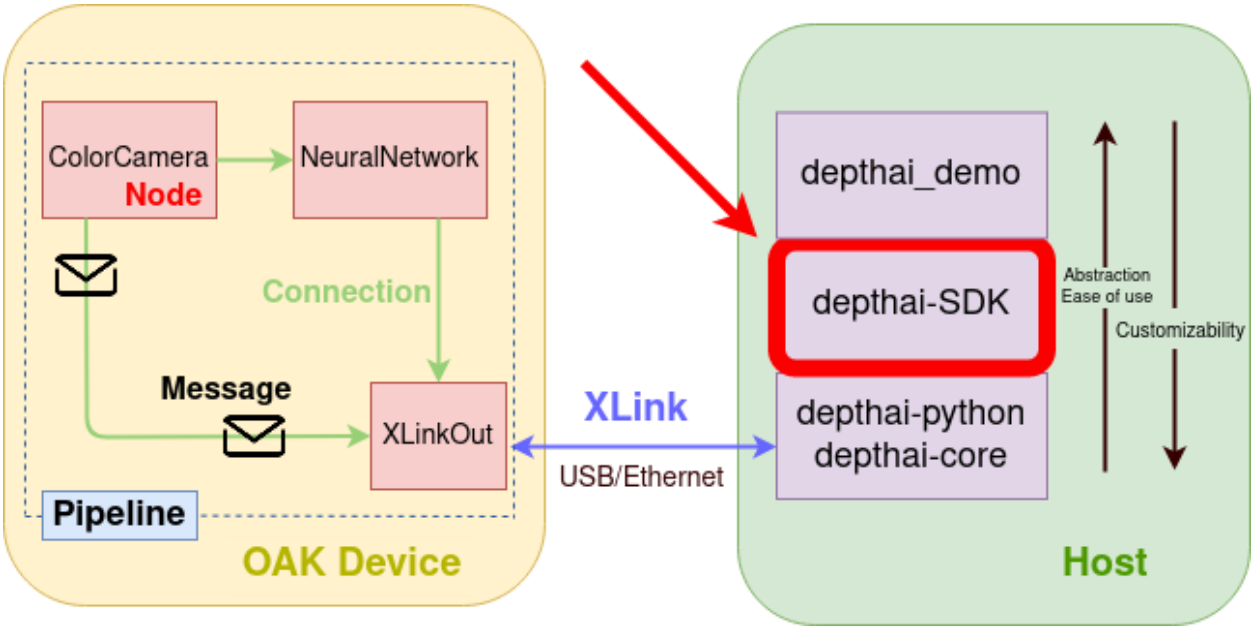
DepthAI SDK Docs

Release 1.0.0

Luxonis

Jan 23, 2023

HOME:



DepthAI SDK is a Python package, containing convenience classes and functions that help in most common tasks while using DepthAI API. It is created on top of the regular DepthAI API. The package was created to make your use of DepthAI easier. It consists of managers, which handle different aspects of development:

depthai_sdk.managers.PipelineManager	Helps in setting up processing pipeline
depthai_sdk.managers.NNetManager	Helps in setting up neural networks
depthai_sdk.managers.PreviewManager	Helps in displaying preview from OAK cameras
depthai_sdk.managers.EncodingManager	Helps in creating videos from OAK cameras
depthai_sdk.managers.BlobManager	Helps in downloading neural networks as MyriadX blobs
depthai_sdk.fps	For FPS calculations
depthai_sdk.previews	For frame handling
depthai_sdk.utils	For various most-common tasks

CHAPTER

ONE

INSTALLATION

To install this package, run the following command in your terminal window

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

Warning: If you're using Raspberry Pi, providing a Pi Wheels extra package url can significantly speed up the installation process by providing prebuilt binaries for OpenCV

```
python3 -m pip install --extra-index-url https://www.piwheels.org/simple/ depthai-sdk
```

To help you understand and learn how to use the package and how the manager classes work, we created a few simple tutorials:

- *Getting started with DepthAI SDK*
- *Pipeline manager*
- *Preview manager*
- *Encoding manager*
- *Blob manager*
- *Nnet manager*

We recommend that you start with the *Getting started with DepthAI SDK*

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

1.1 Getting started with DepthAI SDK

In this tutorial, we'll show you how to use DepthAI SDK for a couple of basic use cases, that can give you an overall idea how to use it and in which cases it might be useful.

1.1.1 What is DepthAI SDK?

DepthAI SDK was created on top of the regular DepthAI API. Originally, it was a part of the demo script, but over time it evolved to become a package containing many convenience methods and classes that aim to help in development process with OAK cameras.

Package is mainly made of **managers**, which handle different aspects of development:

depthai_sdk.managers.PipelineManager	Helps in setting up processing pipeline
depthai_sdk.managers.NNetManager	Helps in setting up neural networks
depthai_sdk.managers.PreviewManager	Helps in displaying preview from OAK cameras
depthai_sdk.managers.EncodingManager	Helps in creating videos from OAK cameras
depthai_sdk.managers.BlobManager	Helps in downloading neural networks as MyriadX blobs
depthai_sdk.fps	For FPS calculations
depthai_sdk.previews	For frame handling
depthai_sdk.utils	For various most-common tasks

In some places, code is also adjusted for modifications - e.g. you can set up a custom handler file for neural network or pass a callback argument to a function to perform additional modifications

1.1.2 Example usages

The original "user" of this SDK was the demo script, where you can see how the SDK is used. Below, you can find a list of other projects that also use the SDK and are available to use as a reference

- https://github.com/luxonis/depthai-experiments/tree/master/gen2-human-pose
- https://github.com/luxonis/depthai-experiments/tree/master/gen2-road-segmentation
- https://github.com/luxonis/depthai-experiments/tree/master/gen2-people-counter

1.1.3 Installation

To install this package, run the following command in your terminal window

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

Warning: If you're using Raspberry Pi, providing a Pi Wheels extra package url can significantly speed up the installation process by providing prebuilt binaries for OpenCV

```
python3 -m pip install --extra-index-url https://www.piwheels.org/simple/ depthai-sdk
```

1.1.4 Learn more

To see more details and examples on how the manager classes works specifically, we created a few simple tutorials:

- *Pipeline manager*
- *Preview manager*
- *Encoding manager*
- *Blob manager*
- *Nnet manager*

For more in-depth informations about the classes and methods, please visit *DepthAI SDK API*

1.1.5 Cookbook

Below you can find various basic usages of DepthAI SDK that can be used as a starting point.

Preview color camera

```
from depthai_sdk import Previews
from depthai_sdk.managers import PipelineManager, PreviewManager
import depthai as dai
import cv2

pm = PipelineManager()
pm.createColorCam(xout=True)

with dai.Device(pm.pipeline) as device:
    pv = PreviewManager(display=[Previews.color.name])
    pv.createQueues(device)

    while True:
        pv.prepareFrames()
        pv.showFrames()

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

Preview color and mono cameras

```
from depthai_sdk import Previews
from depthai_sdk.managers import PipelineManager, PreviewManager
import depthai as dai
import cv2

pm = PipelineManager()
pm.createColorCam(xout=True)
pm.createLeftCam(xout=True)
pm.createRightCam(xout=True)

```

(continues on next page)

(continued from previous page)

```
with dai.Device(pm.pipeline) as device:
    pv = PreviewManager(display=[Previews.color.name, Previews.left.name, Previews.right.name])
    pv.createQueues(device)

    while True:
        pv.prepareFrames()
        pv.showFrames()

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

Run MobilenetSSD on color camera

```
from depthai_sdk import Previews
from depthai_sdk.managers import PipelineManager, PreviewManager, NNetManager, BlobManager
import depthai as dai
import cv2

pm = PipelineManager()
pm.createColorCam(xout=True)

bm = BlobManager(zooName="mobilenet-ssd")
nm = NNetManager(inputSize=(300, 300), nnFamily="mobilenet")
nn = nm.createNN(pipeline=pm.pipeline, nodes=pm.nodes, source=Previews.color.name,
                 blobPath=bm.getBlob(shaves=6, openvinoVersion=pm.pipeline.getOpenVINOVersion()))
pm.addNn(nn)

with dai.Device(pm.pipeline) as device:
    pv = PreviewManager(display=[Previews.color.name])
    pv.createQueues(device)
    nm.createQueues(device)
    nnData = []

    while True:
        pv.prepareFrames()
        inNn = nm.outputQueue.tryGet()

        if inNn is not None:
            nnData = nm.decode(inNn)

        nm.draw(pv, nnData)
        pv.showFrames()

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

Run face-detection-retail-0004 on left camera

```
from depthai_sdk import Previews
from depthai_sdk.managers import PipelineManager, PreviewManager, NNetManager, BlobManager
import depthai as dai
import cv2

pm = PipelineManager()
pm.createLeftCam(xout=True)

bm = BlobManager(zooName="face-detection-retail-0004")
nm = NNetManager(inputSize=(300, 300), nnFamily="mobilenet")
nn = nm.createNN(pipeline=pm.pipeline, nodes=pm.nodes, source=Previews.left.name,
                 blobPath=bm.getBlob(shaves=6, openvinoVersion=pm.pipeline.getOpenVINOVersion()))
pm.addNn(nn)

with dai.Device(pm.pipeline) as device:
    pv = PreviewManager(display=[Previews.left.name])
    pv.createQueues(device)
    nm.createQueues(device)
    nnData = []

    while True:
        pv.prepareFrames()
        inNn = nm.outputQueue.tryGet()

        if inNn is not None:
            nnData = nm.decode(inNn)

        nm.draw(pv, nnData)
        pv.showFrames()

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

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

1.2 Pipeline manager

`PipelineManager` is the first class that we will learn how to use as it is the one that is goes hand in hand with every other class. It is created with the purpose to help you with creating and setting up processing pipelines. In this tutorial bellow we will see and learn how to declare and use them.

1.2.1 Getting started

Before we begin we must first import `pipeline_manager` from `DepthAI_SDK`. After that we will initialize the pipeline and define its sources.

```
from depthai_sdk.managers import PipelineManager
import depthai as dai


# declaring the pipeline
pm = PipelineManager()

# after the declaration, we define it's sources
pm.createColorCam(xout=True)

# connecting to device
with dai.Device(pm.pipeline) as device:
    # pipeline is created and the device is connected
    print("Successfully connected to the device!")
```

We successfully created and initialized the pipeline. If everything was setup correctly, you should receive a message in your terminal, that will inform you that the connecting was successful.

```
Successfully connected to the device!
```

But the above code currently one has one source as it's stream. We can initialize more sources in one pipeline.

```
from depthai_sdk.managers import PipelineManager
import depthai as dai


# declaring the pipeline
pm = PipelineManager()

# after the declaration, we define it's sources

# color camera
pm.createColorCam(xout=True)

# left camera
pm.createLeftCam(xout=True)

# right camera
pm.createRightCam(xout=True)

# depth
pm.createDepth(useDepth=True)

# connecting to device
with dai.Device(pm.pipeline) as device:
    # pipeline is created and the device is connected
    print("Successfully connected to the device!")
```

We now declared more then one source in the pipeline. To fully use the pipeline, you can use it with `PreviewManager` to

see the streams or `EncodingManager` to save streams to files. As you can see above we also added another argument to the color camera stream, called `previewSize` which will resize the stream to wanted ratio (height x width). All sources have many more arguments, `xout` will help us in the next tutorial where we will learn about the `PreviewManager`. In the above example we also declared a `Depth` source. We gave it `useDepth` as an argument which will create a queue for depth frames. If we wish to use a different queue for depth, we can change this argument to:

- `useDisparity` to use disparity frames,
- `useRectifiedLeft` for rectified left frames,
- and `useRectifiedRigh` for rectified right frames.

Of course these are not the only arguments that you can use in the `PipelineManager`.

class depthai_sdk.managers.**PipelineManager**

Manager class handling different `depthai.Pipeline` operations. Most of the functions wrap up nodes creation and connection logic onto a set of convenience functions.

__init__(*openvinoVersion=None*, *poeQuality=100*, *lowCapabilities=False*, *lowBandwidth=False*)

pipeline

Ready to use requested pipeline. Can be passed to `depthai.Device` to start execution

Type
depthai.Pipeline

nodes

Contains all nodes added to the *pipeline* object, can be used to conveniently access nodes by their name

Type
types.SimpleNamespace

openvinoVersion = None

OpenVINO version which will be used in pipeline

Type
depthai.OpenVINO.Version

poeQuality = None

PoE encoding quality, can decrease frame quality but decrease latency

Type
int, Optional

lowBandwidth = False

If set to `True`, manager will MJPEG-encode the packets sent from device to host to lower the bandwidth usage. **Can break** if more than 3 encoded outputs requested

Type
bool

lowCapabilities = False

If set to `True`, manager will try to optimize the pipeline to reduce the amount of host-side calculations (useful for RPi or other embedded systems)

Type
bool

setNnManager(*nnManager*)

Assigns NN manager. It also syncs the pipeline versions between those two objects

Parameters
nnManager (depthai_sdk.managers.NNetManager) – NN manager instance

createDefaultQueues(*device*)

Creates default queues for config updates

Parameters
device (*depthai.Device*) – Running device instance

closeDefaultQueues()

Creates default queues for config updates

Parameters
device (*depthai.Device*) – Running device instance

createColorCam(*previewSize=None*, *res=<SensorResolution.THE_1080_P: 0>*, *fps=30*, *fullFov=True*, *orientation=None*, *colorOrder=<ColorOrder.BGR: 0>*, *xout=False*, *xoutVideo=False*, *xoutStill=False*, *control=True*, *pipeline=None*, *args=None*)

Creates depthai.node.ColorCamera node based on specified attributes

Parameters

- **previewSize** (*tuple, Optional*) – Size of the preview - (width, height)
- **res** (*depthai.ColorCameraProperties.SensorResolution, Optional*) – Camera resolution to be used
- **fps** (*int, Optional*) – Camera FPS set on the device. Can limit / increase the amount of frames produced by the camera
- **fullFov** (*bool, Optional*) – If set to True, full frame will be scaled down to nn size. If to False, it will first center crop the frame to meet the NN aspect ratio and then scale down the image.
- **orientation** (*depthai.CameraImageOrientation, Optional*) – Custom camera orientation to be set on the device
- **colorOrder** (*depthai.ColorCameraProperties, Optional*) – Color order to be used
- **xout** (*bool, Optional*) – If set to True, a dedicated depthai.node.XLinkOut will be created for this node
- **xoutVideo** (*bool, Optional*) – If set to True, a dedicated depthai.node.XLinkOut will be created for *video* output of this node
- **xoutStill** (*bool, Optional*) – If set to True, a dedicated depthai.node.XLinkOut will be created for *still* output of this node
- **args** (*Object, Optional*) – Arguments from the ArgsManager

Return type
ColorCamera

createLeftCam(*res=None*, *fps=30*, *orientation=None*, *xout=False*, *control=True*, *pipeline=None*, *args=None*)

Creates depthai.node.MonoCamera node based on specified attributes, assigned to depthai.CameraBoardSocket.LEFT

Parameters

- **res** (*depthai.MonoCameraProperties.SensorResolution, Optional*) – Camera resolution to be used

- **fps** (*int, Optional*) – Camera FPS set on the device. Can limit / increase the amount of frames produced by the camera
- **orientation** (*depthai.CameraImageOrientation, Optional*) – Custom camera orientation to be set on the device
- **xout** (*bool, Optional*) – If set to `True`, a dedicated `depthai.node.XLinkOut` will be created for this node
- **args** (*Object, Optional*) – Arguments from the ArgsManager

Return type
MonoCamera

createRightCam(*res=None*, *fps=30*, *orientation=None*, *xout=False*, *control=True*, *pipeline=None*, *args=None*)

Creates `depthai.node.MonoCamera` node based on specified attributes, assigned to `depthai.CameraBoardSocket.RIGHT`

Parameters

- **res** (*depthai.MonoCameraProperties.SensorResolution, Optional*) – Camera resolution to be used
- **fps** (*int, Optional*) – Camera FPS set on the device. Can limit / increase the amount of frames produced by the camera
- **orientation** (*depthai.CameraImageOrientation, Optional*) – Custom camera orientation to be set on the device
- **xout** (*bool, Optional*) – If set to `True`, a dedicated `depthai.node.XLinkOut` will be created for this node
- **args** (*Object, Optional*) – Arguments from the ArgsManager

Return type
MonoCamera

updateIrConfig(*device*, *irLaser=None*, *irFlood=None*)

Updates IR configuration

Parameters

- **irLaser** (*int, Optional*) – Sets the IR laser dot projector brightness (0..1200)
- **irFlood** (*int, Optional*) – Sets the IR flood illuminator light brightness (0..1500)

createDepth(*dct=245*, *median=None*, *sigma=0*, *lr=True*, *lrcThreshold=5*, *extended=False*, *subpixel=False*, *useDisparity=False*, *useDepth=False*, *useRectifiedLeft=False*, *useRectifiedRight=False*, *runtimeSwitch=False*, *alignment=None*, *control=True*, *pipeline=None*, *args=None*)

Creates `depthai.node.StereoDepth` node based on specified attributes

Parameters

- **dct** (*int, Optional*) – Disparity Confidence Threshold (0..255). The less confident the network is, the more empty values are present in the depth map.
- **median** (*depthai.MedianFilter, Optional*) – Median filter to be applied on the depth, use with `depthai.MedianFilter.MEDIAN_OFF` to disable median filtering
- **sigma** (*int, Optional*) – Sigma value for bilateral filter (0..65535). If set to `0`, the filter will be disabled.
- **lr** (*bool, Optional*) – Set to `True` to enable Left-Right Check

- **lrcThreshold** (*int, Optional*) – Sets the Left-Right Check threshold value (0..10)
- **extended** (*bool, Optional*) – Set to `True` to enable the extended disparity
- **subpixel** (*bool, Optional*) – Set to `True` to enable the subpixel disparity
- **useDisparity** (*bool, Optional*) – Set to `True` to create output queue for disparity frames
- **useDepth** (*bool, Optional*) – Set to `True` to create output queue for depth frames
- **useRectifiedLeft** (*bool, Optional*) – Set to `True` to create output queue for rectified left frames
- **useRectifiedRigh** (*bool, Optional*) – Set to `True` to create output queue for rectified right frames
- **runtimeSwitch** (*bool, Optional*) – Allows to change the depth configuration during the runtime but allocates resources for worst-case scenario (disabled by default)
- **alignment** (*depthai.CameraBoardSocket, Optional*) – Aligns the depth map to the specified camera socket
- **args** (*Object, Optional*) – Arguments from the ArgsManager

Raises

`RuntimeError` – if left of right mono cameras were not initialized

Return type

StereoDepth

captureStill()

triggerAutoFocus()

triggerAutoExposure()

triggerAutoWhiteBalance()

updateColorCamConfig(*exposure=None*, *sensitivity=None*, *saturation=None*, *contrast=None*, *brightness=None*, *sharpness=None*, *autofocus=None*, *autowhitebalance=None*, *focus=None*, *whitebalance=None*)

Updates `depthai.node.ColorCamera` node config

Parameters

- **exposure** (*int, Optional*) – Exposure time in microseconds. Has to be set together with `sensitivity` (Usual range: 1..33000)
- **sensitivity** (*int, Optional*) – Sensivity as ISO value. Has to be set together with `exposure` (Usual range: 100..1600)
- **saturation** (*int, Optional*) – Image saturation (Allowed range: -10..10)
- **contrast** (*int, Optional*) – Image contrast (Allowed range: -10..10)
- **brightness** (*int, Optional*) – Image brightness (Allowed range: -10..10)
- **sharpness** (*int, Optional*) – Image sharpness (Allowed range: 0..4)
- **autofocus** (*dai.CameraControl.AutoFocusMode, Optional*) – Set the autofocus mode
- **autowhitebalance** (*dai.CameraControl.AutoFocusMode, Optional*) – Set the autowhitebalance mode

- **focus** (*int, Optional*) – Set the manual focus (lens position)
- **whitebalance** (*int, Optional*) – Set the manual white balance

updateLeftCamConfig(*exposure=None*, *sensitivity=None*, *saturation=None*, *contrast=None*, *brightness=None*, *sharpness=None*)

Updates left depthai.node.MonoCamera node config

Parameters

- **exposure** (*int, Optional*) – Exposure time in microseconds. Has to be set together with sensitivity (Usual range: 1..33000)
- **sensitivity** (*int, Optional*) – Sensivity as ISO value. Has to be set together with exposure (Usual range: 100..1600)
- **saturation** (*int, Optional*) – Image saturation (Allowed range: -10..10)
- **contrast** (*int, Optional*) – Image contrast (Allowed range: -10..10)
- **brightness** (*int, Optional*) – Image brightness (Allowed range: -10..10)
- **sharpness** (*int, Optional*) – Image sharpness (Allowed range: 0..4)

updateRightCamConfig(*exposure=None*, *sensitivity=None*, *saturation=None*, *contrast=None*, *brightness=None*, *sharpness=None*)

Updates right depthai.node.MonoCamera node config

Parameters

- **exposure** (*int, Optional*) – Exposure time in microseconds. Has to be set together with sensitivity (Usual range: 1..33000)
- **sensitivity** (*int, Optional*) – Sensivity as ISO value. Has to be set together with exposure (Usual range: 100..1600)
- **saturation** (*int, Optional*) – Image saturation (Allowed range: -10..10)
- **contrast** (*int, Optional*) – Image contrast (Allowed range: -10..10)
- **brightness** (*int, Optional*) – Image brightness (Allowed range: -10..10)
- **sharpness** (*int, Optional*) – Image sharpness (Allowed range: 0..4)

updateDepthConfig(*dct=None*, *sigma=None*, *median=None*, *lrcThreshold=None*)

Updates depthai.node.StereoDepth node config

Parameters

- **dct** (*int, Optional*) – Disparity Confidence Threshold (0..255). The less confident the network is, the more empty values are present in the depth map.
- **median** (*depthai.MedianFilter, Optional*) – Median filter to be applied on the depth, use with depthai.MedianFilter.MEDIANOFF to disable median filtering
- **sigma** (*int, Optional*) – Sigma value for bilateral filter (0..65535). If set to 0, the filter will be disabled.
- **lrc** (*bool, Optional*) – Enables or disables Left-Right Check mode
- **lrcThreshold** (*int, Optional*) – Sets the Left-Right Check threshold value (0..10)

addNn(*nn*, *xoutNnInput=False*, *xoutSbb=False*)

Adds NN node to current pipeline. Usually obtained by calling *depthai_sdk.managers.NNetManager.createNN* method first

Parameters

- **nn** (*depthai.node.NeuralNetwork*) – prepared NeuralNetwork node to be attached to the pipeline
- **xoutNnInput** (*bool*) – Set to `True` to create output queue for NN's passthough frames
- **xoutSbb** (*bool*) – Set to `True` to create output queue for Spatial Bounding Boxes (area that is used to calculate spatial location)

createSystemLogger(*rate=1*)

Creates `depthai.node.SystemLogger` node together with XLinkOut

Parameters

rate (*int, Optional*) – Specify logging rate (in Hz)

createEncoder(*cameraName*, *encFps=30*, *encQuality=100*)

Creates H.264 / H.265 video encoder (`depthai.node.VideoEncoder` instance)

Parameters

- **cameraName** (*str*) – Camera name to create the encoder for
- **encFps** (*int, Optional*) – Specify encoding FPS
- **encQuality** (*int, Optional*) – Specify encoding quality (1-100)

Raises

- **ValueError** – if cameraName is not a supported camera name
- **RuntimeError** – if specified camera node was not present

enableLowBandwidth(*poeQuality*)

Enables low-bandwidth mode

Parameters

poeQuality (*int, Optional*) – PoE encoding quality, can decrease frame quality but decrease latency

setXlinkChunkSize(*chunkSize*)

setCameraTuningBlob(*path*)

1.3 Preview manager

`PreviewManager` is a class that is made to help you with displaying previews / streams from OAK cameras.

1.3.1 Getting started

`PreviewManager` works hand in hand with the `PipelineManager`, so before you can use Preview, you will first have to declare and initialize the `PipelineManager`. But of course you will also have to import both names to use them. If you do not wish to use the `PipelineManager` you can also create and initialize the pipeline without the help of the manager. `PreviewManager` is created so that you can use only it seperatly.

```
from depthai_sdk import Previews
from depthai_sdk.managers import PipelineManager, PreviewManager
import depthai as dai

# create pipeline
pm = PipelineManager()

# creating color source
pm.createColorCam(xout=True)

# connecting to the device
with dai.Device(pm.pipeline) as device:
    # define configs for above sources
    pv = PreviewManager(display=[Previews.color.name])

    # create stream queues
    pv.createQueues(device)
    while True:
        # prepare and show streams
        pv.prepareFrames()
        pv.showFrames()
```

As you can see from the above code, we first initialized the pipeline, after that we defined sources from where the pipeline will stream and after that we connected to the device. When the device is connected, we can declare and initialize the `PreviewManager` and after that we can see the frames as outputs.

1.3.2 Example of use

```
from depthai_sdk import Previews
from depthai_sdk.managers import PipelineManager, PreviewManager
import depthai as dai
import cv2


# create pipeline
pm = PipelineManager()

# define sources (color, left, right, depth)

# creating color source
pm.createColorCam(xout=True)
pm.createLeftCam(xout=True)
pm.createRightCam(xout=True)
pm.createDepth(useDepth=True)

# connecting to the device
with dai.Device(pm.pipeline) as device:
    # define configs for above sources
    pv = PreviewManager(display=[Previews.color.name, Previews.left.name, Previews.right.
↪name, Previews.depth.name])

```

(continues on next page)

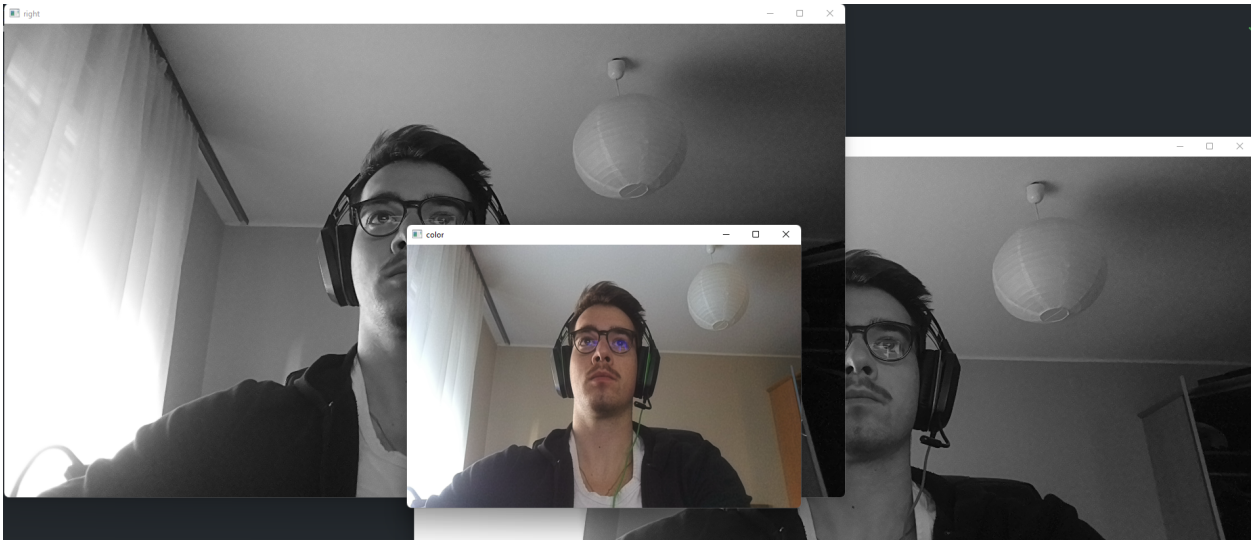
(continued from previous page)

```
    # create stream queues
    pv.createQueues(device)

    while True:
        # prepare and show streams
        pv.prepareFrames()
        pv.showFrames()

        # end program with 'q'
        if cv2.waitKey(1) == ord('q'):
            break
```

In the above example we added a few more sources. Output of the above code should look something like this:



We get frames from all defined sources.

class depthai_sdk.managers.**PreviewManager**

Manager class that handles frames and displays them correctly.

frames = {}

Contains name -> frame mapping that can be used to modify specific frames directly

Type
dict

__init__(*display=[]*, *nnSource=None*, *colorMap=None*, *depthConfig=None*, *dispMultiplier=2.65625*, *mouseTracker=False*, *decode=False*, *fpsHandler=None*, *createWindows=True*)

Parameters

- **display** (*list, Optional*) – List of `depthai_sdk.Previews` objects representing the streams to display
- **mouseTracker** (*bool, Optional*) – If set to `True`, will enable mouse tracker on the preview windows that will display selected pixel value
- **fpsHandler** (*depthai_sdk.fps.FPSHandler, Optional*) – if provided, will use fps handler to modify stream FPS and display it

- **nnSource** (*str, Optional*) – Specifies NN source camera
- **colorMap** (*cv2 color map, Optional*) – Color map applied on the depth frames
- **decode** (*bool, Optional*) – If set to `True`, will decode the received frames assuming they were encoded with MJPEG encoding
- **dispMultiplier** (*float, Optional*) – Multiplier used for depth <-> disparity calculations (calculated on baseline and focal)
- **depthConfig** (*depthai.StereoDepthConfig, optional*) – Configuration used for depth <-> disparity calculations
- **createWindows** (*bool, Optional*) – If True, will create preview windows using OpenCV (enabled by default)

collectCalibData(*device*)

Collects calibration data and calculates `dispScaleFactor` accordingly

Parameters
device (*depthai.Device*) – Running device instance

createQueues(*device*, *callback=None*)

Create output queues for requested preview streams

Parameters

- **device** (*depthai.Device*) – Running device instance
- **callback** (*func, Optional*) – Function that will be executed with preview name once preview window was created

closeQueues()

Closes output queues for requested preview streams

prepareFrames(*blocking=False*, *callback=None*)

This function consumes output queues' packets and parses them to obtain ready to use frames. To convert the frames from packets, this manager uses methods defined in *depthai_sdk.previews.PreviewDecoder*.

Parameters

- **blocking** (*bool, Optional*) – If set to `True`, will wait for a packet in each queue to be available
- **callback** (*func, Optional*) – Function that will be executed once packet with frame has arrived

showFrames(*callback=None*)

Displays stored frame onto preview windows.

Parameters
callback (*func, Optional*) – Function that will be executed right before `cv2.imshow`

has(*name*)

Determines whether manager has a frame assigned to specified preview

Returns
`True` if contains a frame, `False` otherwise

Return type
bool

get(*name*)

Returns a frame assigned to specified preview

Returns

Resolved frame, will default to `None` if not present

Return type

numpy.ndarray

1.4 Encoding manager

`EncodingManager` is a class that is made to help you with creating videos from OAK cameras.

1.4.1 Getting started

Same as `PreviewManager`, `EncodingManager` works hand in hand with the `PipelineManager`, so before you can use the encoder, you will first have to declare and initialize the `PipelineManager`. Again, same as the `PreviewManager`, it is not needed to use `PipelineManager`, you can create the pipeline without it. The managers are created to help you and make your programing experience with `DepthAi SDK` easier.

```
from pathlib import Path
from depthai_sdk import Previews
from depthai_sdk import EncodingManager
from depthai_sdk.managers import PipelineManager
import depthai as dai


# Before declaring the pipeline and encoder, we must create a dictionary of streams, with fps ans values
encodeConfig = dict()
encodeConfig[Previews.color.name] = 30

# create encoder with above declared dictionary and path to save the file ("" will save it next to the program file)
em = EncodingManager(encodeConfig, Path(__file__))

# create pipeline with above mentioned streams
pm = PipelineManager()
pm.createColorCam(xoutVideo=True)

# create encoders for all streams that were initialized
em.createEncoders(pm)

# start device
with dai.Device(pm.pipeline) as device:
    # create stream queues
    em.createDefaultQueues(device)

    while True:
        # save frames to .h256 files
        em.parseQueues()
```

The most important part, when using the `EncodingManager` is that the encoder must be created during the pipeline creation. So to begin, first we create a dictionary, which will contain all streams from the OAK camera as keys, and number of fps as their values. After that we declare the `EncodingManager` with the dictionary, which we just declared, and the wanted path, where the files will be stored. So to store our videos we have to give our encoder the wanted path for saving. We specify our path with `Path(__file__)`. All the files will be stored in `.h265` format, with the file name beeing the source name (so in the above example we will create color.h256).

As you can also see after we declare the pipeline and initialize it's sources, we must set `xoutVideo` to `True` instead of `xout`. And after connecting to the device we parse through the queues and save frames to files.

class depthai_sdk.managers.**EncodingManager**

Manager class that handles video encoding

__init__(*encodeConfig*, *encodeOutput=None*)

Parameters

- **encodeConfig** (*dict*) – Encoding config consisting of keys as preview names and values being the encoding FPS
- **encodeOutput** (*pathlib.Path, Optional*) – Output directory for the recorded videos

createEncoders(*pm*)

Creates VideoEncoder nodes using Pipeline manager, based on config provided during initialization

Parameters
pm (depthai_sdk.managers.PipelineManager) – Pipeline Manager instance

createDefaultQueues(*device*)

Creates output queues for VideoEncoder nodes created in `create_encoders` function. Also, opems up the H.264 / H.265 stream files (e.g. `color.h265`) where the encoded data will be stored.

Parameters
device (*depthai.Device*) – Running device instance

parseQueues()

Parse the output queues, consuming the available data packets in it and storing them inside opened stream files

close()

Closes opened stream files and tries to perform FFMPEG-based conversion from raw stream into mp4 video.

If successful, each stream file (e.g. `color.h265`) will be available along with a ready to use video file (e.g. `color.mp4`).

In case of failure, this method will print traceback and commands that can be used for manual conversion

1.5 Blob manager

`BlobManager` is a class that is made to help you with downloading neural networks as MyriadX blobs.

1.5.1 Getting started

`BlobManager` is very easy and straight forward to use. We declare it and pass which project we want to use as it's argument. Manager supports all models in both Open Model Zoo and our Model zoo. By using `configPath` and `zooDir` when initializing the `BlobManager` you can specify a path to a custom model zoo (path can contain both already compiled blobs and model yml files), which will compile a new blob or read an existing blob. `BlobManager` is able to reuse existing blobs, download models from model zoo and compile custom models based on yml config.

```
from depthai_sdk.managers import BlobManager

# define project that you wish to run
bm = BlobManager(zooName="face-detection-retail-0004")
```

After that, the `blob` is stored in our variable and we can then pass it to our `NNetManager`, as we will see in the next tutorial, or use it in any other project that we wish.

class depthai_sdk.managers.**BlobManager**

Manager class that handles MyriadX blobs.

__init__(*blobPath=None*, *configPath=None*, *zooName=None*, *zooDir=None*, *progressFunc=None*)

Parameters

- **blobPath** (*pathlib.Path, Optional*) – Path to the compiled MyriadX blob file
- **configPath** (*pathlib.Path, Optional*) – Path to model config file that is used to download the model
- **zooName** (*str, Optional*) – Model name to be taken from model zoo
- **zooDir** (*pathlib.Path, Optional*) – Path to model zoo directory
- **progressFunc** (*func, Optional*) – Custom method to show download progress, should accept two arguments - current bytes and max bytes.

getBlob(*shaves=6*, *openvinoVersion=None*, *zooType=None*)

This function is responsible for returning a ready to use MyriadX blob once requested. It will compile the model automatically using our online blobconverter tool. The compilation process will be ran only once, each subsequent call will return a path to previously compiled blob

Parameters

- **shaves** (*int, Optional*) – Specify how many shaves the model will use. Range 1-16
- **openvinoVersion** (*depthai.OpenVINO.Version, Optional*) – OpenVINO version which will be used to compile the MyriadX blob
- **zooType** (*str, Optional*) – Specifies model zoo type to download blob from

Returns

Path to compiled MyriadX blob

Return type

pathlib.Path

Raises

- **SystemExit** – If model name is not found in the zoo, this method will print all available ones and terminate
- **RuntimeError** – If conversion failed with unknown status
- **Exception** – If some unknown error will occur (reraise)

1.6 Nnet manager

`NNetManager` is a class that is made to help you with setting up neural networks (NN). It is also responsible for all NN related functionalities. It's capable of creating appropriate nodes and connections, decoding neural network output automatically or by using external handler file.

1.6.1 Getting started

To get started we first have to know some things that the manager offers. Firstly the manager is responsible for running our NN, which means that our manager will need a blob to work with. We can pass the blob to our `NNetManager` either with our `BlobManager` or we can pass the blob directly from the `blobconverter` module. We now have our blob, now we need to declare our pipeline through which our NN will be receiving data. This step is best done with the help of `PipelineManager`, as the manager already contains methods for NN nodes (`addNN` and `setNnManager` methods). After initializing all of that we are almost done. For convenience we can use `PreviewManager` to parse our frames, but this step is not needed as te `NNetManager` is able to parse raw frames. Bellow you can see 2 projects that use the `NNetManager`. Every major step is also commented for better understanding.

1.6.2 Face detection

```
from depthai_sdk import Previews
from depthai_sdk.managers import PipelineManager, PreviewManager, NNetManager, BlobManager
import depthai as dai
import cv2

# create pipeline
pm = PipelineManager()

# define camera source (in this case color and change winow size to 600 x 500)
pm.createColorCam(xout=True)

# define project that you wish to run
bm = BlobManager(zooName="face-detection-retail-0004")

# define Neural network configs
nm = NNetManager(inputSize=(300, 300), nnFamily="mobilenet")
nn = nm.createNN(pipeline=pm.pipeline, nodes=pm.nodes, source=Previews.color.name,
                 blobPath=bm.getBlob(shaves=6, openvinoVersion=pm.pipeline.getOpenVINOVersion()))
pm.addNn(nn)

# connect to device
with dai.Device(pm.pipeline) as device:
    # define configs for above sources
    pv = PreviewManager(display=[Previews.color.name])

    # create stream and neural network queues
    pv.createQueues(device)
    nm.createQueues(device)
    nnData = []
```

(continues on next page)

(continued from previous page)

```

    while True:
        # read frames
        pv.prepareFrames()
        inNn = nm.outputQueue.tryGet()

        if inNn is not None:
            nnData = nm.decode(inNn)

        # draw information on frame and show frame
        nm.draw(pv, nnData)
        pv.showFrames()

        # end program with 'q'
        if cv2.waitKey(1) == ord('q'):
            break
```

In this above example we will use all classes that we learned before and run the face detection project. First we define the pipeline and initialize the streams. After that we load in our blob (`face-detection-retail-0004`) and send it in to our `NNetManager`. Every project has its own `inputSize` (desired NN input size, which should match input size defined in the network itself (width, height)) the and `familyName` (supported NN types / family names are `YOLO` and `nobilenet`). After all that is initialized, we add our neural network to our pipeline and connect to our device. In our device we set our `Previews`, to see our stream and create our stream queues. Like every other class that we covered, we need a loop, that will keep our project running, and in our loop, we get our frames, use our neural network to draw over our frames, and then we show them on the stream.

Outputs of our above program should look like this:

managers/_static/images/face_detection.png

managers/_static/images/no_face_detection.png

If our face is shown, our neural network detects it, but if we cover it, our neural network will not detect it.

1.6.3 Mobile net

```
from depthai_sdk import Previews
from depthai_sdk.managers import PipelineManager, PreviewManager, NNetManager, BlobManager
import depthai as dai
import cv2

# create pipeline
pm = PipelineManager()

# define camera source (in this case color and change winow size to 600 x 500)
pm.createColorCam(xout=True)

# define project that you wish to run
bm = BlobManager(zooName="mobilenet-ssd")

# define Neural network configs
nm = NNetManager(inputSize=(300, 300), nnFamily="mobilenet")
nn = nm.createNN(pipeline=pm.pipeline, nodes=pm.nodes, source=Previews.color.name,
                 blobPath=bm.getBlob(shaves=6, openvinoVersion=pm.pipeline.getOpenVINOVersion()))
pm.addNn(nn)

# connect to device
with dai.Device(pm.pipeline) as device:
    # define configs for above sources
    pv = PreviewManager(display=[Previews.color.name])

    # create stream and neural network queues
    pv.createQueues(device)
    nm.createQueues(device)
    nnData = []

```

(continues on next page)

(continued from previous page)

```
    while True:
        # read frames
        pv.prepareFrames()
        inNn = nm.outputQueue.tryGet()

        if inNn is not None:
            nnData = nm.decode(inNn)

        # draw information on frame and show frame
        nm.draw(pv, nnData)
        pv.showFrames()

        # end program with 'q'
        if cv2.waitKey(1) == ord('q'):
            break
```

This example shows how to use the `MobileNetSSD` project. The code should be almost the same as the one that we used in the above example, with the only difference being the blob. In this example we load the `mobilenet-ssd` blob and pass it to our neural network.

class depthai_sdk.managers.**NNetManager**

Manager class handling all NN-related functionalities. It's capable of creating appropriate nodes and connections, decoding neural network output automatically or by using external handler file.

__init__(*inputSize*, *nnFamily=None*, *labels=[]*, *confidence=0.5*, *sync=False*)

Parameters

- **inputSize** (*tuple*) – Desired NN input size, should match the input size defined in the network itself (width, height)
- **nnFamily** (*str, Optional*) – type of NeuralNetwork to be processed. Supported: `"YOLO"` and `mobilenet`
- **labels** (*list, Optional*) – Allows to display class label instead of ID when drawing nn detections.
- **confidence** (*float, Optional*) – Specify detection nn's confidence threshold
- **sync** (*bool, Optional*) – Store NN results for preview syncing (to be used with SyncedPreviewManager

sourceChoices = ('color', 'left', 'right', 'rectifiedLeft', 'rectifiedRight', 'host')

List of available neural network inputs

Type
list

source = None

Selected neural network input

Type
str

inputSize = None

NN input size (width, height)

Type
tuple

`openvinoVersion = None`

OpenVINO version, available only if parsed from config file (see *readConfig()*)

Type
depthai.OpenVINO.Version

`inputQueue = None`

DepthAI input queue object that allows to send images from host to device (used only with `host` source)

Type
depthai.DataInputQueue

`outputQueue = None`

DepthAI output queue object that allows to receive NN results from the device.

Type
depthai.DataOutputQueue

`buffer = {}`

nn data buffer, disabled by default. Stores parsed nn data with packet sequence number as dict key

Type
dict

`readConfig`(*path*)

Parses the model config file and adjusts NNetManager values accordingly. It's advised to create a config file for every new network, as it allows to use dedicated NN nodes (for MobilenetSSD and YOLO) or use custom handler to process and display custom network results

Parameters
path (*pathlib.Path*) – Path to model config file (.json)

Raises

- **ValueError** – If path to config file does not exist
- **RuntimeError** – If custom handler does not contain `draw` or `show` methods

`createNN`(*pipeline*, *nodes*, *blobPath*, *source='color'*, *useDepth=False*, *minDepth=100*, *maxDepth=10000*, *sbbScaleFactor=0.3*, *fullFov=True*, *useImageManip=True*)

Creates nodes and connections in provided pipeline that will allow to run NN model and consume it's results.

Parameters

- **pipeline** (*depthai.Pipeline*) – Pipeline instance
- **nodes** (*types.SimpleNamespace*) – Object cointaining all of the nodes added to the pipeline. Available in *depthai_sdk.managers.PipelineManager.nodes*
- **blobPath** (*pathlib.Path*) – Path to MyriadX blob. Might be useful to use together with *depthai_sdk.managers.BlobManager.getBlob()* for dynamic blob compilation
- **source** (*str, Optional*) – Neural network input source, one of *sourceChoices*
- **useDepth** (*bool, Optional*) – If set to True, produced detections will have spatial coordinates included
- **minDepth** (*int, Optional*) – Minimum depth distance in centimeters

- **maxDepth** (*int, Optional*) – Maximum depth distance in centimeters
- **sbbScaleFactor** (*float, Optional*) – Scale of the bounding box that will be used to calculate spatial coordinates for detection. If set to 0.3, it will scale down center-wise the bounding box to 0.3 of it's original size and use it to calculate spatial location of the object
- **fullFov** (*bool, Optional*) – If set to False, manager will include crop offset when scaling the detections. Usually should be set to True (if you don't perform aspect ratio crop or when *keepAspectRatio* flag on camera/manip node is set to False
- **useImageManip** (*bool, Optional*) – If set to False, manager will not create an image manip node for input image scaling - which may result in an input image being not adjusted for the NeuralNetwork node. Can be useful when we want to limit the amount of nodes running simultaneously on device

Returns
Configured NN node that was added to the pipeline

Return type
depthai.node.NeuralNetwork

Raises
RuntimeError – If source is not a valid choice or when input size has not been set.

getLabelText(*label*)

Retrieves text assigned to specific label

Parameters
label (*int*) – Integer representing detection label, usually returned from NN node

Returns
Label text assigned to specific label id or label id

Return type
str

Raises
RuntimeError – If source is not a valid choice or when input size has not been set.

parse(*blocking=False*)

decode(*inNn*)

Decodes NN output. Performs generic handling for supported detection networks or calls custom handler methods

Parameters
inNn (*depthai.NNData*) – Integer representing detection label, usually returned from NN node

Returns
Decoded NN data

Raises
RuntimeError – if outputFormat specified in model config file is not recognized

draw(*source*, *decodedData*)

Draws NN results onto the frames. It's responsible to correctly map the results onto each frame requested, including applying crop offset or preparing a correct normalization frame, then draws them with all information provided (confidence, label, spatial location, label count).

Also, it's able to call custom nn handler method `draw` to hand over drawing the results

Parameters

- **source** (depthai_sdk.managers.PreviewManager | *numpy.ndarray*) – Draw target. If supplied with a regular frame, it will draw the count on that frame

  If supplied with *depthai_sdk.managers.PreviewManager* instance, it will print the count label on all of the frames that it stores

- **decodedData** – Detections from neural network node, usually returned from *decode()* method

createQueues(*device*)

Creates output queue for NeuralNetwork node and, if using `host` as a *source*, it will also create input queue.

Parameters
device (*depthai.Device*) – Running device instance

closeQueues()

Closes output queues created by *createQueues()*

sendInputFrame(*frame*, *seqNum=None*)

Sends a frame into *inputQueue* object. Handles scaling down the frame, creating a proper depthai.ImgFrame and sending it to the queue. Be sure to use `host` as a *source* and call *createQueues()* prior input queue.

Parameters

- **frame** (*numpy.ndarray*) – Frame to be sent to the device
- **seqNum** (*int, Optional*) – Sequence number set on ImgFrame. Useful in synchronization scenarios

Returns
scaled frame that was sent to the NN (same width/height as NN input)

Return type
numpy.ndarray

Raises
RuntimeError – if *inputQueue* is `None` (unable to send the image)

countLabel(*label*)

Enables object count for specific label. Label count will be printed once *draw()* method is called

Parameters
label (*str | int*) – Label to be counted. If model is using mappings in model config file, supply here a `str` label to be tracked. If no mapping is present, specify the label as `int` (NN-default)

1.7 DepthAI SDK API

depthai_sdk.managers.PipelineManager	Helps in setting up processing pipeline
depthai_sdk.managers.NNetManager	Helps in setting up neural networks
depthai_sdk.managers.PreviewManager	Helps in displaying preview from OAK cameras
depthai_sdk.managers.EncodingManager	Helps in creating videos from OAK cameras
depthai_sdk.managers.BlobManager	Helps in downloading neural networks as MyriadX blobs
depthai_sdk.fps	For FPS calculations
depthai_sdk.previews	For frame handling
depthai_sdk.utils	For various most-common tasks

1.7.1 Managers

class depthai_sdk.managers.**BlobManager**

Manager class that handles MyriadX blobs.

__init__(*blobPath=None*, *configPath=None*, *zooName=None*, *zooDir=None*, *progressFunc=None*)

Parameters

- **blobPath** (*pathlib.Path, Optional*) – Path to the compiled MyriadX blob file
- **configPath** (*pathlib.Path, Optional*) – Path to model config file that is used to download the model
- **zooName** (*str, Optional*) – Model name to be taken from model zoo
- **zooDir** (*pathlib.Path, Optional*) – Path to model zoo directory
- **progressFunc** (*func, Optional*) – Custom method to show download progress, should accept two arguments - current bytes and max bytes.

getBlob(*shaves=6*, *openvinoVersion=None*, *zooType=None*)

This function is responsible for returning a ready to use MyriadX blob once requested. It will compile the model automatically using our online blobconverter tool. The compilation process will be ran only once, each subsequent call will return a path to previously compiled blob

Parameters

- **shaves** (*int, Optional*) – Specify how many shaves the model will use. Range 1-16
- **openvinoVersion** (*depthai.OpenVINO.Version, Optional*) – OpenVINO version which will be used to compile the MyriadX blob
- **zooType** (*str, Optional*) – Specifies model zoo type to download blob from

Returns

Path to compiled MyriadX blob

Return type

pathlib.Path

Raises

- **SystemExit** – If model name is not found in the zoo, this method will print all available ones and terminate
- **RuntimeError** – If conversion failed with unknown status

- **Exception** – If some unknown error will occur (reraise)

class depthai_sdk.managers.**EncodingManager**

Manager class that handles video encoding

__init__(*encodeConfig*, *encodeOutput=None*)

Parameters

- **encodeConfig** (*dict*) – Encoding config consisting of keys as preview names and values being the encoding FPS
- **encodeOutput** (*pathlib.Path, Optional*) – Output directory for the recorded videos

createEncoders(*pm*)

Creates VideoEncoder nodes using Pipeline manager, based on config provided during initialization

Parameters
pm (depthai_sdk.managers.PipelineManager) – Pipeline Manager instance

createDefaultQueues(*device*)

Creates output queues for VideoEncoder nodes created in `create_encoders` function. Also, opems up the H.264 / H.265 stream files (e.g. `color.h265`) where the encoded data will be stored.

Parameters
device (*depthai.Device*) – Running device instance

parseQueues()

Parse the output queues, consuming the available data packets in it and storing them inside opened stream files

close()

Closes opened stream files and tries to perform FFMPEG-based conversion from raw stream into mp4 video.

If successful, each stream file (e.g. `color.h265`) will be available along with a ready to use video file (e.g. `color.mp4`).

In case of failure, this method will print traceback and commands that can be used for manual conversion

class depthai_sdk.managers.**NNetManager**

Manager class handling all NN-related functionalities. It's capable of creating appropriate nodes and connections, decoding neural network output automatically or by using external handler file.

__init__(*inputSize*, *nnFamily=None*, *labels=[]*, *confidence=0.5*, *sync=False*)

Parameters

- **inputSize** (*tuple*) – Desired NN input size, should match the input size defined in the network itself (width, height)
- **nnFamily** (*str, Optional*) – type of NeuralNetwork to be processed. Supported: `"YOLO"` and `mobilenet`
- **labels** (*list, Optional*) – Allows to display class label instead of ID when drawing nn detections.
- **confidence** (*float, Optional*) – Specify detection nn's confidence threshold
- **sync** (*bool, Optional*) – Store NN results for preview syncing (to be used with SyncedPreviewManager

sourceChoices = ('color', 'left', 'right', 'rectifiedLeft', 'rectifiedRight', 'host')

List of available neural network inputs

Type
list

source = None

Selected neural network input

Type
str

inputSize = None

NN input size (width, height)

Type
tuple

openvinoVersion = None

OpenVINO version, available only if parsed from config file (see *readConfig()*)

Type
depthai.OpenVINO.Version

inputQueue = None

DepthAI input queue object that allows to send images from host to device (used only with `host` source)

Type
depthai.DataInputQueue

outputQueue = None

DepthAI output queue object that allows to receive NN results from the device.

Type
depthai.DataOutputQueue

buffer = {}

nn data buffer, disabled by default. Stores parsed nn data with packet sequence number as dict key

Type
dict

readConfig(*path*)

Parses the model config file and adjusts NNetManager values accordingly. It's advised to create a config file for every new network, as it allows to use dedicated NN nodes (for MobilenetSSD and YOLO) or use custom handler to process and display custom network results

Parameters
path (*pathlib.Path*) – Path to model config file (.json)

Raises

- **ValueError** – If path to config file does not exist
- **RuntimeError** – If custom handler does not contain `draw` or `show` methods

createNN(*pipeline*, *nodes*, *blobPath*, *source='color'*, *useDepth=False*, *minDepth=100*, *maxDepth=10000*, *sbbScaleFactor=0.3*, *fullFov=True*, *useImageManip=True*)

Creates nodes and connections in provided pipeline that will allow to run NN model and consume it's results.

Parameters

- **pipeline** (*depthai.Pipeline*) – Pipeline instance
- **nodes** (*types.SimpleNamespace*) – Object cointaining all of the nodes added to the pipeline. Available in *depthai_sdk.managers.PipelineManager.nodes*
- **blobPath** (*pathlib.Path*) – Path to MyriadX blob. Might be useful to use together with *depthai_sdk.managers.BlobManager.getBlob()* for dynamic blob compilation
- **source** (*str, Optional*) – Neural network input source, one of *sourceChoices*
- **useDepth** (*bool, Optional*) – If set to True, produced detections will have spatial coordinates included
- **minDepth** (*int, Optional*) – Minimum depth distance in centimeters
- **maxDepth** (*int, Optional*) – Maximum depth distance in centimeters
- **sbbScaleFactor** (*float, Optional*) – Scale of the bounding box that will be used to calculate spatial coordinates for detection. If set to 0.3, it will scale down center-wise the bounding box to 0.3 of it's original size and use it to calculate spatial location of the object
- **fullFov** (*bool, Optional*) – If set to False, manager will include crop offset when scaling the detections. Usually should be set to True (if you don't perform aspect ratio crop or when *keepAspectRatio* flag on camera/manip node is set to False
- **useImageManip** (*bool, Optional*) – If set to False, manager will not create an image manip node for input image scaling - which may result in an input image being not adjusted for the NeuralNetwork node. Can be useful when we want to limit the amount of nodes running simultaneously on device

Returns
Configured NN node that was added to the pipeline

Return type
depthai.node.NeuralNetwork

Raises
RuntimeError – If source is not a valid choice or when input size has not been set.

getLabelText(*label*)

Retrieves text assigned to specific label

Parameters
label (*int*) – Integer representing detection label, usually returned from NN node

Returns
Label text assigned to specific label id or label id

Return type
str

Raises
RuntimeError – If source is not a valid choice or when input size has not been set.

parse(*blocking=False*)

decode(*inNn*)

Decodes NN output. Performs generic handling for supported detection networks or calls custom handler methods

Parameters
inNn (*depthai.NNData*) – Integer representing detection label, usually returned from NN node

Returns
Decoded NN data

Raises
RuntimeError – if outputFormat specified in model config file is not recognized

draw(*source*, *decodedData*)

Draws NN results onto the frames. It's responsible to correctly map the results onto each frame requested, including applying crop offset or preparing a correct normalization frame, then draws them with all information provided (confidence, label, spatial location, label count).

Also, it's able to call custom nn handler method `draw` to hand over drawing the results

Parameters

- **source** (depthai_sdk.managers.PreviewManager | *numpy.ndarray*) – Draw target. If supplied with a regular frame, it will draw the count on that frame

  If supplied with *depthai_sdk.managers.PreviewManager* instance, it will print the count label on all of the frames that it stores

- **decodedData** – Detections from neural network node, usually returned from *decode()* method

createQueues(*device*)

Creates output queue for NeuralNetwork node and, if using `host` as a *source*, it will also create input queue.

Parameters
device (*depthai.Device*) – Running device instance

closeQueues()

Closes output queues created by *createQueues()*

sendInputFrame(*frame*, *seqNum=None*)

Sends a frame into *inputQueue* object. Handles scaling down the frame, creating a proper depthai.ImgFrame and sending it to the queue. Be sure to use `host` as a *source* and call *createQueues()* prior input queue.

Parameters

- **frame** (*numpy.ndarray*) – Frame to be sent to the device
- **seqNum** (*int, Optional*) – Sequence number set on ImgFrame. Useful in synchronization scenarios

Returns
scaled frame that was sent to the NN (same width/height as NN input)

Return type
numpy.ndarray

Raises
RuntimeError – if *inputQueue* is `None` (unable to send the image)

countLabel(*label*)

Enables object count for specific label. Label count will be printed once *draw()* method is called

Parameters
label (*str | int*) – Label to be counted. If model is using mappings in model config file, supply here a `str` label to be tracked. If no mapping is present, specify the label as `int` (NN-default)

class depthai_sdk.managers.**PipelineManager**

Manager class handling different `depthai.Pipeline` operations. Most of the functions wrap up nodes creation and connection logic onto a set of convenience functions.

__init__(*openvinoVersion=None*, *poeQuality=100*, *lowCapabilities=False*, *lowBandwidth=False*)

pipeline

Ready to use requested pipeline. Can be passed to `depthai.Device` to start execution

Type
depthai.Pipeline

nodes

Contains all nodes added to the *pipeline* object, can be used to conveniently access nodes by their name

Type
types.SimpleNamespace

openvinoVersion = None

OpenVINO version which will be used in pipeline

Type
depthai.OpenVINO.Version

poeQuality = None

PoE encoding quality, can decrease frame quality but decrease latency

Type
int, Optional

lowBandwidth = False

If set to `True`, manager will MJPEG-encode the packets sent from device to host to lower the bandwidth usage. **Can break** if more than 3 encoded outputs requested

Type
bool

lowCapabilities = False

If set to `True`, manager will try to optimize the pipeline to reduce the amount of host-side calculations (useful for RPi or other embedded systems)

Type
bool

setNnManager(*nnManager*)

Assigns NN manager. It also syncs the pipeline versions between those two objects

Parameters
nnManager (*depthai_sdk.managers.NNetManager*) – NN manager instance

createDefaultQueues(*device*)

Creates default queues for config updates

Parameters
device (*depthai.Device*) – Running device instance

`closeDefaultQueues()`

Creates default queues for config updates

Parameters
device (*depthai.Device*) – Running device instance

`createColorCam(previewSize=None, res=<SensorResolution.THE_1080_P: 0>, fps=30, fullFov=True, orientation=None, colorOrder=<ColorOrder.BGR: 0>, xout=False, xoutVideo=False, xoutStill=False, control=True, pipeline=None, args=None)`

Creates depthai.node.ColorCamera node based on specified attributes

Parameters

- **previewSize** (*tuple, Optional*) – Size of the preview - `(width, height)`
- **res** (*depthai.ColorCameraProperties.SensorResolution, Optional*) – Camera resolution to be used
- **fps** (*int, Optional*) – Camera FPS set on the device. Can limit / increase the amount of frames produced by the camera
- **fullFov** (*bool, Optional*) – If set to `True`, full frame will be scaled down to nn size. If to `False`, it will first center crop the frame to meet the NN aspect ratio and then scale down the image.
- **orientation** (*depthai.CameraImageOrientation, Optional*) – Custom camera orientation to be set on the device
- **colorOrder** (*depthai.ColorCameraProperties, Optional*) – Color order to be used
- **xout** (*bool, Optional*) – If set to `True`, a dedicated depthai.node.XLinkOut will be created for this node
- **xoutVideo** (*bool, Optional*) – If set to `True`, a dedicated depthai.node.XLinkOut will be created for *video* output of this node
- **xoutStill** (*bool, Optional*) – If set to `True`, a dedicated depthai.node.XLinkOut will be created for *still* output of this node
- **args** (*Object, Optional*) – Arguments from the ArgsManager

Return type
ColorCamera

`createLeftCam(res=None, fps=30, orientation=None, xout=False, control=True, pipeline=None, args=None)`

Creates depthai.node.MonoCamera node based on specified attributes, assigned to depthai.CameraBoardSocket.LEFT

Parameters

- **res** (*depthai.MonoCameraProperties.SensorResolution, Optional*) – Camera resolution to be used
- **fps** (*int, Optional*) – Camera FPS set on the device. Can limit / increase the amount of frames produced by the camera
- **orientation** (*depthai.CameraImageOrientation, Optional*) – Custom camera orientation to be set on the device
- **xout** (*bool, Optional*) – If set to `True`, a dedicated depthai.node.XLinkOut will be created for this node

- **args** (*Object, Optional*) – Arguments from the ArgsManager

Return type
MonoCamera

createRightCam(*res=None*, *fps=30*, *orientation=None*, *xout=False*, *control=True*, *pipeline=None*, *args=None*)

Creates depthai.node.MonoCamera node based on specified attributes, assigned to depthai.CameraBoardSocket.RIGHT

Parameters

- **res** (*depthai.MonoCameraProperties.SensorResolution, Optional*) – Camera resolution to be used
- **fps** (*int, Optional*) – Camera FPS set on the device. Can limit / increase the amount of frames produced by the camera
- **orientation** (*depthai.CameraImageOrientation, Optional*) – Custom camera orientation to be set on the device
- **xout** (*bool, Optional*) – If set to True, a dedicated depthai.node.XLinkOut will be created for this node
- **args** (*Object, Optional*) – Arguments from the ArgsManager

Return type
MonoCamera

updateIrConfig(*device*, *irLaser=None*, *irFlood=None*)

Updates IR configuration

Parameters

- **irLaser** (*int, Optional*) – Sets the IR laser dot projector brightness (0..1200)
- **irFlood** (*int, Optional*) – Sets the IR flood illuminator light brightness (0..1500)

createDepth(*dct=245*, *median=None*, *sigma=0*, *lr=True*, *lrcThreshold=5*, *extended=False*, *subpixel=False*, *useDisparity=False*, *useDepth=False*, *useRectifiedLeft=False*, *useRectifiedRight=False*, *runtimeSwitch=False*, *alignment=None*, *control=True*, *pipeline=None*, *args=None*)

Creates depthai.node.StereoDepth node based on specified attributes

Parameters

- **dct** (*int, Optional*) – Disparity Confidence Threshold (0..255). The less confident the network is, the more empty values are present in the depth map.
- **median** (*depthai.MedianFilter, Optional*) – Median filter to be applied on the depth, use with depthai.MedianFilter.MEDIAN_OFF to disable median filtering
- **sigma** (*int, Optional*) – Sigma value for bilateral filter (0..65535). If set to 0, the filter will be disabled.
- **lr** (*bool, Optional*) – Set to True to enable Left-Right Check
- **lrcThreshold** (*int, Optional*) – Sets the Left-Right Check threshold value (0..10)
- **extended** (*bool, Optional*) – Set to True to enable the extended disparity
- **subpixel** (*bool, Optional*) – Set to True to enable the subpixel disparity
- **useDisparity** (*bool, Optional*) – Set to True to create output queue for disparity frames

- **useDepth** (*bool, Optional*) – Set to `True` to create output queue for depth frames
- **useRectifiedLeft** (*bool, Optional*) – Set to `True` to create output queue for rectified left frames
- **useRectifiedRigh** (*bool, Optional*) – Set to `True` to create output queue for rectified right frames
- **runtimeSwitch** (*bool, Optional*) – Allows to change the depth configuration during the runtime but allocates resources for worst-case scenario (disabled by default)
- **alignment** (*depthai.CameraBoardSocket, Optional*) – Aligns the depth map to the specified camera socket
- **args** (*Object, Optional*) – Arguments from the ArgsManager

Raises
`RuntimeError` – if left of right mono cameras were not initialized

Return type
StereoDepth

captureStill()

triggerAutoFocus()

triggerAutoExposure()

triggerAutoWhiteBalance()

updateColorCamConfig(*exposure=None*, *sensitivity=None*, *saturation=None*, *contrast=None*, *brightness=None*, *sharpness=None*, *autofocus=None*, *autowhitebalance=None*, *focus=None*, *whitebalance=None*)

Updates `depthai.node.ColorCamera` node config

Parameters

- **exposure** (*int, Optional*) – Exposure time in microseconds. Has to be set together with `sensitivity` (Usual range: 1..33000)
- **sensitivity** (*int, Optional*) – Sensivity as ISO value. Has to be set together with `exposure` (Usual range: 100..1600)
- **saturation** (*int, Optional*) – Image saturation (Allowed range: -10..10)
- **contrast** (*int, Optional*) – Image contrast (Allowed range: -10..10)
- **brightness** (*int, Optional*) – Image brightness (Allowed range: -10..10)
- **sharpness** (*int, Optional*) – Image sharpness (Allowed range: 0..4)
- **autofocus** (*dai.CameraControl.AutoFocusMode, Optional*) – Set the autofocus mode
- **autowhitebalance** (*dai.CameraControl.AutoFocusMode, Optional*) – Set the autowhitebalance mode
- **focus** (*int, Optional*) – Set the manual focus (lens position)
- **whitebalance** (*int, Optional*) – Set the manual white balance

updateLeftCamConfig(*exposure=None*, *sensitivity=None*, *saturation=None*, *contrast=None*, *brightness=None*, *sharpness=None*)

Updates left `depthai.node.MonoCamera` node config

Parameters

- **exposure** (*int, Optional*) – Exposure time in microseconds. Has to be set together with `sensitivity` (Usual range: 1..33000)
- **sensitivity** (*int, Optional*) – Sensivity as ISO value. Has to be set together with `exposure` (Usual range: 100..1600)
- **saturation** (*int, Optional*) – Image saturation (Allowed range: -10..10)
- **contrast** (*int, Optional*) – Image contrast (Allowed range: -10..10)
- **brightness** (*int, Optional*) – Image brightness (Allowed range: -10..10)
- **sharpness** (*int, Optional*) – Image sharpness (Allowed range: 0..4)

updateRightCamConfig(*exposure=None*, *sensitivity=None*, *saturation=None*, *contrast=None*, *brightness=None*, *sharpness=None*)

Updates right `depthai.node.MonoCamera` node config

Parameters

- **exposure** (*int, Optional*) – Exposure time in microseconds. Has to be set together with `sensitivity` (Usual range: 1..33000)
- **sensitivity** (*int, Optional*) – Sensivity as ISO value. Has to be set together with `exposure` (Usual range: 100..1600)
- **saturation** (*int, Optional*) – Image saturation (Allowed range: -10..10)
- **contrast** (*int, Optional*) – Image contrast (Allowed range: -10..10)
- **brightness** (*int, Optional*) – Image brightness (Allowed range: -10..10)
- **sharpness** (*int, Optional*) – Image sharpness (Allowed range: 0..4)

updateDepthConfig(*dct=None*, *sigma=None*, *median=None*, *lrcThreshold=None*)

Updates `depthai.node.StereoDepth` node config

Parameters

- **dct** (*int, Optional*) – Disparity Confidence Threshold (0..255). The less confident the network is, the more empty values are present in the depth map.
- **median** (*depthai.MedianFilter, Optional*) – Median filter to be applied on the depth, use with `depthai.MedianFilter.MEDIANOFF` to disable median filtering
- **sigma** (*int, Optional*) – Sigma value for bilateral filter (0..65535). If set to `0`, the filter will be disabled.
- **lrc** (*bool, Optional*) – Enables or disables Left-Right Check mode
- **lrcThreshold** (*int, Optional*) – Sets the Left-Right Check threshold value (0..10)

addNn(*nn*, *xoutNnInput=False*, *xoutSbb=False*)

Adds NN node to current pipeline. Usually obtained by calling *depthai_sdk.managers.NNetManager.createNN* method first

Parameters

- **nn** (*depthai.node.NeuralNetwork*) – prepared NeuralNetwork node to be attached to the pipeline
- **xoutNnInput** (*bool*) – Set to `True` to create output queue for NN's passthough frames

- **xoutSbb** (*bool*) – Set to `True` to create output queue for Spatial Bounding Boxes (area that is used to calculate spatial location)

createSystemLogger(*rate=1*)

Creates `depthai.node.SystemLogger` node together with XLinkOut

Parameters

rate (*int, Optional*) – Specify logging rate (in Hz)

createEncoder(*cameraName*, *encFps=30*, *encQuality=100*)

Creates H.264 / H.265 video encoder (`depthai.node.VideoEncoder` instance)

Parameters

- **cameraName** (*str*) – Camera name to create the encoder for
- **encFps** (*int, Optional*) – Specify encoding FPS
- **encQuality** (*int, Optional*) – Specify encoding quality (1-100)

Raises

- **ValueError** – if cameraName is not a supported camera name
- **RuntimeError** – if specified camera node was not present

enableLowBandwidth(*poeQuality*)

Enables low-bandwidth mode

Parameters

poeQuality (*int, Optional*) – PoE encoding quality, can decrease frame quality but decrease latency

setXlinkChunkSize(*chunkSize*)

setCameraTuningBlob(*path*)

class depthai_sdk.managers.**PreviewManager**

Manager class that handles frames and displays them correctly.

frames = {}

Contains name -> frame mapping that can be used to modify specific frames directly

Type

dict

__init__(*display=[]*, *nnSource=None*, *colorMap=None*, *depthConfig=None*, *dispMultiplier=2.65625*, *mouseTracker=False*, *decode=False*, *fpsHandler=None*, *createWindows=True*)

Parameters

- **display** (*list, Optional*) – List of `depthai_sdk.Previews` objects representing the streams to display
- **mouseTracker** (*bool, Optional*) – If set to `True`, will enable mouse tracker on the preview windows that will display selected pixel value
- **fpsHandler** (*depthai_sdk.fps.FPSHandler, Optional*) – if provided, will use fps handler to modify stream FPS and display it
- **nnSource** (*str, Optional*) – Specifies NN source camera
- **colorMap** (*cv2 color map, Optional*) – Color map applied on the depth frames

- **decode** (*bool, Optional*) – If set to `True`, will decode the received frames assuming they were encoded with MJPEG encoding
- **dispMultiplier** (*float, Optional*) – Multiplier used for depth <-> disparity calculations (calculated on baseline and focal)
- **depthConfig** (*depthai.StereoDepthConfig, optional*) – Configuration used for depth <-> disparity calculations
- **createWindows** (*bool, Optional*) – If True, will create preview windows using OpenCV (enabled by default)

collectCalibData(*device*)

Collects calibration data and calculates `dispScaleFactor` accordingly

Parameters
device (*depthai.Device*) – Running device instance

createQueues(*device*, *callback=None*)

Create output queues for requested preview streams

Parameters

- **device** (*depthai.Device*) – Running device instance
- **callback** (*func, Optional*) – Function that will be executed with preview name once preview window was created

closeQueues()

Closes output queues for requested preview streams

prepareFrames(*blocking=False*, *callback=None*)

This function consumes output queues' packets and parses them to obtain ready to use frames. To convert the frames from packets, this manager uses methods defined in *depthai_sdk.previews.PreviewDecoder*.

Parameters

- **blocking** (*bool, Optional*) – If set to `True`, will wait for a packet in each queue to be available
- **callback** (*func, Optional*) – Function that will be executed once packet with frame has arrived

showFrames(*callback=None*)

Displays stored frame onto preview windows.

Parameters
callback (*func, Optional*) – Function that will be executed right before `cv2.imshow`

has(*name*)

Determines whether manager has a frame assigned to specified preview

Returns
`True` if contains a frame, `False` otherwise

Return type
bool

get(*name*)

Returns a frame assigned to specified preview

Returns
Resolved frame, will default to `None` if not present

Return type
numpy.ndarray

1.7.2 Previews

class depthai_sdk.previews.**PreviewDecoder**

static jpegDecode(*data*, *type*)

static nnInput(*packet*, *manager=None*)
Produces NN passthough frame from raw data packet

Parameters
- **packet** (*depthai.ImgFrame*) – Packet received from output queue
- **manager** (depthai_sdk.managers.PreviewManager, *optional*) – PreviewManager instance

Returns
Ready to use OpenCV frame

Return type
numpy.ndarray

static color(*packet*, *manager=None*)
Produces color camera frame from raw data packet

Parameters
- **packet** (*depthai.ImgFrame*) – Packet received from output queue
- **manager** (depthai_sdk.managers.PreviewManager, *optional*) – PreviewManager instance

Returns
Ready to use OpenCV frame

Return type
numpy.ndarray

static left(*packet*, *manager=None*)
Produces left camera frame from raw data packet

Parameters
- **packet** (*depthai.ImgFrame*) – Packet received from output queue
- **manager** (depthai_sdk.managers.PreviewManager, *optional*) – PreviewManager instance

Returns
Ready to use OpenCV frame

Return type
numpy.ndarray

static right(*packet*, *manager=None*)

Produces right camera frame from raw data packet

Parameters

- **packet** (*depthai.ImgFrame*) – Packet received from output queue
- **manager** (depthai_sdk.managers.PreviewManager, *optional*) – PreviewManager instance

Returns
Ready to use OpenCV frame

Return type
numpy.ndarray

static rectifiedLeft(*packet*, *manager=None*)

Produces rectified left frame (depthai.node.StereoDepth.rectifiedLeft) from raw data packet

Parameters

- **packet** (*depthai.ImgFrame*) – Packet received from output queue
- **manager** (depthai_sdk.managers.PreviewManager, *optional*) – PreviewManager instance

Returns
Ready to use OpenCV frame

Return type
numpy.ndarray

static rectifiedRight(*packet*, *manager=None*)

Produces rectified right frame (depthai.node.StereoDepth.rectifiedRight) from raw data packet

Parameters

- **packet** (*depthai.ImgFrame*) – Packet received from output queue
- **manager** (depthai_sdk.managers.PreviewManager, *optional*) – PreviewManager instance

Returns
Ready to use OpenCV frame

Return type
numpy.ndarray

static depthRaw(*packet*, *manager=None*)

Produces raw depth frame (depthai.node.StereoDepth.depth) from raw data packet

Parameters

- **packet** (*depthai.ImgFrame*) – Packet received from output queue
- **manager** (depthai_sdk.managers.PreviewManager, *optional*) – PreviewManager instance

Returns
Ready to use OpenCV frame

Return type
numpy.ndarray

static depth(*depthRaw*, *manager=None*)

Produces depth frame from raw depth frame (converts to disparity and applies color map)

Parameters

- **depthRaw** (*numpy.ndarray*) – OpenCV frame containing raw depth frame
- **manager** (depthai_sdk.managers.PreviewManager, *optional*) – PreviewManager instance

Returns
Ready to use OpenCV frame

Return type
numpy.ndarray

static disparity(*packet*, *manager=None*)

Produces disparity frame (depthai.node.StereoDepth.disparity) from raw data packet

Parameters

- **packet** (*depthai.ImgFrame*) – Packet received from output queue
- **manager** (depthai_sdk.managers.PreviewManager, *optional*) – PreviewManager instance

Returns
Ready to use OpenCV frame

Return type
numpy.ndarray

static disparityColor(*disparity*, *manager=None*)

Applies color map to disparity frame

Parameters

- **disparity** (*numpy.ndarray*) – OpenCV frame containing disparity frame
- **manager** (depthai_sdk.managers.PreviewManager, *optional*) – PreviewManager instance

Returns
Ready to use OpenCV frame

Return type
numpy.ndarray

class depthai_sdk.previews.**Previews**

Enum class, assigning preview name with decode function.

Usually used as e.g. `Previews.color.name` when specifying color preview name.

Can be also used as e.g. `Previews.color.value(packet)` to transform queue output packet to color camera frame

nnInput = functools.partial(<function PreviewDecoder.nnInput>)

color = functools.partial(<function PreviewDecoder.color>)

left = functools.partial(<function PreviewDecoder.left>)

right = functools.partial(<function PreviewDecoder.right>)

rectifiedLeft = functools.partial(<function PreviewDecoder.rectifiedLeft>)

rectifiedRight = functools.partial(<function PreviewDecoder.rectifiedRight>)

depthRaw = functools.partial(<function PreviewDecoder.depthRaw>)

depth = functools.partial(<function PreviewDecoder.depth>)

disparity = functools.partial(<function PreviewDecoder.disparity>)

disparityColor = functools.partial(<function PreviewDecoder.disparityColor>)

class depthai_sdk.previews.**MouseClickTracker**

Class that allows to track the click events on preview windows and show pixel value of a frame in coordinates pointed by the user.

Used internally by *depthai_sdk.managers.PreviewManager*

points = {}

Stores selected point position per frame

Type
dict

values = {}

Stores values assigned to specific point per frame

Type
dict

selectPoint(*name*)

Returns callback function for `cv2.setMouseCallback` that will update the selected point on mouse click event from frame.

Usually used as

```
mct = MouseClickTracker()
# create preview window
cv2.setMouseCallback(window_name, mct.select_point(window_name))
```

Parameters
name (*str*) – Name of the frame

Returns
Callback function for `cv2.setMouseCallback`

extractValue(*name*, *frame*)

Extracts value from frame for a specific point

Parameters

- **name** (*str*) – Name of the frame
- **frame** (*ndarray*) –

1.7.3 FPS

class depthai_sdk.fps.**FPSHandler**

Class that handles all FPS-related operations. Mostly used to calculate different streams FPS, but can also be used to feed the video file based on it's FPS property, not app performance (this prevents the video from being sent to quickly if we finish processing a frame earlier than the next video frame should be consumed)

__init__(*cap=None*, *maxTicks=100*)

Parameters

- **cap** (*cv2.VideoCapture, Optional*) – handler to the video file object
- **maxTicks** (*int, Optional*) – maximum ticks amount for FPS calculation

nextIter()

Marks the next iteration of the processing loop. Will use `time.sleep` method if initialized with video file object

tick(*name*)

Marks a point in time for specified name

Parameters
name (*str*) – Specifies timestamp name

tickFps(*name*)

Calculates the FPS based on specified name

Parameters
name (*str*) – Specifies timestamps' name

Returns
Calculated FPS or `0.0` (default in case of failure)

Return type
float

fps()

Calculates FPS value based on *nextIter()* calls, being the FPS of processing loop

Returns
Calculated FPS or `0.0` (default in case of failure)

Return type
float

printStatus()

Prints total FPS for all names stored in *tick()* calls

drawFps(*frame*, *name*)

Draws FPS values on requested frame, calculated based on specified name

Parameters

- **frame** (*numpy.ndarray*) – Frame object to draw values on
- **name** (*str*) – Specifies timestamps' name

1.7.4 Utils

depthai_sdk.utils.**cosDist**(*a*, *b*)

Calculates cosine distance - https://en.wikipedia.org/wiki/Cosine_similarity

depthai_sdk.utils.**frameNorm**(*frame*, *bbox*)

Mapps bounding box coordinates (0..1) to pixel values on frame

Parameters

- **frame** (*numpy.ndarray*) – Frame to which adjust the bounding box
- **bbox** (*list*) – list of bounding box points in a form of `[x1, y1, x2, y2, ...]`

Returns
Bounding box points mapped to pixel values on frame

Return type
list

depthai_sdk.utils.**toPlanar**(*arr*, *shape=None*)

Converts interleaved frame into planar

Parameters

- **arr** (*numpy.ndarray*) – Interleaved frame
- **shape** (*tuple, optional*) – If provided, the interleaved frame will be scaled to specified shape before converting into planar

Returns
Planar frame

Return type
numpy.ndarray

depthai_sdk.utils.**toTensorResult**(*packet*)

Converts NN packet to dict, with each key being output tensor name and each value being correctly reshaped and converted results array

Useful as a first step of processing NN results for custom neural networks

Parameters
packet (*depthai.NNData*) – Packet returned from NN node

Returns
Dict containing prepared output tensors

Return type
dict

depthai_sdk.utils.**merge**(*source*, *destination*)

Utility function to merge two dictionaries

```
a = { 'first' : { 'all_rows' : { 'pass' : 'dog', 'number' : '1' } } }
b = { 'first' : { 'all_rows' : { 'fail' : 'cat', 'number' : '5' } } }
print(merge(b, a))
# { 'first' : { 'all_rows' : { 'pass' : 'dog', 'fail' : 'cat', 'number' : '5' } } }
```

Parameters

- **source** (*dict*) – first dict to merge

- **destination** (*dict*) – second dict to merge

Returns
merged dict

Return type
dict

depthai_sdk.utils.**loadModule**(*path*)

Loads module from specified path. Used internally e.g. to load a custom handler file from path

Parameters
path (*pathlib.Path*) – path to the module to be loaded

Returns
loaded module from provided path

Return type
module

depthai_sdk.utils.**getDeviceInfo**(*deviceId=None*, *debug=False*)

Find a correct depthai.DeviceInfo object, either matching provided `deviceId` or selected by the user (if multiple devices available) Useful for almost every app where there is a possibility of multiple devices being connected simultaneously

Parameters
deviceId (*str, optional*) – Specifies device MX ID, for which the device info will be collected

Returns
Object representing selected device info

Return type
depthai.DeviceInfo

Raises

- **RuntimeError** – if no DepthAI device was found or, if `deviceId` was specified, no device with matching MX ID was found
- **ValueError** – if value supplied by the user when choosing the DepthAI device was incorrect

depthai_sdk.utils.**showProgress**(*curr*, *max*)

Print progressbar to stdout. Each call to this method will write exactly to the same line, so usually it's used as

```
print("Staring processing")
while processing:
    showProgress(currProgress, maxProgress)
print(" done") # prints in the same line as progress bar and adds a new line
print("Processing finished!")
```

Parameters

- **curr** (*int*) – Current position on progress bar
- **max** (*int*) – Maximum position on progress bar

depthai_sdk.utils.**downloadYTVideo**(*video*, *outputDir*)

Downloads a video from YouTube and returns the path to video. Will choose the best resolutuion if possible.

Parameters

- **video** (*str*) – URL to YouTube video
- **outputDir** (*pathlib.Path*) – Path to directory where youtube video should be downloaded.

Returns
Path to downloaded video file

Return type
pathlib.Path

Raises
RuntimeError – thrown when video download was unsuccessful

depthai_sdk.utils.**cropToAspectRatio**(*frame*, *size*)

Crop the frame to desired aspect ratio and then scales it down to desired size :param frame: Source frame that will be cropped :type frame: numpy.ndarray :param size: Desired frame size (width, height) :type size: tuple

Returns
Cropped frame

Return type
numpy.ndarray

depthai_sdk.utils.**resizeLetterbox**(*frame*, *size*)

Transforms the frame to meet the desired size, preserving the aspect ratio and adding black borders (letterboxing) :param frame: Source frame that will be resized :type frame: numpy.ndarray :param size: Desired frame size (width, height) :type size: tuple

Returns
Resized frame

Return type
numpy.ndarray

depthai_sdk.utils.**createBlankFrame**(*width*, *height*, *rgb_color=(0, 0, 0)*)

Create new image(numpy array) filled with certain color in RGB

Parameters

- **width** (*int*) – New frame width
- **height** (*int*) – New frame height
- **rgb_color** (*tuple, Optional*) – Specify frame fill color in RGB format (default (0,0,0) - black)

Returns
New frame filled with specified color

Return type
numpy.ndarray

PYTHON MODULE INDEX

d

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P