pipeline) as device:
    print("Starting pipeline")
    device.startPipeline()

    inStreams = ['in_right', 'in_left']
    inStreamsCameraID = [dai.CameraBoardSocket.RIGHT, dai.CameraBoardSocket.LEFT]
    in_q_list = []
    for s in inStreams:
        q = device.getInputQueue(s)
        in_q_list.append(q)

    # Create a receive queue for each stream
    q_list = []
    for s in streams:
        q = device.getOutputQueue(s, 8, blocking=False)
        q_list.append(q)

    # Need to set a timestamp for input frames, for the sync stage in Stereo node
    timestamp_ms = 0
    index = 0
    while True:
        # Handle input streams, if any
        if in_q_list:
            dataset_size = 2  # Number of image pairs
            frame_interval_ms = 500
            for i, q in enumerate(in_q_list):
```

(continues on next page)

(continued from previous page)

```
                path = args.dataset + '/' + str(index) + '/' + q.getName() + '.png'
                data = cv2.imread(path, cv2.IMREAD_GRAYSCALE).reshape(720*1280)
                tstamp = datetime.timedelta(seconds = timestamp_ms // 1000,
                                            milliseconds = timestamp_ms % 1000)
                img = dai.ImgFrame()
                img.setData(data)
                img.setTimestamp(tstamp)
                img.setInstanceNum(inStreamsCameraID[i])
                img.setType(dai.ImgFrame.Type.RAW8)
                img.setWidth(1280)
                img.setHeight(720)
                q.send(img)
                if timestamp_ms == 0:  # Send twice for first iteration
                    q.send(img)
                print("Sent frame: {:25s}".format(path), 'timestamp_ms:', timestamp_ms)
            timestamp_ms += frame_interval_ms
            index = (index + 1) % dataset_size
            sleep(frame_interval_ms / 1000)
        # Handle output streams
        for q in q_list:
            if q.getName() in ['left', 'right', 'depth']: continue
            frame = convert_to_cv2_frame(q.getName(), q.get())
            cv2.imshow(q.getName(), frame)
        if cv2.waitKey(1) == ord('q'):
            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:

ADatatype	Abstract message
Asset	Asset is identified with string key and can store arbitrary binary data
AssetManager	AssetManager can store assets and serialize
Buffer	Base message - buffer of binary data
CameraBoardSocket	Which Camera socket to use.
CameraControl	CameraControl message Specifies various camera control commands like:
CameraImageOrientation	Camera sensor image orientation / pixel readout.
ChipTemperature	Chip temperature information.
ColorCamera	ColorCamera node.
ColorCameraProperties	Specify ColorCamera options such as camera ID, ...
CpuUsage	CpuUsage structure
DataInputQueue	Access to send messages through XLink stream
DataOutputQueue	Access to receive messages coming from XLink stream
DetectionNetwork	DetectionNetwork.
DetectionNetworkProperties	Properties for DetectionNetwork
Device	Represents the DepthAI device with the methods to interact with it.
DeviceBootloader	Represents the DepthAI bootloader with the methods to interact with it.
DeviceDesc	
DeviceInfo	
GlobalProperties	Specify properties which apply for whole pipeline
ImageManip	ImageManip node.
ImageManipConfig	ImageManipConfig message.
ImgDetection	
ImgDetections	ImgDetections message.
ImgFrame	ImgFrame message.
LogLevel	Members:
MemoryInfo	MemoryInfo structure
MobileNetDetectionNetwork	MobileNetDetectionNetwork node.
MobileNetSpatialDetectionNetwork	MobileNetSpatialDetectionNetwork.
MonoCamera	MonoCamera node.
MonoCameraProperties	Specify MonoCamera options such as camera ID, ...
NNData	NNData message.
NeuralNetwork	NeuralNetwork node.
NeuralNetworkProperties	Specify NeuralNetwork options such as blob path, ...
Node	Abstract Node
ObjectTracker	ObjectTracker node.
ObjectTrackerProperties	Properties for ObjectTracker
OpenVINO	Support for basic OpenVINO related actions like version identification of neural network blobs,...
Pipeline	Represents the pipeline, set of nodes and connections between them
Point2f	Point2f structure
Point3f	Point3f structure

continues on next page

Table 1 – continued from previous page

RawBuffer	
RawCameraControl	
RawImageManipConfig	
RawImgDetections	
RawImgFrame	
RawNNData	
RawSpatialImgDetections	
RawSystemInformation	System information of device
RawTracklets	
Rect	Rect structure
RotatedRect	
SPIOut	SPIOut node.
Size2f	
SpatialDetectionNetwork	SpatialDetectionNetwork node.
SpatialDetectionNetworkProperties	Properties for SpatialDetectionNetwork
SpatialImgDetection	Spatial image detection structure
SpatialImgDetections	SpatialImgDetections message.
SpatialLocationCalculator	SpatialLocationCalculator node.
SpatialLocationCalculatorConfig	SpatialLocationCalculatorConfig message.
SpatialLocationCalculatorConfigData	
SpatialLocationCalculatorConfigThresholds	Spatial location configuration thresholds structure
SpatialLocationCalculatorData	SpatialLocationCalculatorData message.
SpatialLocationCalculatorProperties	Specify SpatialLocationCalculator options
SpatialLocations	Spatial location information structure
StereoDepth	StereoDepth node.
StereoDepthProperties	Specify StereoDepth options
SystemInformation	SystemInformation message.
SystemLogger	SystemLogger node.
SystemLoggerProperties	
TensorInfo	
Timestamp	
TrackerIdAssigmentPolicy	Members:
TrackerType	Members:
Tracklet	Tracklet structure
Tracklets	Tracklets message.
VideoEncoder	VideoEncoder node.
VideoEncoderProperties	Specify VideoEncoder options such as profile, bitrate, …
XLinkConnection	
XLinkDeviceState	Members:
XLinkIn	XLinkIn node.
XLinkOut	XLinkOut node.
XLinkPlatform	Members:
XLinkProtocol	Members:
YoloDetectionNetwork	YoloDetectionNetwork node.
YoloSpatialDetectionNetwork	YoloSpatialDetectionNetwork.

class depthai.**ADatatype**

Bases: `pybind11_builtins.pybind11_object`

Abstract message

Methods:

`__init__`(*args, **kwargs)	Initialize self.
`getRaw`(self)	

`__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:

`__init__`(*args, **kwargs)	Overloaded function.

Attributes:

`alignment`
`data`
`key`

`__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:

`__init__`(self)	
`add`(*args, **kwargs)	Overloaded function.
`addExisting`(self, assets)	Adds all assets in an array to the AssetManager
`get`(*args, **kwargs)	Overloaded function.
`getAll`(*args, **kwargs)	Overloaded function.
`remove`(self, key)	Removes asset with key
`set`(self, key, asset)	Adds or overwrites existing asset with a specificied key.
`size`(self)	**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 specificied 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 specificied 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:

__init__(self)	Creates Buffer message
getData(self)	**returns** Reference to internal buffer
setData(*args, **kwargs)	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:

AUTO
LEFT
RGB
RIGHT
name
value

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:

`AntiBandingMode`	Members:
`AutoFocusMode`	Members:
`AutoWhiteBalanceMode`	Members:
`EffectMode`	Members:
`SceneMode`	Members:

Methods:

`__init__`(self)	Construct CameraControl message
`getCaptureStill`(self)	Check whether command to capture a still is set
`setAntiBandingMode`(self, mode)	Set a command to specify auto banding mode
`setAutoExposureCompensation`(self, compensation)	Set a command to specify auto exposure compenstaion
`setAutoExposureEnable`(self)	Set a command to enable auto exposure
`setAutoExposureLock`(self, lock)	Set a command to specify lock auto exposure
`setAutoExposureRegion`(self, startX, startY, …)	Set a command to specify auto exposure region in pixels
`setAutoFocusMode`(self, mode)	Set a command to specify autofocus mode
`setAutoFocusRegion`(self, startX, startY, …)	Set a command to specify focus region in pixels
`setAutoFocusTrigger`(self)	Set a command to trigger autofocus
`setAutoWhiteBalanceLock`(self, lock)	Set a command to specify auto white balance lock
`setAutoWhiteBalanceMode`(self, mode)	Set a command to specify auto white balance mode

continues on next page

Table 10 – continued from previous page

setBrightness(self, value)	Set a command to specify auto white balance lock
setCaptureStill(self, capture)	Set a command to capture a still image
setChromaDenoise(self, value)	Set a command to specify chroma denoise value
setContrast(self, value)	Set a command to specify auto white balance lock
setEffectMode(self, mode)	Set a command to specify effect mode
setLumaDenoise(self, value)	Set a command to specify luma denoise value
setManualExposure(self, exposureTimeUs, ...)	Set a command to manually specify exposure
setManualFocus(self, lensPosition)	Set a command to specify manual focus position
setNoiseReductionStrength(self, value)	Set a command to specify noise reduction strength
setSaturation(self, value)	Set a command to specify saturation value
setSceneMode(self, mode)	Set a command to specify scene mode
setSharpness(self, value)	Set a command to specify sharpness value
setStartStreaming(self)	Set a command to start streaming
setStopStreaming(self)	Set a command to stop streaming

class AntiBandingMode

Bases: `pybind11_builtins.pybind11_object`

Members:

OFF

MAINS_50_HZ

MAINS_60_HZ

AUTO

Attributes:

AUTO
MAINS_50_HZ
MAINS_60_HZ
OFF
name
value

Methods:

__init__(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:

AUTO
CONTINUOUS_PICTURE
CONTINUOUS_VIDEO
EDOF
MACRO
OFF
name
value

Methods:

__init__(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:

AUTO
CLOUDY_DAYLIGHT
DAYLIGHT
FLUORESCENT
INCANDESCENT
OFF
SHADE
TWILIGHT
WARM_FLUORESCENT
name
value

Methods:

__init__(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:

AQUA
BLACKBOARD
MONO
NEGATIVE
OFF
POSTERIZE
SEPIA
SOLARIZE
WHITEBOARD
name
value

Methods:

__init__(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:

ACTION
BARCODE
BEACH
CANDLELIGHT
FACE_PRIORITY
FIREWORKS
LANDSCAPE
NIGHT
NIGHT_PORTRAIT
PARTY
PORTRAIT
SNOW
SPORTS
STEADYPHOTO
SUNSET
THEATRE
UNSUPPORTED
name
value

Methods:

__init__(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:

`__init__`(*args, **kwargs)	Initialize self.
`getBoardSocket`(self)	Retrieves which board socket to use
`getCamId`(self)	
`getColorOrder`(self)	Get color order of preview output frames.
`getFp16`(self)	Get fp16 (0..255) data of preview output frames
`getFps`(self)	Get rate at which camera should produce frames
`getImageOrientation`(self)	Get camera image orientation
`getInterleaved`(self)	Get planar or interleaved data of preview output frames
`getPreviewHeight`(self)	Get preview height
`getPreviewKeepAspectRatio`(self)	**See also:** setPreviewKeepAspectRatio
`getPreviewSize`(self)	Get preview size as tuple
`getPreviewWidth`(self)	Get preview width
`getResolution`(self)	Get sensor resolution
`getResolutionHeight`(self)	Get sensor resolution height
`getResolutionSize`(self)	Get sensor resolution as size
`getResolutionWidth`(self)	Get sensor resolution width
`getSensorCrop`(self)	**returns** Sensor top left crop coordinates
`getSensorCropX`(self)	Get sensor top left x crop coordinate
`getSensorCropY`(self)	Get sensor top left y crop coordinate
`getStillHeight`(self)	Get still height
`getStillSize`(self)	Get still size as tuple
`getStillWidth`(self)	Get still width
`getVideoHeight`(self)	Get video height
`getVideoSize`(self)	Get video size as tuple
`getVideoWidth`(self)	Get video width
`getWaitForConfigInput`(self)	**See also:** setWaitForConfigInput
`sensorCenterCrop`(self)	Specify sensor center crop.
`setBoardSocket`(self, boardSocket)	Specify which board socket to use
`setCamId`(self, arg0)	
`setColorOrder`(self, colorOrder)	Set color order of preview output images.
`setFp16`(self, fp16)	Set fp16 (0..255) data type of preview output frames
`setFps`(self, fps)	Set rate at which camera should produce frames
`setImageOrientation`(self, boardSocket)	Set camera image orientation
`setInterleaved`(self, interleaved)	Set planar or interleaved data of preview output frames
`setPreviewKeepAspectRatio`(self, keep)	Specifies whether preview output should preserve aspect ratio, after downscaling from video size or not.
`setPreviewSize`(self, width, height)	Set preview output size
`setResolution`(self, resolution)	Set sensor resolution
`setSensorCrop`(self, x, y)	Specifies sensor crop rectangle
`setStillSize`(self, width, height)	Set still output size
`setVideoSize`(self, width, height)	Set video output size
`setWaitForConfigInput`(self, wait)	Specify to wait until inputConfig receives a configuration message, before sending out a frame.

Attributes:

initialControl	Initial control options to apply to sensor
inputConfig	Input for ImageManipConfig message, which can modify crop paremeters in runtime
inputControl	Input for CameraControl message, which can modify camera parameters in runtime
preview	Outputs ImgFrame message that carries BGR/RGB planar/interleaved encoded frame data.
still	Outputs ImgFrame message that carries NV12 encoded (YUV420, UV plane interleaved) frame data.
video	Outputs ImgFrame message that carries NV12 encoded (YUV420, UV plane interleaved) frame data.

Properties

alias of *depthai.ColorCameraProperties* **Classes:**

`ColorOrder`	For 24 bit color these can be either RGB or BGR
`SensorResolution`	Select the camera sensor resolution

Methods:

`__init__`(*args, **kwargs)	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:

ColorOrder	For 24 bit color these can be either RGB or BGR
SensorResolution	Select the camera sensor resolution

Methods:

__init__(*args, **kwargs)	Initialize self.

Attributes:

boardSocket
colorOrder

continues on next page

Table 33 – continued from previous page

fps
initialControl
interleaved
previewHeight
previewWidth
resolution
sensorCropX
sensorCropY
stillHeight
stillWidth
videoHeight
videoWidth

class ColorOrder

Bases: `pybind11_builtins.pybind11_object`

For 24 bit color these can be either RGB or BGR

Members:

BGR

RGB

Attributes:

BGR
RGB
name
value

Methods:

__init__(self, value)

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.
`getBlocking`(self)	Gets current queue behavior when full (maxSize)
`getMaxSize`(self, arg0)	Gets queue maximum size
`getName`(self)	Gets queues name
`send`(*args, **kwargs)	Overloaded function.
`setBlocking`(self, blocking)	Sets queue behavior when full (maxSize)
`setMaxSize`(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 queue

2. 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:

`__init__`(*args, **kwargs)	Initialize self.
`addCallback`(*args, **kwargs)	Overloaded function.
`get`(self)	Block until a message is available.
`getAll`(self)	Block until at least one message in the queue.
`getBlocking`(self)	Gets current queue behavior when full (maxSize)
`getMaxSize`(self, arg0)	Gets queue maximum size
`getName`(self)	Gets queues name
`has`(self)	Check whether front of the queue has a message (isn't empty)
`removeCallback`(self, callbackId)	Removes a callback
`setBlocking`(self, blocking)	Sets queue behavior when full (maxSize)
`setMaxSize`(self, maxSize)	Sets queue maximum size
`tryGet`(self)	Try to retrieve message from queue.
`tryGetAll`(self)	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 immidiately 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:

Properties	alias of *depthai.DetectionNetworkProperties*

Methods:

__init__(*args, **kwargs)	Initialize self.
setConfidenceThreshold(self, thresh)	Specifies confidence threshold at which to filter the rest of the detections.

Attributes:

input	Input message with data to be infered upon Default queue is blocking with size 5
out	Outputs ImgDetections message that carries parsed detection results.
passthrough	Passthrough message on which the inference was performed.

Properties

alias of *depthai.DetectionNetworkProperties* **Methods:**

__init__(*args, **kwargs)	Initialize self.

Attributes:

anchorMasks
anchors
classes
confidenceThreshold
coordinates
iouThreshold
nnFamily

__init__(**args*, ***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

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.

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:

__init__(*args, **kwargs)	Initialize self.

Attributes:

anchorMasks
anchors
classes
confidenceThreshold
coordinates
iouThreshold
nnFamily

__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:

__init__(*args, **kwargs)	Overloaded function.
addLogCallback(self, callback)	Add a callback for device logging.
close(self)	Closes the connection to device.
getAllAvailableDevices()	Returns all connected devices
getAnyAvailableDevice(*args, **kwargs)	Overloaded function.
getChipTemperature(self)	Retrieves current chip temperature as measured by device
getCmxMemoryUsage(self)	Retrieves current CMX memory information from device

continues on next page

Table 49 – continued from previous page

`getDdrMemoryUsage`(self)	Retrieves current DDR memory information from device
`getDeviceByMxId`(mxId)	Finds a device by MX ID.
`getEmbeddedDeviceBinary`(usb2Mode, version)	Gets device firmware binary for a specific OpenVINO version
`getFirstAvailableDevice`()	Gets first available device.
`getInputQueue`(*args, **kwargs)	Overloaded function.
`getInputQueueNames`(self)	Get all available input queue names
`getLeonCssCpuUsage`(self)	Retrieves average CSS Leon CPU usage
`getLeonCssHeapUsage`(self)	Retrieves current CSS Leon CPU heap information from device
`getLeonMssCpuUsage`(self)	Retrieves average MSS Leon CPU usage
`getLeonMssHeapUsage`(self)	Retrieves current MSS Leon CPU heap information from device
`getLogLevel`(self)	Gets current logging severity level of the device.
`getLogOutputLevel`(self)	Gets logging level which decides printing level to standard output.
`getOutputQueue`(*args, **kwargs)	Overloaded function.
`getOutputQueueNames`(self)	Get all available output queue names
`getQueueEvent`(*args, **kwargs)	Overloaded function.
`getQueueEvents`(*args, **kwargs)	Overloaded function.
`getSystemInformationLoggingRate`(self)	Gets current rate of system information logging ("info" severity) in Hz.
`isPipelineRunning`(self)	Checks if devices pipeline is already running
`removeLogCallback`(self, callbackId)	Removes a callback
`setLogLevel`(self, level)	Sets the devices logging severity level.
`setLogOutputLevel`(self, level)	Sets logging level which decides printing level to standard output.
`setSystemInformationLoggingRate`(self, rateHz)	Sets rate of system information logging ("info" severity).
`startPipeline`(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)>*) → 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

`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*) → 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*) → 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 transfered 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:

`Version`	Bootloader version structure

Methods:

`__init__`(*args, **kwargs)	Overloaded function.
`close`(self)	Closes the connection to device.
`createDepthaiApplicationPackage`(pipeline, ...)	Creates application package which can be flashed to depthai device.
`flash`(self, progressCallback, None], pipeline)	Flashes a give pipeline to the board.
`flashBootloader`(self, progressCallback, ...)	Flashes bootloader to the current board
`flashDepthaiApplicationPackage`(self, ...)	Flashes a specific depthai application package that was generated using createDepthaiApplicationPackage or saveDepthaiApplicationPackage
`getAllAvailableDevices`()	Searches for connected devices in either UNBOOTED or BOOTLOADER states.
`getEmbeddedBootloaderBinary`()	**returns** Embedded bootloader binary
`getEmbeddedBootloaderVersion`()	**returns** Embedded bootloader version
`getFirstAvailableDevice`()	Searches for connected devices in either UNBOOTED or BOOTLOADER states and returns first available.
`getVersion`(self)	**returns** Version of current running bootloader

continues on next page

Table 51 – continued from previous page

isEmbeddedVersion(self)	**returns** True whether the bootloader running is flashed or booted by library
saveDepthaiApplicationPackage(path, ...)	Saves application package to a file which can be flashed to depthai device.

class Version

Bases: `pybind11_builtins.pybind11_object`

Bootloader version structure

Methods:

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

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: depthai.DeviceBootloader.Version, v: str) -> None

Construct Version from string

2. __init__(self: depthai.DeviceBootloader.Version, major: int, minor: int, patch: int) -> None

Construct Version major, minor and patch numbers

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init__(self: depthai.DeviceBootloader, deviceDesc: depthai.DeviceInfo) -> None

2. __init__(self: depthai.DeviceBootloader, deviceDesc: depthai.DeviceInfo, pathToCmd: str) -> None

Connects to or boots device in bootloader mode depending on devInfo state.

Parameter `devInfo`: DeviceInfo of which to boot or connect to

close(*self:* depthai.DeviceBootloader) → None

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

static createDepthaiApplicationPackage(*pipeline:* depthai.Pipeline, *pathToCmd:* *str* = *''*) → List[int]

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

flash(*self:* depthai.DeviceBootloader, *progressCallback:* *Callable[[float], None]*, *pipeline:* depthai.Pipeline) → Tuple[bool, str]

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:

__init__(*args, **kwargs)	Initialize self.
setCenterCrop(self, arg0, arg1)	
setCropRect(self, arg0, arg1, arg2, arg3)	
setFrameType(self, arg0)	
setHorizontalFlip(self, arg0)	
setKeepAspectRatio(self, arg0)	
setMaxOutputFrameSize(self, arg0)	Specify maximum size of output image.
setNumFramesPool(self, arg0)	Specify number of frames in pool.
setResize(self, arg0, arg1)	
setResizeThumbnail(self, arg0, arg1, arg2, …)	
setWaitForConfigInput(self, arg0)	Specify whether or not wait until configuration message arrives to inputConfig Input.

Attributes:

initialConfig	Initial config to use when manipulating frames
inputConfig	Input ImageManipConfig message with ability to modify parameters in runtime Default queue is blocking with size 8
inputImage	Input image to be modified Default queue is blocking with size 8
out	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:

__init__(self)	
getCropXMax(self)	**returns** Bottom right X coordinate of crop region
getCropXMin(self)	**returns** Top left X coordinate of crop region
getCropYMax(self)	**returns** Bottom right Y coordinate of crop region
getCropYMin(self)	**returns** Top left Y coordinate of crop region
getResizeHeight(self)	**returns** Output image height
getResizeWidth(self)	**returns** Output image width

continues on next page

Table 61 – continued from previous page

`isResizeThumbnail`(self)	**returns** True if resize thumbnail mode is set, false otherwise
`setCenterCrop`(self, ratio, whRatio)	Specifies a centered crop.
`setCropRect`(self, xmin, ymin, xmax, xmax)	Specifies crop with rectangle with normalized values (0..1)
`setCropRotatedRect`(self, rr, normalizedCoords)	Specifies crop with rotated rectangle.
`setFrameType`(self, name)	Specify output frame type.
`setHorizontalFlip`(self, flip)	Specify horizontal flip
`setKeepAspectRatio`(self, keep)	Specifies to whether to keep aspect ratio or not
`setResize`(self, w, h)	Specifies output image size.
`setResizeThumbnail`(self, w, h, bgRed, …)	Specifies output image size.
`setReusePreviousImage`(self, reuse)	Instruct ImageManip to not remove current image from its queue and use the same for next message.
`setRotationDegrees`(self, deg)	Specifies clockwise rotation in degrees
`setRotationRadians`(self, rad)	Specifies clockwise rotation in radians
`setSkipCurrentImage`(self, skip)	Instructs ImageManip to skip current image and wait for next in queue.
`setWarpBorderFillColor`(self, red, green, blue)	Specifies fill color for border pixels.
`setWarpBorderReplicatePixels`(self)	Specifies that warp replicates border pixels
`setWarpTransformFourPoints`(self, pt, …)	Specifies warp by suppling 4 points in either absolute or normalized coordinates
`setWarpTransformMatrix3x3`(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, *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

`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 streched 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 suppling 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.**ImgDetections**

Bases: *depthai.Buffer*

ImgDetections message. Carries normalized detection results

Methods:

__init__(self)	Construct ImgDetections message

Attributes:

detections	Detections

__init__(*self*: depthai.ImgDetections) → None

Construct ImgDetections message

property detections

Detections

class depthai.**ImgFrame**

Bases: *depthai.Buffer*

ImgFrame message. Carries image data and metadata.

Classes:

Specs	
Type	Members:

Methods:

__init__(self)	
getCategory(self)	Retrievies image category
getCvFrame(self)	Returns BGR or grayscale frame compatible with use in other opencv functions
getFrame(self, copy)	Returns numpy array with shape as specified by width, height and type
getHeight(self)	Retrievies image height in pixels
getInstanceNum(self)	Retrievies instance number
getSequenceNum(self)	Retrievies image sequence number
getTimestamp(self)	Retrievies image timestamp related to steady_clock / time.monotonic
getType(self)	Retrieves image type
getWidth(self)	Retrievies image width in pixels
setCategory(self, category)	Parameter `category`:
setFrame(self, array)	Copies array bytes to ImgFrame buffer
setHeight(self, height)	Specifies frame height

continues on next page

Table 67 – continued from previous page

setInstanceNum(self, instance)	Instance number relates to the origin of the frame (which camera)
setSequenceNum(self, seq)	Specifies sequence number
setTimestamp(self, timestamp)	Specifies current timestamp, related to steady_clock / time.monotonic
setType(self, type)	Specifies frame type, RGB, BGR, …
setWidth(self, width)	Specifies frame width

class Specs

Bases: `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:

BGR888i
BGR888p
BGRF16F16F16i
BGRF16F16F16p
BITSTREAM
GRAY8
GRAYF16
HDR

continues on next page

Table 70 – continued from previous page

LUT16
LUT2
LUT4
NONE
NV12
NV21
PACK10
PACK12
RAW10
RAW12
RAW14
RAW16
RAW8
RGB161616
RGB888i
RGB888p
RGBA8888
RGBF16F16F16i
RGBF16F16F16p
YUV400p
YUV420p
YUV422i
YUV422p
YUV444i
YUV444p
name
value

Methods:

__init__(self, value)

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.ImgFrame*) → None

getCategory(*self: depthai.ImgFrame*) → int
Retrievies 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
Retrievies image height in pixels

getInstanceNum(*self: depthai.ImgFrame*) → int
Retrievies instance number

getSequenceNum(*self: depthai.ImgFrame*) → int
Retrievies image sequence number

getTimestamp(*self: depthai.ImgFrame*) → datetime.timedelta
Retrievies image timestamp related to steady_clock / time.monotonic

getType (*self: depthai.ImgFrame*) → *depthai.RawImgFrame.Type*
Retrieves image type

getWidth (*self: depthai.ImgFrame*) → int
Retrievies 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

INFO
OFF
TRACE
WARN
name
value

Methods:

__init__(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>

__init__(*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:

__init__(self)

Attributes:

remaining
total
used

__init__(*self: depthai.MemoryInfo*) → None

property remaining

property total

property used

class depthai.**MobileNetDetectionNetwork**

Bases: *depthai.DetectionNetwork*

MobileNetDetectionNetwork node. Parses MobileNet results

Methods:

`__init__`(*args, **kwargs)	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:

`__init__`(*args, **kwargs)	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:

`Properties`	alias of `depthai.MonoCameraProperties`

Methods:

`__init__`(*args, **kwargs)	Initialize self.
`getBoardSocket`(self)	Retrieves which board socket to use
`getCamId`(self)	
`getFps`(self)	Get rate at which camera should produce frames
`getImageOrientation`(self)	Get camera image orientation
`getResolution`(self)	Get sensor resolution
`getResolutionHeight`(self)	Get sensor resolution height
`getResolutionSize`(self)	Get sensor resolution as size
`getResolutionWidth`(self)	Get sensor resolution width
`setBoardSocket`(self, boardSocket)	Specify which board socket to use
`setCamId`(self, arg0)	
`setFps`(self, fps)	Set rate at which camera should produce frames
`setImageOrientation`(self, imageOrientation)	Set camera image orientation
`setResolution`(self, resolution)	Set sensor resolution

Attributes:

`initialControl`	Initial control options to apply to sensor
`inputControl`	Input for CameraControl message, which can modify camera parameters in runtime Default queue is blocking with size 8
`out`	Outputs ImgFrame message that carries RAW8 encoded (grayscale) frame data.

Properties
alias of *depthai.MonoCameraProperties* **Classes:**

SensorResolution	Select the camera sensor resolution: 1280×720, 1280×800, 640×400

Methods:

__init__(*args, **kwargs)	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:

SensorResolution	Select the camera sensor resolution: 1280×720, 1280×800, 640×400

Methods:

__init__(*args, **kwargs)	Initialize self.

Attributes:

boardSocket
fps
initialControl
resolution

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:

THE_400_P
THE_720_P
THE_800_P

continues on next page

Table 87 – continued from previous page

name
value

Methods:

`__init__`(self, value)	

THE_400_P = <SensorResolution.THE_400_P: 2>

THE_720_P = <SensorResolution.THE_720_P: 0>

THE_800_P = <SensorResolution.THE_800_P: 1>

__init__(*self: depthai.MonoCameraProperties.SensorResolution*, *value: int*) → None

property name

property value

__init__(**args*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

property boardSocket

property fps

property initialControl

property resolution

class depthai.**NNData**
Bases: `depthai.Buffer`

NNData message. Carries tensors and their metadata

Methods:

`__init__`(self)	Construct NNData message.
`getAllLayerNames`(self)	**returns** Names of all layers added
`getAllLayers`(self)	**returns** All layers and their information
`getFirstLayerFp16`(self)	Convinience function to retrieve float values from first layers FP16 tensor
`getFirstLayerInt32`(self)	Convinience function to retrieve INT32 values from first layers tensor
`getFirstLayerUInt8`(self)	Convinience function to retrieve U8 data from first layer
`getLayer`(self, name, tensor)	Retrieve layers tensor information
`getLayerDatatype`(self, name, datatype)	Retrieve datatype of a layers tensor
`getLayerFp16`(self, name)	Convinience function to retrieve float values from layers FP16 tensor
`getLayerInt32`(self, name)	Convinience function to retrieve INT32 values from layers tensor
`getLayerUInt8`(self, name)	Convinience function to retrieve U8 data from layer

continues on next page

Table 89 – continued from previous page

hasLayer(self, name)	Checks if given layer exists
setLayer(*args, **kwargs)	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]
Convinience function to retrieve float values from first layers FP16 tensor

Returns Float data

getFirstLayerInt32(*self: depthai.NNData*) → List[int]
Convinience function to retrieve INT32 values from first layers tensor

Returns INT32 data

getFirstLayerUInt8(*self: depthai.NNData*) → List[int]
Convinience 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 infromation 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]
Convinience 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]
Convinience 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]
Convinience 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:

Properties	alias of *depthai.NeuralNetworkProperties*

Methods:

`__init__`(*args, **kwargs)	Initialize self.
`getNumInferenceThreads`(self)	How many inference threads will be used to run the network
`setBlobPath`(self, path)	Load network blob into assets and use once pipeline is started.
`setNumInferenceThreads`(self, numThreads)	How many threads should the node use to run the network.
`setNumNCEPerInferenceThread`(self, …)	How many Neural Compute Engines should a single thread use for inference
`setNumPoolFrames`(self, numFrames)	Specifies how many frames will be avilable in the pool

Attributes:

`input`	Input message with data to be infered upon Default queue is blocking with size 5
`out`	Outputs NNData message that carries inference results
`passthrough`	Passthrough message on which the inference was performed.

Properties

alias of `depthai.NeuralNetworkProperties` **Methods:**

`__init__`(*args, **kwargs)	Initialize self.

Attributes:

`blobSize`
`blobUri`
`numFrames`
`numNCEPerThread`
`numThreads`

`__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 infered 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 avilable 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 identificator.

continues on next page

Table 97 – continued from previous page

Input
Output

Methods:

__init__(*args, **kwargs)	Initialize self.
getAssets(self)	Retrieves all nodes assets
getInputs(self)	Retrieves all nodes inputs
getName(self)	Retrieves nodes name
getOutputs(self)	Retrieves all nodes outputs

Attributes:

id	Id of node

class Connection

Bases: pybind11_builtins.pybind11_object

Connection between an Input and Output

Methods:

__init__(*args, **kwargs)	Initialize self.

Attributes:

inputId
inputName
outputId
outputName

__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:

__init__(*args, **kwargs)	Initialize self.

__init__(**args, **kwargs*)

Initialize self. See help(type(self)) for accurate signature.

class Input

Bases: `pybind11_builtins.pybind11_object`

Methods:

`__init__`(*args, **kwargs)	Initialize self.
`getBlocking`(self)	Get input queue behavior
`getQueueSize`(self)	Get input queue size.
`setBlocking`(self, blocking)	Overrides default input queue behavior.
`setQueueSize`(self, size)	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:

`__init__`(*args, **kwargs)	Initialize self.
`canConnect`(self, in)	Check if connection is possible
`getConnections`(self)	Retrieve all connections from this output
`link`(self, in)	Link current output to input.
`unlink`(self, in)	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:

Properties	alias of *depthai.ObjectTrackerProperties*

Methods:

__init__(*args, **kwargs)	Initialize self.
setDetectionLabelsToTrack(self, labels)	Specify detection labels to track.
setMaxObjectsToTrack(self, maxObjectsToTrack)	Specify maximum number of object to track.
setTrackerIdAssigmentPolicy(self, type)	Specify tracker ID assigment policy.
setTrackerThreshold(self, threshold)	Specify tracker threshold.
setTrackerType(self, type)	Specify tracker type algorithm.

Attributes:

inputDetectionFrame	Input ImgFrame message on which object detection was performed.
inputDetections	Input message with image detection from neural network.
inputTrackerFrame	Input ImgFrame message on which tracking will be performed.
out	Outputs Tracklets message that carries object tracking results.
passthroughDetectionFrame	Passthrough ImgFrame message on which object detection was performed.

continues on next page

Table 107 – continued from previous page

passthroughDetections	Passthrough image detections message from neural nework output.
passthroughTrackerFrame	Passthrough ImgFrame message on which tracking was performed.

Properties
alias of *depthai.ObjectTrackerProperties* **Methods:**

`__init__`(*args, **kwargs)	Initialize self.

Attributes:

`detectionLabelsToTrack`
`maxObjectsToTrack`
`trackerIdAssigmentPolicy`
`trackerThreshold`
`trackerType`

`__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 nework 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.

setTrackerIdAssigmentPolicy(*self: depthai.ObjectTracker*, *type: dai::TrackerIdAssigmentPolicy*) → None
Specify tracker ID assigment policy.

Parameter type: Tracker ID assigment 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:

`__init__`(*args, **kwargs)	Initialize self.

Attributes:

`detectionLabelsToTrack`
`maxObjectsToTrack`
`trackerIdAssigmentPolicy`
`trackerThreshold`
`trackerType`

__init__(**args*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

property detectionLabelsToTrack

property maxObjectsToTrack

property trackerIdAssigmentPolicy

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:

`VERSION_2020_1`
`VERSION_2020_2`
`VERSION_2020_3`
`VERSION_2020_4`
`VERSION_2021_1`

continues on next page

Table 112 – continued from previous page

VERSION_2021_2
VERSION_2021_3

Classes:

Version	OpenVINO Version supported version information

Methods:

__init__(*args, **kwargs)	Initialize self.
areVersionsBlobCompatible(v1, v2)	Checks whether two blob versions are compatible
getBlobLatestSupportedVersion(majorVersion, ...)	Returns latest potentionally supported version by a given blob version.
getBlobSupportedVersions(majorVersion, ...)	Returns a list of potentionally supported versions for a specified blob major and minor versions.
getVersionName(version)	Returns string representation of a given version
getVersions()	**returns** Supported versions
parseVersionName(versionString)	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:

VERSION_2020_1
VERSION_2020_2
VERSION_2020_3
VERSION_2020_4
VERSION_2021_1
VERSION_2021_2
VERSION_2021_3
name
value

Methods:

__init__(self, value)

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, majorVersion: int) -> dai::Op
Returns latest potentionally supported version by a given blob version.

Parameter majorVersion: Major version from OpenVINO blob

Parameter minorVersion: Minor version from OpenVINO blob

Returns Latest potentionally supported version

OpenVINO.getBlobSupportedVersions(majorVersion: int, majorVersion: int) -> List[dai::Op
Returns a list of potentionally 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 potentionally 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:

__init__(self)	Constructs a new pipeline
createColorCamera(self)	
createImageManip(self)	
createMobileNetDetectionNetwork(self)	
createMobileNetSpatialDetectionNetwork(self)	
createMonoCamera(self)	
createNeuralNetwork(self)	
createObjectTracker(self)	
createSPIOut(self)	
createSpatialLocationCalculator(self)	
createStereoDepth(self)	
createSystemLogger(self)	
createVideoEncoder(self)	
createXLinkIn(self)	
createXLinkOut(self)	
createYoloDetectionNetwork(self)	
createYoloSpatialDetectionNetwork(self)	
getAllAssets(self)	Get assets on the pipeline includes nodes assets
getAllNodes(*args, **kwargs)	Overloaded function.
getAssetManager(*args, **kwargs)	Overloaded function.
getConnectionMap(self)	Get a reference to internal connection representation
getConnections(self)	Get all connections
getGlobalProperties(self)	**returns** Global properties of current pipeline
getNode(*args, **kwargs)	Overloaded function.
getNodeMap(self)	Get a reference to internal node map
link(self, arg0, arg1)	Link output to an input.
remove(self, node)	Removes a node from pipeline
setOpenVINOVersion(self, version)	Set a specific OpenVINO version to use with this pipeline

continues on next page

Table 117 – continued from previous page

unlink(self, arg0, arg1)	Unlink output from an input.

__init__(*self*: depthai.Pipeline) → None
Constructs a new pipeline

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

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:

AntiBandingMode	Members:
AutoFocusMode	Members:
AutoWhiteBalanceMode	Members:
EffectMode	Members:
SceneMode	Members:

Methods:

__init__(*args, **kwargs)	Initialize self.

Attributes:

autoFocusMode
cmdMask
lensPosition

class AntiBandingMode

Bases: `pybind11_builtins.pybind11_object`

Members:

OFF

MAINS_50_HZ

MAINS_60_HZ

AUTO

Attributes:

AUTO
MAINS_50_HZ
MAINS_60_HZ
OFF
name
value

Methods:

__init__(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:

AUTO
CONTINUOUS_PICTURE
CONTINUOUS_VIDEO
EDOF
MACRO
OFF
name
value

Methods:

__init__(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:

AUTO	
CLOUDY_DAYLIGHT	
DAYLIGHT	
FLUORESCENT	
INCANDESCENT	
OFF	
SHADE	
TWILIGHT	
WARM_FLUORESCENT	
name	
value	

Methods:

__init__(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:

AQUA	
BLACKBOARD	
MONO	
NEGATIVE	
OFF	
POSTERIZE	
SEPIA	
SOLARIZE	
WHITEBOARD	
name	
value	

Methods:

__init__(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:

ACTION
BARCODE
BEACH
CANDLELIGHT
FACE_PRIORITY
FIREWORKS
LANDSCAPE
NIGHT
NIGHT_PORTRAIT
PARTY
PORTRAIT
SNOW
SPORTS
STEADYPHOTO
SUNSET
THEATRE
UNSUPPORTED
name
value

Methods:

__init__(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__(**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 Specs

Bases: `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:

BGR888i
BGR888p
BGRF16F16F16i
BGRF16F16F16p
BITSTREAM
GRAY8
GRAYF16
HDR
LUT16
LUT2

continues on next page

Table 155 – continued from previous page

LUT4
NONE
NV12
NV21
PACK10
PACK12
RAW10
RAW12
RAW14
RAW16
RAW8
RGB161616
RGB888i
RGB888p
RGBA8888
RGBF16F16F16i
RGBF16F16F16p
YUV400p
YUV420p
YUV422i
YUV422p
YUV444i
YUV444p
name
value

Methods:

__init__(self, value)

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:

`__init__`(*args, **kwargs)	Initialize self.

Attributes:

`tracklets`

`__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:

`__init__`(*args, **kwargs)	Overloaded function.
`area`(self)	Area (width*height) of the rectangle
`bottomRight`(self)	The bottom-right corner
`contains`(self, arg0)	Checks whether the rectangle contains the point.
`denormalize`(self, width, height)	Denormalize rectangle.
`empty`(self)	True if rectangle is empty.
`isNormalized`(self)	Whether rectangle is normalized (coordinates in [0,1] range) or not.
`normalize`(self, width, height)	Normalize rectangle.
`size`(self)	Size (width, height) of the rectangle
`topLeft`(self)	The top-left corner.

Attributes:

`height`
`width`
`x`
`y`

`__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*) → 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:

`__init__`(*args, **kwargs)	Initialize self.
`setBusId`(self, id)	Specifies SPI Bus number to use
`setStreamName`(self, name)	Specifies stream name over which the node will send data

Attributes:

`input`	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:

`__init__`(*args, **kwargs)	Overloaded function.

Attributes:

`height`
`width`

`__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:

Properties	alias of *depthai.SpatialDetectionNetworkProperties*

Methods:

__init__(*args, **kwargs)	Initialize self.
setBoundingBoxScaleFactor(self, scaleFactor)	Specifies scale factor for detected bounding boxes.
setDepthLowerThreshold(self, lowerThreshold)	Specifies lower threshold in milimeters for depth values which will used to calculate spatial data
setDepthUpperThreshold(self, upperThreshold)	Specifies upper threshold in milimeters for depth values which will used to calculate spatial data

Attributes:

boundingBoxMapping	Outputs mapping of detected bounding boxes relative to depth map
input	Input message with data to be infered upon Default queue is blocking with size 5
inputDepth	Input message with depth data used to retrieve spatial information about detected object Default queue is non-blocking with size 4
out	Outputs ImgDetections message that carries parsed detection results.
passthrough	Passthrough message on which the inference was performed.
passthroughDepth	Passthrough message for depth frame on which the spatial location calculation was performed.

Properties

alias of *depthai.SpatialDetectionNetworkProperties* **Methods:**

__init__(*args, **kwargs)	Initialize self.

Attributes:

depthThresholds
detectedBBScaleFactor

__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 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.

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 milimeters 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 milimeters 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:

__init__(*args, **kwargs)	Initialize self.

Attributes:

depthThresholds
detectedBBScaleFactor

__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:

`Properties`	alias of `depthai.SpatialLocationCalculatorProperties`

Methods:

`__init__`(*args, **kwargs)	Initialize self.
`setWaitForConfigInput`(self, wait)	Specify whether or not wait until configuration message arrives to inputConfig Input.

Attributes:

`initialConfig`	Initial config to use when calculating spatial location data.
`inputConfig`	Input SpatialLocationCalculatorConfig message with ability to modify parameters in runtime.
`inputDepth`	Input message with depth data used to retrieve spatial information about detected object.
`out`	Outputs SpatialLocationCalculatorData message that carries spatial location results.

continues on next page

Table 186 – continued from previous page

passthroughDepth	Passthrough message on which the calculation was performed.

Properties
alias of *depthai.SpatialLocationCalculatorProperties* **Methods:**

`__init__`(*args, **kwargs)	Initialize self.

Attributes:

`inputConfigSync`
`roiConfig`

`__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:

`__init__`(self)	
addROI(self, ROI)	Add a new ROI to configuration data.
getConfigData(self)	Retrieve configuration data for SpatialLocationCalculator
setROIs(self, ROIs)	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:

`__init__`(self)	
`getSpatialLocations`(self)	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:

`__init__`(*args, **kwargs)	Initialize self.

Attributes:

`inputConfigSync`
`roiConfig`

`__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:

`__init__`(self)

Attributes:

`config`
`depthAverage`
`depthAveragePixelCount`
`spatialCoordinates`

`__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:

Properties	alias of *depthai.StereoDepthProperties*

Methods:

__init__(*args, **kwargs)	Initialize self.
loadCalibrationData(self, data)	Specify calibration data as a vector of bytes
loadCalibrationFile(self, path)	Specify local filesystem path to the calibration file
setConfidenceThreshold(self, confThr)	Confidence threshold for disparity calculation
setEmptyCalibration(self)	Specify that a passthrough/dummy calibration should be used, when input frames are already rectified (e.g.
setExtendedDisparity(self, enable)	Disparity range increased from 96 to 192, combined from full resolution and downscaled images.
setInputResolution(self, width, height)	Specify input resolution size
setLeftRightCheck(self, enable)	Computes and combines disparities in both L-R and R-L directions, and combine them.
setMedianFilter(self, median)	Parameter `median`:
setOutputDepth(self, enable)	Enable outputting 'depth' stream (converted from disparity).
setOutputRectified(self, enable)	Enable outputting rectified frames.
setRectifyEdgeFillColor(self, color)	Fill color for missing data at frame edges
setRectifyMirrorFrame(self, enable)	Mirror rectified frames
setSubpixel(self, enable)	Computes disparity with sub-pixel interpolation (5 fractional bits).

Attributes:

depth	Outputs ImgFrame message that carries RAW16 encoded (0..65535) depth data in millimeters.
disparity	Outputs ImgFrame message that carries RAW8 encoded (0..96 or 0..192 for Extended mode) disparity data.
left	Input for left ImgFrame of left-right pair
rectifiedLeft	Outputs ImgFrame message that carries RAW8 encoded (grayscale) rectified frame data.
rectifiedRight	Outputs ImgFrame message that carries RAW8 encoded (grayscale) rectified frame data.
right	Input for right ImgFrame of left-right pair
syncedLeft	Passthrough ImgFrame message from 'left' Input.
syncedRight	Passthrough ImgFrame message from 'right' Input.

Properties

alias of *depthai.StereoDepthProperties* **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`

`__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 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.

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:

__init__(*args, **kwargs)	Initialize self.
setRate(self, hz)	Specify logging rate, at which messages will be sent to out output

Attributes:

out	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:

__init__(*args, **kwargs)	Initialize self.

Attributes:

rateHz

__init__(**args*, ***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

property rateHz

class depthai.**TensorInfo**

Bases: `pybind11_builtins.pybind11_object`

Classes:

DataType	Members:
StorageOrder	Members:

Methods:

`__init__`(self)

Attributes:

`dataType`
`dims`
`name`
`numDimensions`
`offset`
`order`
`strides`

class DataType

Bases: `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:

C
CHW
CN
CWH
H
HCW
HWC
NC
NCHW
NHCW
NHWC
W
WCH
WHC
name
value

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.**TrackerIdAssigmentPolicy**

Bases: pybind11_builtins.pybind11_object

Members:

UNIQUE_ID

SMALLEST_ID

Attributes:

SMALLEST_ID
UNIQUE_ID
name
value

Methods:

__init__(self, value)

SMALLEST_ID = <TrackerIdAssigmentPolicy.SMALLEST_ID: 1>

UNIQUE_ID = <TrackerIdAssigmentPolicy.UNIQUE_ID: 0>

__init__(*self: depthai.TrackerIdAssigmentPolicy*, *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:

tracklets	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:

Properties	alias of *depthai.VideoEncoderProperties*

Methods:

__init__(*args, **kwargs)	Initialize self.
getBitrate(self)	Get bitrate in bps
getBitrateKbps(self)	Get bitrate in kbps
getFrameRate(self)	Get frame rate
getHeight(self)	Get input height
getKeyframeFrequency(self)	Get keyframe frequency
getLossless(self)	Get lossless mode.
getNumBFrames(self)	Get number of B frames
getNumFramesPool(self)	Get number of frames in pool
getProfile(self)	Get profile
getQuality(self)	Get quality
getRateControlMode(self)	Get rate control mode
getSize(self)	Get input size
getWidth(self)	Get input width
setBitrate(self, bitrateKbps)	Set output bitrate in bps.

continues on next page

Table 237 – continued from previous page

`setBitrateKbps`(self, bitrateKbps)	Set output bitrate in kbps.
`setDefaultProfilePreset`(*args, **kwargs)	Overloaded function.
`setFrameRate`(self, frameRate)	Sets expected frame rate
`setKeyframeFrequency`(self, freq)	Set keyframe frequency.
`setLossless`(self, arg0)	Set lossless mode.
`setNumBFrames`(self, numBFrames)	Set number of B frames to be inserted
`setNumFramesPool`(self, frames)	Set number of frames in pool
`setProfile`(*args, **kwargs)	Overloaded function.
`setQuality`(self, quality)	Set quality
`setRateControlMode`(self, mode)	Set rate control mode

Attributes:

`bitstream`	Outputs ImgFrame message that carries BITSTREAM encoded (MJPEG, H264 or H265) frame data.
`input`	Input for NV12 ImgFrame to be encoded Default queue is blocking with size set by 'setNumFramesPool' (4).

Properties

alias 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:

`__init__`(*args, **kwargs)	Initialize self.

Attributes:

`bitrate`
`height`
`keyframeFrequency`
`maxBitrate`
`numBFrames`
`numFramesPool`
`profile`
`quality`
`rateCtrlMode`
`width`

`__init__`(**args, **kwargs*)

Initialize self. See help(type(self)) for accurate signature.

property bitstream

Outputs 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*) → 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

class depthai.**VideoEncoderProperties**
Bases: `pybind11_builtins.pybind11_object`

Specify VideoEncoder options such as profile, bitrate, . . .

Classes:

Profile	Encoding profile, H264, H265 or MJPEG
RateControlMode	Rate control mode specifies if constant or variable bitrate should be used (H264 / H265)

Methods:

__init__(*args, **kwargs)	Initialize self.

Attributes:

bitrate
height
keyframeFrequency
maxBitrate
numBFrames
numFramesPool
profile
quality
rateCtrlMode
width

class Profile

Bases: `pybind11_builtins.pybind11_object`

Encoding profile, H264, H265 or MJPEG

Members:

H264_BASELINE

H264_HIGH

H264_MAIN

H265_MAIN

MJPEG

Attributes:

H264_BASELINE
H264_HIGH
H264_MAIN
H265_MAIN
MJPEG
name
value

Methods:

__init__(self, value)

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:

`__init__`(*args, **kwargs)	Overloaded function.
`getAllConnectedDevices`(state)	
`getDeviceByMxId`(mxId, state)	
`getFirstDevice`(state)	

__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:

`X_LINK_ANY_STATE`
`X_LINK_BOOTED`
`X_LINK_BOOTLOADER`
`X_LINK_UNBOOTED`
`name`
`value`

Methods:

`__init__`(self, value)

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:

__init__(*args, **kwargs)	Initialize self.
getMaxDataSize(self)	Get maximum messages size in bytes
getNumFrames(self)	Get number of frames in pool
getStreamName(self)	Get stream name
setMaxDataSize(self, maxDataSize)	Set maximum message size it can receive
setNumFrames(self, numFrames)	Set number of frames in pool for sending messages forward
setStreamName(self, streamName)	Specifies XLink stream name to use.

Attributes:

out	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:

`__init__`(*args, **kwargs)	Initialize self.
`getFpsLimit`(self)	Get rate limit in messages per second
`getMetadataOnly`(self)	Get whether to transfer only messages attributes and not buffer data
`getStreamName`(self)	Get stream name
`setFpsLimit`(self, fpsLimit)	Specifies a message sending limit.
`setMetadataOnly`(self, arg0)	Specify whether to transfer only messages attributes and not buffer data
`setStreamName`(self, streamName)	Specifies XLink stream name to use.

Attributes:

`input`	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:

__init__(*args, **kwargs)	Initialize self.
setAnchorMasks(self, anchorMasks, List[int]])	Set anchor masks
setAnchors(self, anchors)	Set anchors
setCoordinateSize(self, coordinates)	Set coordianate size
setIouThreshold(self, thresh)	Set Iou threshold
setNumClasses(self, numClasses)	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:

__init__(*args, **kwargs)	Initialize self.
setAnchorMasks(self, anchorMasks, List[int]])	Set anchor masks
setAnchors(self, anchors)	Set anchors
setCoordinateSize(self, coordinates)	Set coordianate size
setIouThreshold(self, thresh)	Set Iou threshold
setNumClasses(self, numClasses)	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 coordianate 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 TrackerIdAssigmentPolicy
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 specificied 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 specificied 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 immidiately 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 occured
Parameters
- `timeout`: Duration for which the function should block
- `[out] hasTimedout`: Outputs true if timeout occured, 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 occured
Parameters
- `timeout`: Duration for which the function should block
- `[out] hasTimedout`: Outputs true if timeout occured, 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 &*hasTimedout*)
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 occured, 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 occured, 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> *&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**(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**(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.

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 suppling 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 streched 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**
Retrievies image timestamp related to steady_clock / time.monotonic

unsigned int **getInstanceNum**() **const**
Retrievies instance number

unsigned int **getCategory**() **const**
Retrievies image category

unsigned int **getSequenceNum**() **const**
Retrievies image sequence number

unsigned int **getWidth**() **const**
Retrievies image width in pixels

unsigned int **getHeight**() **const**
Retrievies 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 infromation 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**
Convinience 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**
Convinience 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**
Convinience function to retrieve INT32 values from layers tensor
Return INT32 data
Parameters
- `name`: Name of the layer

std::vector<std::uint8_t> **getFirstLayerUInt8**() **const**
Convinience function to retrieve U8 data from first layer
Return U8 binary data

std::vector<float> **getFirstLayerFp16**() **const**
Convinience function to retrieve float values from first layers FP16 tensor
Return Float data

std::vector<std::int32_t> **getFirstLayerInt32**() **const**
Convinience 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 identificator. 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 potentionally supported versions for a specified blob major and minor versions.
Return Vector of potentionally 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 potentionally supported version by a given blob version.
Return Latest potentionally 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* openvino 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 downscaled 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 prefered 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*,

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*, fa

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 infered 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*, fa
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 avilable 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 infered 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 **setTrackerIdAssigmentPolicy**(*TrackerIdAssigmentPolicy* *type*)
Specify tracker ID assigment policy.
Parameters
- `type`: Tracker ID assigment policy.

Public Members

Input **inputTrackerFrame** = {*this, "inputTrackerFrame", Input::Type::SReceiver, false, 4, {{*DatatypeEnum*::*Img*
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*
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*::*ImgDetect*

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, {{*DatatypeE*

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, {{*Data*

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*::

Passthrough image detections message from neural nework 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 milimeters for depth values which will 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 milimeters for depth values which will 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 infered 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*::*Spati*

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*, f

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*::*SpatialLocationCalc*

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*, f

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 transfered 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 *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.

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 coordianate 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 network with spatial location data.

Public Functions

void **setNumClasses**(**const** int *numClasses*)
Set num classes.

void **setCoordinateSize**(**const** int *coordinates*)
Set coordianate 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

INDEX

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z