DepthAI SDK Docs

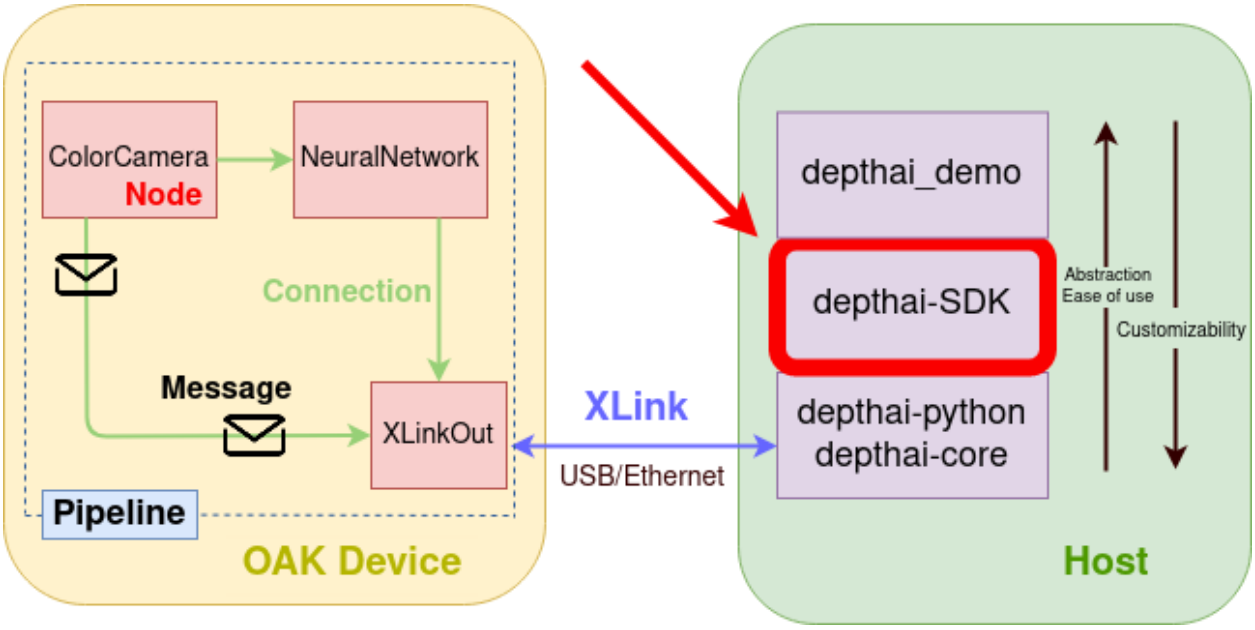
Release 1.13.1

Luxonis

Apr 10, 2024

GETTING STARTED

DepthAI SDK is a Python package built on top of the depthai-python API library that **improves ease of use when developing apps for OAK devices**.



Note: DepthAI SDK is in **alpha stage** until **depthai-sdk 2.0**, so there will likely be API changes during the development.

CHAPTER

ONE

QUICKSTART

The DepthAI SDK is a powerful tool for building computer vision applications using Luxonis devices. This quickstart guide will help you get started with the SDK.

1.1 Installation

DepthAI SDK is available on PyPI. You can install it with the following command:

```
# Linux and macOS
python3 -m pip install depthai-sdk

# Windows
 py -m pip install depthai-sdk
```

1.2 Working with camera

The `OakCamera` class is a fundamental part of the DepthAI SDK, providing a high-level interface for accessing the features of the OAK device. This class simplifies the creation of pipelines that capture video from the OAK camera, run neural networks on the video stream, and visualize the results.

With `OakCamera`, you can easily create color and depth streams using the `create_camera()` and `create_stereo()` methods respectively, and add pre-trained neural networks using the `create_nn()` method. Additionally, you can add custom callbacks to the pipeline using the `callback()` method and record the outputs using the `record()` method.

1.2.1 Blocking behavior

When starting the `OakCamera` object, you can specify whether the `start()` method should block the main thread or not. By default, the `start()` method does not block the main thread, which means you will need to manually poll the camera using the `oak.poll()` method.

```
from depthai_sdk import OakCamera

with OakCamera() as oak:
    color = oak.create_camera('color', resolution='1080p')
    oak.visualize([color])
    oak.start(blocking=False)
```

(continues on next page)

(continued from previous page)

```
    while oak.running():
        oak.poll()
        # this code is executed while the pipeline is running
```

Alternatively, setting the `blocking` argument to `True` will loop and continuously poll the camera, blocking the rest of the code.

```
from depthai_sdk import OakCamera

with OakCamera() as oak:
    color = oak.create_camera('color', resolution='1080p')
    oak.visualize([color])
    oak.start(blocking=True)
    # this code doesn't execute until the pipeline is stopped
```

1.3 Creating color and depth streams

To create a color stream we can use the `OakCamera.create_camera()` method. This method takes the name of the sensor as an argument and returns a *`CameraComponent`* object.

The full list of supported sensors: `color`; `left`; `right`; `cam_{socket},color`, `cam_{socket},mono`, where `{socket}` is a letter from A to H representing the socket on the OAK device. Custom socket names are usually used for FFC devices.

To visualize the stream, we can use the `OakCamera.visualize()` method. This method takes a list of outputs and displays them. Each component has its own outputs, which can be found in the *Components* section.

Here is an example which creates color and depth streams and visualizes the stream:

```
from depthai_sdk import OakCamera

with OakCamera() as oak:
    color = oak.create_camera('color', resolution='1080p')
    stereo = oak.create_stereo(resolution='800p')  # works with stereo devices only!
    oak.visualize([color, stereo])
    oak.start(blocking=True)
```

1.4 Creating YOLO neural network for object detection

DepthAI SDK provides a number of pre-trained neural networks that can be used for object detection, pose estimation, semantic segmentation, and other tasks. To create a neural network, we can use the `OakCamera.create_nn()` method and pass the name of the neural network as an argument.

Similarly to the `OakCamera.create_camera()` method, the `OakCamera.create_nn()` method returns a *`NNComponent`* object.

Here is an example which creates a YOLO neural network for object detection and visualizes the results:

```
from depthai_sdk import OakCamera

with OakCamera() as oak:
    color = oak.create_camera('color', resolution='1080p')
```

(continues on next page)

(continued from previous page)

```
    # List of models that are supported out-of-the-box by the SDK:
    # https://docs.luxonis.com/projects/sdk/en/latest/features/ai_models/#sdk-
↪supported-models
    yolo = oak.create_nn('yolov6n_coco_640x640', input=color)

    oak.visualize([color, yolo])
    oak.start(blocking=True)
```

1.5 Adding custom callbacks

Callbacks are functions that are called when a new frame is available from the camera or neural network. `OakCamera` provides a mechanism for adding custom callbacks to the pipeline using the `OakCamera.callback()` method.

Here is an example which creates a YOLO neural network for object detection and prints the number of detected objects:

```
from depthai_sdk import OakCamera

def print_num_objects(packet):
    print(f'Number of objects detected: {len(packet.detections)}')

with OakCamera() as oak:
    color = oak.create_camera('color', resolution='1080p')
    yolo = oak.create_nn('yolov6n_coco_640x640', input=color)

    oak.callback(yolo, callback=print_num_objects)
    oak.start(blocking=True)
```

1.6 Recording

DepthAI SDK provides a simple API for recording the outputs. The `OakCamera.record()` method takes a list of outputs and a path to the output file. Here is an example which creates a YOLO neural network for object detection and records the results:

```
from depthai_sdk import OakCamera
from depthai_sdk.record import RecordType

with OakCamera() as oak:
    color = oak.create_camera('color', resolution='1080p')
    yolo = oak.create_nn('yolov6n_coco_640x640', input=color)

    oak.record([color, yolo], path='./records', record_type=RecordType.VIDEO)
    oak.start(blocking=True)
```

There are several formats supported by the SDK for recording the outputs:

1. `depthai_sdk.record.RecordType.VIDEO` - record video files.
2. `depthai_sdk.record.RecordType.MCAP` - record MCAP files.
3. `depthai_sdk.record.RecordType.BAG` - record ROS bag files.

You can find more information about recording in the *Recording* section.

1.7 Output syncing

There is a special case when one needs to synchronize multiple outputs. For example, recording color stream and neural network output at the same time. In this case, one can use the `OakCamera.sync()`. This method takes a list of outputs and returns a synchronized output to the specified callback function. Here is an example which synchronizes color stream and YOLO neural network output:

```
from depthai_sdk import OakCamera

def callback(synced_packets):
    print(synced_packets)

with OakCamera() as oak:
    color = oak.create_camera('color', resolution='1080p')
    yolo = oak.create_nn('yolov6n_coco_640x640', input=color)

    oak.sync([color.out.main, yolo.out.main], callback=callback)
    oak.start(blocking=True)
```

1.8 Encoded streams

Luxonis devices support on-device encoding of the outputs to `H.264`, `H.265` and `MJPEG` formats. To enable encoding, we should simply pass the `encode` argument to the `OakCamera.create_camera()` or `OakCamera.create_stereo()` methods. Possible values for the `encode` argument are `h264`, `h265` and `mjpeg`.

Each component has its own encoded output:

- `CameraComponent.Out.encoded`
- `StereoComponent.Out.encoded`
- `NNComponent.Out.encoded`

Here is an example which visualizes the encoded color, YOLO neural network and disparity streams:

```
from depthai_sdk import OakCamera

with OakCamera() as oak:
    color = oak.create_camera('color', resolution='1080p', fps=20, encode='h264')
    stereo = oak.create_stereo('400p', encode='h264')
    yolo = oak.create_nn('yolov6nr3_coco_640x352', input=color)

    oak.visualize([color.out.encoded, stereo.out.encoded, yolo.out.encoded])
    oak.start(blocking=True)
```

CHAPTER

TWO

COMPONENTS

Components are part of the OakCamera class and abstract DepthAI API nodes; their initialization, configuration, and linking. This improves ease of use when developing OAK aplications.

2.1 Available components

- *CameraComponent*
- *NNComponent*
- *StereoComponent*
- *IMUComponent*

2.2 Reference

class depthai_sdk.components.**Component**

SDK component is used as an abstraction to the current DepthAI API node or group of nodes.

on_pipeline_started(*device*)

This function gets called after the pipeline has been started. It is called from the main thread. It can be used to eg. initialize XlinkIn queues.

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

Return type None

2.2.1 CameraComponent

CameraComponent abstracts ColorCamera and MonoCamera nodes and supports mocking the camera when recording is passed during OakCamera initialization. When using *Replaying* feature, this component will mock the camera by sending frames from the host to the OAK device (via XLinkIn node).

Usage

```
from depthai_sdk import OakCamera

with OakCamera() as oak:
    # Create color camera
    color = oak.create_camera('color')

    # Visualize color camera frame stream
    oak.visualize(color.out.main, fps=True)
    # Start the pipeline, continuously poll
    oak.start(blocking=True)
```

Component outputs

- `main` - Uses one of the outputs below.
- `camera` - Default output. Streams either ColorCamera's video (NV12) or MonoCamera's out (GRAY8) frames. Produces *FramePacket*.
- `replay` - If we are using *Replaying* feature. It doesn't actually stream these frames back to the host, but rather sends read frames to syncing mechanism directly (to reduce bandwidth by avoiding loopback). Produces *FramePacket*.
- `encoded` - If we are encoding frames, this will send encoded bitstream to the host. When visualized, it will decode frames (using cv2.imdecode for MJPEG, or pyav for H.26x). Produces *FramePacket*.

Reference

class depthai_sdk.components.**CameraComponent**(*device, pipeline, source, resolution=None, fps=None, encode=None, sensor_type=None, rotation=None, replay=None, name=None, args=None*)

on_pipeline_started(*device*)

This function gets called after the pipeline has been started. It is called from the main thread. It can be used to eg. initialize XlinkIn queues.

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

config_camera(*size=None, resize_mode=<ResizeMode.CROP: 2>, fps=None, resolution=None*)

Configure resolution, scale, FPS, etc.

Parameters

- **size** (*Union[None, Tuple[int, int], str]*) –
- **resize_mode** (*depthai_sdk.classes.enum.ResizeMode*) –
- **fps** (*Optional[float]*) –
- **resolution** (*Optional[Union[str, depthai.ColorCameraProperties.SensorResolution, depthai.MonoCameraProperties.SensorResolution]]*) –

Return type None

control_with_nn(*detection_component, auto_focus=True, auto_exposure=True, debug=False*)

Control the camera AF/AE/AWB based on the object detection results.

Parameters

- **detection_component** (NNComponent) – NNComponent that will be used to control the camera
- **auto_focus** – Enable auto focus to the object
- **auto_exposure** – Enable auto exposure to the object

config_color_camera(*interleaved=None*, *color_order=None*, *manual_focus=None*, *af_mode=None*, *awb_mode=None*, *scene_mode=None*, *anti_banding_mode=None*, *effect_mode=None*, *isp_scale=None*, *sharpness=None*, *luma_denoise=None*, *chroma_denoise=None*)

Parameters

- **interleaved** (*Optional[bool]*) –
- **color_order** (*Union[None, depthai.ColorCameraProperties.ColorOrder, str]*) –
- **manual_focus** (*Optional[int]*) –
- **af_mode** (*Optional[depthai.RawCameraControl.AutoFocusMode]*) –
- **awb_mode** (*Optional[depthai.RawCameraControl.AutoWhiteBalanceMode]*) –
- **scene_mode** (*Optional[depthai.RawCameraControl.SceneMode]*) –
- **anti_banding_mode** (*Optional[depthai.RawCameraControl.AntiBandingMode]*) –
- **effect_mode** (*Optional[depthai.RawCameraControl.EffectMode]*) –
- **isp_scale** (*Optional[Tuple[int, int]]*) –
- **sharpness** (*Optional[int]*) –
- **luma_denoise** (*Optional[int]*) –
- **chroma_denoise** (*Optional[int]*) –

Return type None

is_replay()

Return type bool

is_color()

Return type bool

is_mono()

Return type bool

get_fps()

Return type float

set_fps(*fps*)

Parameters **fps** (*float*) –

config_encoder_h26x(*rate_control_mode=None*, *keyframe_freq=None*, *bitrate_kbps=None*, *num_b_frames=None*)

Parameters

- **rate_control_mode** (*Optional[depthai.VideoEncoderProperties.RateControlMode]*) –
- **keyframe_freq** (*Optional[int]*) –
- **bitrate_kbps** (*Optional[int]*) –
- **num_b_frames** (*Optional[int]*) –

config_encoder_mjpeg(*quality=None*, *lossless=False*)

Parameters

- **quality** (*Optional[int]*) –
- **lossless** (*bool*) –

get_stream_xout(*fourcc=None*)

Parameters **fourcc** (*Optional[str]*) –

Return type depthai_sdk.oak_outputs.xout.xout_base.StreamXout

set_num_frames_pool(*num_frames*, *preview_num_frames=None*)

Set the number of frames to be stored in the pool.

Parameters

- **num_frames** (*int*) – Number of frames to be stored in the pool.
- **preview_num_frames** (*Optional[int]*) – Number of frames to be stored in the pool for the preview stream.

get_fourcc()

Return type Optional[str]

class Out(*camera_component*)

class CameraOut(*component*)

class ReplayOut(*component*)

class EncodedOut(*component*)

2.2.2 IMUComponent

IMUComponent abstracts IMU node and its configuration.

Usage

```
from depthai_sdk import OakCamera
from depthai_sdk.classes import IMUPacket

with OakCamera() as oak:
    imu = oak.create_imu()
    imu.config_imu(report_rate=400, batch_report_threshold=5)

    def callback(packet: IMUPacket):
        print(packet)
```

(continues on next page)

(continued from previous page)

```
    oak.callback(imu.out.main, callback=callback)
    oak.start(blocking=True)
```

Component outputs

- `main` - Main output, produces *IMUPacket*.

Reference

class `depthai_sdk.components.`**`IMUComponent`**(*device*, *pipeline*)

`get_imu_name`()

Return type str

`config_imu`(*sensors=None*, *report_rate=100*, *batch_report_threshold=1*, *max_batch_reports=10*, *enable_firmware_update=False*)

Configure IMU node.

Parameters

- **`sensors`** (*`Optional[List[depthai.IMUSensor]]`*) – List of sensors to enable.
- **`report_rate`** (*`int`*) – Report rate in Hz.
- **`batch_report_threshold`** (*`int`*) – Number of reports to batch before sending them to the host.
- **`max_batch_reports`** (*`int`*) – Maximum number of batched reports to send to the host.
- **`enable_firmware_update`** (*`bool`*) – Enable firmware update if true, disable otherwise.

Return type None

class `Out`(*imu_component*)

class `ImuOut`(*component*)

2.2.3 NNComponent

NNComponent abstracts sourcing & decoding *AI models*, creating a DepthAI API node for neural inferencing, object tracking, and MultiStage pipelines setup. It also supports Roboflow integration.

DepthAI API nodes

For neural inference, NNComponent will use DepthAI API node:

- If we are using MobileNet-SSD based AI model, this component will create MobileNetDetectionNetwork (or MobileNetSpatialDetectionNetwork if `spatial` argument is set).
- If we are using YOLO based AI model, this component will create YoloDetectionNetwork (or YoloSpatialDetectionNetwork if `spatial` argument is set).
- If it's none of the above, component will create NeuralNetwork node.

If `tracker` argument is set and we have YOLO/MobileNet-SSD based model, this component will also create ObjectTracker node, and connect the two nodes togeter.

Usage

```
from depthai_sdk import OakCamera, ResizeMode

with OakCamera(recording='cars-tracking-above-01') as oak:
    color = oak.create_camera('color')
    nn = oak.create_nn('vehicle-detection-0202', color, tracker=True)
    nn.config_nn(resize_mode='stretch')

    oak.visualize([nn.out.tracker, nn.out.passthrough], fps=True)
    oak.start(blocking=True)
```

Component outputs

- `main` - Default output. Streams NN results and high-res frames that were downscaled and used for inferencing. Produces *DetectionPacket* or *TwoStagePacket* (if it's 2. stage NNComponent).
- `passthrough` - Default output. Streams NN results and passthrough frames (frames used for inferencing). Produces *DetectionPacket* or *TwoStagePacket* (if it's 2. stage NNComponent).
- `spatials` - Streams depth and bounding box mappings (`SpatialDetectionNework.boundingBoxMapping`). Produces *SpatialBbMappingPacket*.
- `twostage_crops` - Streams 2. stage cropped frames to the host. Produces *FramePacket*.
- `tracker` - Streams ObjectTracker's tracklets and high-res frames that were downscaled and used for inferencing. Produces *TrackerPacket*.
- `nn_data` - Streams NN raw output. Produces *NNDataPacket*.

Decoding outputs

NNComponent allows user to define their own decoding functions. There is a set of standardized outputs:

- *`Detections`*
- *`SemanticSegmentation`*
- *`ImgLandmarks`*
- *`InstanceSegmentation`*

Note: This feature is still in development and is not guaranteed to work correctly in all cases.

Example usage:

```
import numpy as np
from depthai import NNData

from depthai_sdk import OakCamera
from depthai_sdk.classes import Detections

def decode(nn_data: NNData):
    layer = nn_data.getFirstLayerFp16()
    results = np.array(layer).reshape((1, 1, -1, 7))
    dets = Detections(nn_data)

    for result in results[0][0]:
        if result[2] > 0.5:
            dets.add(result[1], result[2], result[3:])

    return dets


def callback(packet: DetectionPacket, visualizer: Visualizer):
    detections: Detections = packet.img_detections
    ...


with OakCamera() as oak:
    color = oak.create_camera('color')

    nn = oak.create_nn(..., color, decode_fn=decode)

    oak.visualize(nn, callback=callback)
    oak.start(blocking=True)
```

Reference

class depthai_sdk.components.**NNComponent**(*device*, *pipeline*, *model*, *input*, *nn_type=None*, *decode_fn=None*, *tracker=False*, *spatial=None*, *replay=None*, *args=None*)

get_name()

get_labels()

config_multistage_nn(*debug=False*, *labels=None*, *scale_bb=None*, *num_frame_pool=None*)

Configures the MultiStage NN pipeline. Available if the input to this NNComponent is Detection NNComponent.

Parameters

- **debug** (*bool, default False*) – Debug script node
- **labels** (*List[int], optional*) – Crop & run inference only on objects with these labels

- **scale_bb** (*Tuple[int, int], optional*) – Scale detection bounding boxes (x, y) before cropping the frame. In %.
- **num_frame_pool** (*int, optional*) – Number of frames to pool for inference. If None, will use the default value.

Return type None

config_tracker(*tracker_type=None, track_labels=None, assignment_policy=None, max_obj=None, threshold=None, apply_tracking_filter=None, forget_after_n_frames=None, calculate_speed=None*)

Configure Object Tracker node (if it's enabled).

Parameters

- **tracker_type** (*dai.TrackerType, optional*) – Set object tracker type
- **track_labels** (*List[int], optional*) – Set detection labels to track
- **assignment_policy** (*dai.TrackerType, optional*) – Set object tracker ID assignment policy
- **max_obj** (*int, optional*) – Set max objects to track. Max 60.
- **threshold** (*float, optional*) – Specify tracker threshold. Default: 0.0
- **apply_tracking_filter** (*bool, optional*) – Set whether to apply Kalman filter to the tracked objects. Done on the host.
- **forget_after_n_frames** (*int, optional*) – Set how many frames to track an object before forgetting it.
- **calculate_speed** (*bool, optional*) – Set whether to calculate object speed. Done on the host.

Return type None

config_yolo_from_metadata(*metadata*)

Configures (Spatial) Yolo Detection Network node with a dictionary. Calls config_yolo().

Parameters **metadata** (*Dict*) –

Return type None

config_yolo(*num_classes, coordinate_size, anchors, masks, iou_threshold, conf_threshold=None*)

Configures (Spatial) Yolo Detection Network node.

Parameters

- **num_classes** (*int*) –
- **coordinate_size** (*int*) –
- **anchors** (*List[float]*) –
- **masks** (*Dict[str, List[int]]*) –
- **iou_threshold** (*float*) –
- **conf_threshold** (*Optional[float]*) –

Return type None

config_nn(*conf_threshold=None, resize_mode=None*)

Configures the Detection Network node.

Parameters

- **conf_threshold** (*Optional[float]*) – (float, optional): Confidence threshold for the detections (0..1]
- **resize_mode** (*Optional[Union[depthai_sdk.classes.enum.ResizeMode, str]]*) – (ResizeMode, optional): Change aspect ratio resizing mode - to either STRETCH, CROP, or LETTERBOX.

Return type None

config_spatial(*bb_scale_factor=None*, *lower_threshold=None*, *upper_threshold=None*, *calc_algo=None*)

Configures the Spatial Detection Network node.

Parameters

- **bb_scale_factor** (*float, optional*) – Specifies scale factor for detected bounding boxes (0..1]
- **lower_threshold** (*int, optional*) – Specifies lower threshold in depth units (millimeter by default) for depth values which will used to calculate spatial data
- **upper_threshold** (*int, optional*) – Specifies upper threshold in depth units (millimeter by default) for depth values which will used to calculate spatial data
- **calc_algo** (*dai.SpatialLocationCalculatorAlgorithm, optional*) – Specifies spatial location calculator algorithm: Average/Min/Max

Return type None

get_bbox()

Return type depthai_sdk.visualize.bbox.BoundingBox

class Out(*nn_component*)

class MainOut(*component*)

Default output. Streams NN results and high-res frames that were downscaled and used for inferencing. Produces DetectionPacket or TwoStagePacket (if it's 2. stage NNComponent).

class PassThroughOut(*component*)

class ImgManipOut(*component*)

class InputOut(*component*)

class SpatialOut(*component*)

class TwoStageOut(*component*)

class TrackerOut(*component*)

class EncodedOut(*component*)

class NnDataOut(*component*)

is_spatial()

Return type bool

is_tracker()

Return type bool

is_yolo()

Return type bool

is_mobile_net()

Return type bool

is_detector()

Currently these 2 object detectors are supported

Return type bool

is_multi_stage()

General (standarized) NN outputs, to be used for higher-level abstractions (eg. automatic visualization of results). "SDK supported NN models" will have to have standard NN output, so either dai.ImgDetections, or one of the outputs below. If the latter, model json config will incldue handler.py logic for decoding to the standard NN output. These will be integrated into depthai-core, bonus points for on-device decoding of some popular models.

class depthai_sdk.classes.nn_results.**Detection**(*img_detection: Union[NoneType, depthai.ImgDetection, depthai.SpatialImgDetection], label_str: str, confidence: float, color: Tuple[int, int, int], bbox: depthai_sdk.visualize.bbox.BoundingBox, angle: Union[int, NoneType], ts: Union[datetime.timedelta, NoneType]*)

img_detection: Union[None, depthai.ImgDetection, depthai.SpatialImgDetection]

label_str: str

confidence: float

color: Tuple[int, int, int]

bbox: depthai_sdk.visualize.bbox.BoundingBox

angle: Optional[int]

ts: Optional[datetime.timedelta]

property top_left

property bottom_right

class depthai_sdk.classes.nn_results.**TrackingDetection**(*img_detection: Union[NoneType, depthai.ImgDetection, depthai.SpatialImgDetection], label_str: str, confidence: float, color: Tuple[int, int, int], bbox: depthai_sdk.visualize.bbox.BoundingBox, angle: Union[int, NoneType], ts: Union[datetime.timedelta, NoneType], tracklet: depthai.Tracklet, filtered_2d: depthai_sdk.visualize.bbox.BoundingBox, filtered_3d: depthai.Point3f, speed: Union[float, NoneType]*)

tracklet: depthai.Tracklet

filtered_2d: depthai_sdk.visualize.bbox.BoundingBox

filtered_3d: depthai.Point3f

speed: Optional[float]

property speed_kmph

property speed_mph

class depthai_sdk.classes.nn_results.**TwoStageDetection**(*img_detection: Union[NoneType, depthai.ImgDetection, depthai.SpatialImgDetection], label_str: str, confidence: float, color: Tuple[int, int, int], bbox: depthai_sdk.visualize.bbox.BoundingBox, angle: Union[int, NoneType], ts: Union[datetime.timedelta, NoneType], nn_data: depthai.NNData*)

nn_data: depthai.NNData

class depthai_sdk.classes.nn_results.**GenericNNOutput**(*nn_data*)

Generic NN output, to be used for higher-level abstractions (eg. automatic visualization of results).

getTimestamp()

Return type datetime.timedelta

getSequenceNum()

Return type int

class depthai_sdk.classes.nn_results.**ExtendedImgDetection**(*angle: int*)

class depthai_sdk.classes.nn_results.**Detections**(*nn_data, is_rotated=False*)

Detection results containing bounding boxes, labels and confidences. Optionally can contain rotation angles.

class depthai_sdk.classes.nn_results.**SemanticSegmentation**(*nn_data, mask*)

Semantic segmentation results, with a mask for each class.

Examples: *DeeplabV3*, *Lanenet*, *road-segmentation-adas-0001*.

mask: List[numpy.ndarray]

class depthai_sdk.classes.nn_results.**ImgLandmarks**(*nn_data, landmarks=None, landmarks_indices=None, pairs=None, colors=None*)

Landmarks results, with a list of landmarks and pairs of landmarks to draw lines between.

Examples: *human-pose-estimation-0001*, *openpose2*, *facial-landmarks-68*, *landmarks-regression-retail-0009*.

class depthai_sdk.classes.nn_results.**InstanceSegmentation**(*nn_data, masks, labels*)

Instance segmentation results, with a mask for each instance.

masks: List[numpy.ndarray]

labels: List[int]

2.2.4 StereoComponent

`StereoComponent` abstracts StereoDepth node, its configuration, filtering (eg. WLS filter), and disparity/depth viewing.

Usage

```
from depthai_sdk import OakCamera

with OakCamera() as oak:
    # Create stereo component, initialize left/right MonoCamera nodes for 800P and␣
↪60FPS
    stereo = oak.create_stereo('800p', fps=60)

    # Visualize normalized and colorized disparity stream
    oak.visualize(stereo.out.depth)
    # Start the pipeline, continuously poll
    oak.start(blocking=True)
```

Component outputs

- `main` - Default output. Uses `depth`.
- `disparity` - Streams StereoDepth's disparity frames to the host. When visualized, these get normalized and colorized. Produces *DepthPacket*.
- `depth` - Streams StereoDepth's depth frames to the host. When visualized, depth gets converted to disparity (for nicer visualization), normalized and colorized. Produces *DepthPacket*.
- `rectified_left` - Streams StereoDepth's rectified left frames to the host.
- `rectified_right` - Streams StereoDepth's rectified right frames to the host.
- `encoded` - Provides an encoded version of `disparity` stream.

Reference

class depthai_sdk.components.**StereoComponent**(*device*, *pipeline*, *left*, *right*, *replay=None*, *args=None*, *encode=None*)

property depth

property disparity

on_pipeline_started(*device*)

This function gets called after the pipeline has been started. It is called from the main thread. It can be used to eg. initialize XlinkIn queues.

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

config_undistortion(*M2_offset=0*)

Parameters **M2_offset** (*int*) –

config_stereo(*confidence=None*, *align=None*, *median=None*, *extended=None*, *subpixel=None*, *lr_check=None*, *sigma=None*, *lr_check_threshold=None*, *subpixel_bits=None*)

Configures StereoDepth modes and options.

Parameters

- **confidence** (*Optional[int]*) –
- **align** (*Optional[depthai_sdk.components.camera_component.CameraComponent]*) –
- **median** (*Union[None, int, depthai.MedianFilter]*) –
- **extended** (*Optional[bool]*) –
- **subpixel** (*Optional[bool]*) –
- **lr_check** (*Optional[bool]*) –
- **sigma** (*Optional[int]*) –
- **lr_check_threshold** (*Optional[int]*) –
- **subpixel_bits** (*Optional[int]*) –

Return type None

config_postprocessing (*colorize=None*, *colormap=None*)

Configures postprocessing options.

Parameters

- **colorize** (*Optional[Union[depthai_sdk.visualize.configs.StereoColor, bool]]*) – Colorize the disparity map. Can be either a StereoColor enum, string or bool.
- **colormap** (*Optional[int]*) – Colormap to use for colorizing the disparity map.

Return type None

config_wls (*wls_level=None*, *wls_lambda=None*, *wls_sigma=None*)

Configures WLS filter options.

Parameters

- **wls_level** (*Optional[Union[depthai_sdk.components.stereo_component.WLSLevel, str]]*) – WLS filter level. Can be either a WLSLevel enum or string.
- **wls_lambda** (*Optional[float]*) – WLS filter lambda.
- **wls_sigma** (*Optional[float]*) – WLS filter sigma.

Return type None

set_colormap (*colormap*)

Sets the colormap to use for colorizing the disparity map. Used for on-device postprocessing. Works only with *encoded* output. Note: This setting can affect the performance.

Parameters **colormap** (*depthai.Colormap*) – Colormap to use for colorizing the disparity map.

set_auto_ir (*auto_mode*, *continuous_mode=False*)

Enables/disables auto IR dot projector and flood brightness. Selects the best IR brightness level automatically. Can be set to continious mode, which will continuously adjust the IR brightness. Otherwise, it will adjust the brightness only once when the device is started.

Parameters

- **auto_mode** (*bool*) – Enable/disable auto IR.

- **continuous_mode** (*bool*) – Enable/disable continious mode.

Return type None

set_ir(*dot_projector_brightness=None*, *flood_brightness=None*)

Sets IR brightness and flood.

Parameters

- **dot_projector_brightness** (*Optional[int]*) –
- **flood_brightness** (*Optional[int]*) –

get_fourcc()

Return type Optional[str]

class Out(*stereo_component*)

class DepthOut(*component*)

class DisparityOut(*component*)

class RectifiedLeftOut(*component*)

class RectifiedRightOut(*component*)

class EncodedOut(*component*)

CHAPTER

THREE

PACKETS

Packets are **synchronized collections** of one or more DepthAI messages. They are used **internally for visualization** and also forwarded to the callback function if the user:

1. Specified a callback for visualizing of an output via `OakCamera.visualize(..., callback=fn)`.
2. Used callback output via `OakCamera.callback(..., callback=fn, enable_visualizer=True)`.

3.1 API Usage

1. `OakCamera.visualize()`: In the example below SDK won't show the frame to the user, but instead it will send the packet to the callback function. SDK will draw detections (bounding boxes, labels) on the `packet.frame`.
2. `OakCamera.callback()`: This will also send *`DetectionPacket`* to the callback function, the only difference is that the SDK won't draw on the frame, so you can draw detections on the frame yourself.

Note: If you specify callback function in `OakCamera.visualize()`, you need to trigger drawing of detections yourself via `Visualizer.draw()` method.

```
import cv2
from depthai_sdk import OakCamera
from depthai_sdk.classes import DetectionPacket

with OakCamera() as oak:
    color = oak.create_camera('color')
    nn = oak.create_nn('mobilenet-ssd', color)

    # Callback
    def cb(packet: DetectionPacket):
        print(packet.img_detections)
        cv2.imshow(packet.name, packet.frame)

    # 1. Callback after visualization:
    oak.visualize(nn.out.main, fps=True, callback=cb)

    # 2. Callback:
    oak.callback(nn.out.main, callback=cb, enable_visualizer=True)

    oak.start(blocking=True)
```

3.2 Reference

3.2.1 FramePacket

class depthai_sdk.classes.packets.**FramePacket**(*name*, *msg*)

Contains only dai.ImgFrame message and cv2 frame, which is used by visualization logic.

property frame

get_timestamp()

Return type datetime.timedelta

get_sequence_num()

Return type int

set_decode_codec(*get_codec*)

Parameters **get_codec** (*Callable*) –

decode()

Return type Optional[numpy.ndarray]

get_size()

Return type Tuple[int, int]

3.2.2 SpatialBbMappingPacket

class depthai_sdk.classes.packets.**SpatialBbMappingPacket**(*name*, *msg*, *spatials*, *disp_scale_factor*)

Output from Spatial Detection nodes - depth frame + bounding box mappings. Inherits FramePacket.

prepare_visualizer_objects(*vis*)

Prepare visualizer objects (boxes, lines, text, etc.), so visualizer can draw them on the frame.

Parameters

- **visualizer** – Visualizer object.
- **vis** (depthai_sdk.visualize.visualizer.Visualizer) –

Return type None

3.2.3 DetectionPacket

class depthai_sdk.classes.packets.**DetectionPacket**(*name*, *msg*, *dai_msg*, *bbox*)

Output from Detection Network nodes - image frame + image detections. Inherits FramePacket.

prepare_visualizer_objects(*vis*)

Prepare visualizer objects (boxes, lines, text, etc.), so visualizer can draw them on the frame.

Parameters

- **visualizer** – Visualizer object.
- **vis** (depthai_sdk.visualize.visualizer.Visualizer) –

Return type None

3.2.4 NNDataPacket

class depthai_sdk.classes.packets.**NNDataPacket**(*name*, *nn_data*)

Contains only dai.NNData message

get_timestamp()

Return type datetime.timedelta

get_sequence_num()

Return type int

3.2.5 DepthPacket

class depthai_sdk.classes.packets.**DepthPacket**(*name*, *msg*)

3.2.6 TrackerPacket

class depthai_sdk.classes.packets.**TrackerPacket**(*name*, *msg*, *tracklets*, *bbox*)

Output of Object Tracker node. Tracklets + Image frame. Inherits FramePacket.

prepare_visualizer_objects(*visualizer*)

Prepare visualizer objects (boxes, lines, text, etc.), so visualizer can draw them on the frame.

Parameters **visualizer** (depthai_sdk.visualize.visualizer.Visualizer) – Visualizer object.

Return type None

3.2.7 TwoStagePacket

class depthai_sdk.classes.packets.**TwoStagePacket**(*name*, *msg*, *img_detections*, *nn_data*, *labels*, *bbox*)

Output of 2-stage NN pipeline; Image frame, Image detections and multiple NNData results. Inherits DetectionPacket.

3.2.8 IMUPacket

class depthai_sdk.classes.packets.**IMUPacket**(*name*, *packet*, *rotation=None*)

get_imu_vals()

Returns imu values in a tuple. Returns in format (accelerometer_values, gyroscope_values, quaternion, magnetometer_values)

Return type Tuple[Sequence, Sequence, Sequence, Sequence]

get_timestamp()

Return type datetime.timedelta

get_sequence_num()

Return type int

CHAPTER

FOUR

VISUALIZER

DepthAI SDK visualizer serves as a tool to visualize the output of the DepthAI pipeline.

It can be used to visualize the output of the camera, neural network, depth and disparity map, the rectified streams, the spatial location of the detected objects, and more.

4.1 Getting Started

Visualizer is created upon calling `OakCamera.visualize()`, which returns *Visualizer* instance. Once it is created, the visualizer configs can be modified using *output()*, *stereo()*, *text()*, *detections()*, *tracking()* methods.

Example how *Visualizer* can be created:

```
from depthai_sdk import OakCamera

with OakCamera() as oak:
    cam = oak.create_camera('color')
    visualizer = oak.visualize(cam.out.main)
    oak.start(blocking=True)
```

Visualizer is primarily used alongside with *Packets* in `depthai_sdk.oak_outputs` module.

4.2 Configs

Visualizer is configurable via *VisConfig* that consists of five auxiliary configs: *OutputConfig*, *StereoConfig*, *TextConfig*, *DetectionConfig*, and *TrackingConfig*. Each config's type has its own set of parameters, which effects how the corresponding object will be visualized.

There are the following methods for modifying the default configuration: *output()*, *stereo()*, *text()*, *detections()*, *tracking()*. The arguments should be passed as key-value arguments with the same signature as the corresponding config, e.g., *Visualizer.text(font_size=2, font_color=(255,123,200))*.

The modified configuration will be applied to every created objects. The methods support fluent interface and can be chained, e.g., *Visualizer.text(font_size=2).detections(color=(255, 0, 0))*.

Example how to configure the visualizer:

```
visualizer = oak.visualize(camera.out.main)
visualizer.detections(
    bbox_style=BboxStyle.RECTANGLE,
    label_position=TextPosition.MID,
```

(continues on next page)

(continued from previous page)

```
).text(
    auto_scale=True
)
```

4.3 Objects

Visualizer operates with objects. Objects can be seen as a hierarchical structure. The root object is `self`, and the children are the list of the created objects. *add_child* should be used to add the object to the children list. The parent object shares the config and frame shape with all children.

All objects must be derived from *GenericObject*.

Implemented objects:

- *VisDetections*,
- *VisText*,
- *VisLine*,
- `VisCircle`,
- *VisTrail*.

Objects can be added to the visualizer using the following methods:

- *add_text()*,
- *add_detections()*,
- *add_trail()*,
- *add_circle()*,
- *add_line()*.

4.4 Create your own object

If the provided functionality is not enough, you can create your own object. To do so, you need to create a class derived from *GenericObject* and implement the *prepare*, *serialize*, and `draw` methods. The `draw` method should draw the object on the passed `frame` argument.

```
class YourOwnObject:
    def __init__(self, ...):
        ...

    def prepare(self) -> None:
        ...

    def serialize(self) -> str:
        ...

    def draw(self, frame) -> None:
        ...

with OakCamera() as oak:
```

(continues on next page)

(continued from previous page)

```
    cam = oak.create_camera(...)
    visualizer = cam.visualize(cam.out.main)
    visualizer.add_object(YourOwnObject(...))
```

4.5 Example usage

The following script will visualize the output of face detection model:

```
from depthai_sdk import OakCamera
from depthai_sdk.visualize.configs import BboxStyle, TextPosition

with OakCamera() as oak:
    camera = oak.create_camera('color')

    det = oak.create_nn('face-detection-retail-0004', camera)

    visualizer = oak.visualize(det.out.main, fps=True)
    visualizer.detections(
        color=(0, 255, 0),
        thickness=2,
        bbox_style=BboxStyle.RECTANGLE,
        label_position=TextPosition.MID,
    ).text(
        font_color=(255, 255, 0),
        auto_scale=True
    ).tracking(
        line_thickness=5
    )

    oak.start(blocking=True)
```

4.6 Serialization

The visualizer provides a way to serialize the output objects to JSON, which can be used for further processing.

4.6.1 JSON schemas

General config

```
{
  "frame_shape": {
    "type": "array",
    "items": {
      "type": "integer"
    },
    "description": "Frame shape in (height, width) format."
  },
  "config": {
    "type": "object",
```

(continues on next page)

(continued from previous page)

```
    "output": {
      "img_scale": {
        "type": "number",
        "minimum": 0.0,
        "maximum": 1.0,
        "default": 1.0,
        "description": "Scale of output image."
      },
      "show_fps": {
        "type": "boolean",
        "default": false,
        "description": "Show FPS on output image."
      },
      "clickable": {
        "type": "boolean",
        "default": false,
        "description": "Show disparity or depth value on mouse hover."
      }
    },
    "stereo": {
      "type": "object",
      "colorize": {
        "type": "integer",
        "default": 2,
        "description": "cv2 colormap."
      },
      "colormap": {
        "type": "integer",
        "default": 2,
        "description": "0 - gray, 1 - color, 2 - blended color and depth."
      },
      "wls_filter": {
        "type": "boolean",
        "default": false
      },
      "wls_lambda": {
        "type": "number",
        "default": 8000.0
      },
      "wls_sigma": {
        "type": "number",
        "default": 1.5
      }
    },
    "detection": {
      "type": "object",
      "thickness": {
        "type": "integer",
        "default": 1
      },
      "fill_transparency": {
        "type": "number",
        "default": 0.15,
        "minimum": 0.0,
        "maximum": 1.0,
        "description": "Transparency of bbox fill."
      },
```

(continues on next page)

(continued from previous page)

```
      "box_roundness": {
        "type": "integer",
        "default": 0,
        "description": "Roundness of bbox corners, used only when bbox_style is set␣
↪to BboxStyle.ROUNDED_*."
      },
      "color": {
        "type": "array",
        "items": {
          "type": "integer"
        },
        "default": [0, 255, 0],
        "description": "Default bbox color in RGB format."
      },
      "bbox_style": {
        "type": "integer",
        "default": 0,
        "description": "``depthai_sdk.visualize.configs.BBoxStyle`` enum value."
      },
      "line_width": {
        "type": "number",
        "default": 0.5,
        "minimum": 0.0,
        "maximum": 1.0,
        "description": "Horizontal line width of bbox."
      },
      "line_height": {
        "type": "number",
        "default": 0.5,
        "minimum": 0.0,
        "maximum": 1.0,
        "description": "Vertical line height of bbox."
      },
      "hide_label": {
        "type": "boolean",
        "default": false,
        "description": "Hide class label on output image."
      },
      "label_position": {
        "type": "integer",
        "default": 0,
        "description": "``depthai_sdk.visualize.configs.TextPosition`` enum value."
      },
      "label_padding": {
        "type": "integer",
        "default": 10,
        "description": "Padding between label and bbox."
      }
    },
    "text": {
      "font_face": {
        "type": "integer",
        "default": 0,
        "description": "cv2 font face."
      },
      "font_color": {
        "type": "array",
```

(continues on next page)

(continued from previous page)

```
        "items": {
          "type": "integer"
        },
        "default": [255, 255, 255],
        "description": "Font color in RGB format."
      },
      "font_transparency": {
        "type": "number",
        "default": 0.5,
        "minimum": 0.0,
        "maximum": 1.0
      },
      "font_scale": {
        "type": "number",
        "default": 1.0
      },
      "font_thickness": {
        "type": "integer",
        "default": 2
      },
      "font_position": {
        "type": "integer",
        "default": 0,
        "description": "``depthai_sdk.visualize.configs.TextPosition`` enum value."
      },
      "bg_transparency": {
        "type": "number",
        "default": 0.5,
        "minimum": 0.0,
        "maximum": 1.0,
        "description": "Text outline transparency."
      },
      "bg_color": {
        "type": "array",
        "items": {
          "type": "integer"
        },
        "default": [0, 0, 0],
        "description": "Text outline color in RGB format."
      },
      "line_type": {
        "type": "integer",
        "default": 16,
        "description": "cv2 line type."
      },
      "auto_scale": {
        "type": "boolean",
        "default": true,
        "description": "Automatically scale font size based on bbox size."
      }
    },
    "tracking": {
      "max_length": {
        "type": "integer",
        "default": -1,
        "description": "Maximum length of tracking line, -1 for infinite."
      },
```

(continues on next page)

(continued from previous page)

```
      "deletion_lost_threshold": {
        "type": "integer",
        "default": 5,
        "description": "Number of frames after which lost track is deleted."
      },
      "line_thickness": {
        "type": "integer",
        "default": 1
      },
      "fading_tails": {
        "type": "boolean",
        "default": false,
        "description": "Enable fading tails - reduces line thickness over time."
      },
      "line_color": {
        "type": "array",
        "items": {
          "type": "integer"
        },
        "default": [255, 255, 255],
        "description": "Tracking line color in RGB format."
      },
      "line_type": {
        "type": "integer",
        "default": 16,
        "description": "cv2 line type."
      }
    },
    "circle": {
      "thickness": {
        "type": "integer",
        "default": 1
      },
      "color": {
        "type": "array",
        "items": {
          "type": "integer"
        },
        "default": [0, 255, 0],
        "description": "Circle color in RGB format."
      },
      "line_type": {
        "type": "integer",
        "default": 16,
        "description": "cv2 line type."
      }
    }
  },
  "objects": {
    "type": "array",
    "items": {
      "type": "object"
    },
    "description": "Array of objects (e.g. detection, text, line).",
    "default": []
  }
}
```

Objects

- Detection:

```
{
  "type": "detections",
  "detections": {
    "type": "array",
    "items": {
      "type": "object",
      "bbox": {
        "type": "array",
        "items": {
          "type": "number"
        },
        "description": "bbox absolute coordinates in format [x1, y1, x2, y2]"
      },
      "label": {
        "type": "string",
        "description": "class label"
      },
      "color": {
        "type": "array",
        "items": {
          "type": "integer"
        },
        "description": "bbox color in RGB format"
      }
    }
  },
  "children": {
    "type": "array",
    "items": {
      "type": "object"
    },
    "description": "array of child objects (e.g. detection, text, line)",
    "default": []
  }
}
```

- Text:

```
{
  "type": "text",
  "text": {
    "type": "plain_text"
  },
  "coords": {
    "type": "array",
    "items": {
      "type": "number"
    },
    "description": "The absolute coordinates of the text in the format (x1, y1)."
  }
}
```

- Line:

```
{
  "type": "line",
  "pt1": {
    "type": "array",
    "items": {
      "type": "number"
    },
    "description": "Absolute (x, y) coordinates of the first point."
  },
  "pt2": {
    "type": "array",
    "items": {
      "type": "number"
    },
    "description": "Absolute (x, y) coordinates of the second point."
  },
  "children": {
    "type": "array",
    "items": {
      "type": "object"
    },
    "description": "array of child objects (e.g. detection, text, line).",
    "default": []
  }
}
```

Example JSON output

```
{
  "frame_shape": [720, 1280],
  "config": {
    "output": {
      "img_scale": 1.0,
      "show_fps": false,
      "clickable": true
    },
    "stereo": {
      "colorize": 2,
      "colormap": 2,
      "wls_filter": false,
      "wls_lambda": 8000,
      "wls_sigma": 1.5
    },
    "detection": {
      "thickness": 1,
      "fill_transparency": 0.15,
      "box_roundness": 0,
      "color": [0, 255, 0],
      "bbox_style": 0,
      "line_width": 0.5,
      "line_height": 0.5,
      "hide_label": false,
      "label_position": 0,
      "label_padding": 10
    },
    "text": {
```

(continues on next page)

(continued from previous page)

```
      "font_face": 0,
      "font_color": [255, 255, 255],
      "font_transparency": 0.5,
      "font_scale": 1.0,
      "font_thickness": 2,
      "font_position": 0,
      "bg_transparency": 0.5,
      "bg_color": [0, 0, 0],
      "line_type": 16,
      "auto_scale": true
    },
    "tracking": {
      "max_length": -1,
      "deletion_lost_threshold": 5,
      "line_thickness": 1,
      "fading_tails": false,
      "line_color": [255, 255, 255],
      "line_type": 16
    },
    "circle": {
      "thickness": 1,
      "color": [255, 255, 255],
      "line_type": 16
    }
  },
  "objects": [
    {
      "type": "detections",
      "detections": [
        {
          "bbox": [101, 437, 661, 712],
          "label": "laptop",
          "color": [210, 167, 218]
        }
      ],
      "children": [
        {
          "type": "text",
          "text": "Laptop",
          "coords": [111, 469]
        }
      ]
    }
  ]
}
```

4.7 References

class depthai_sdk.visualize.visualizer.**Visualizer**(*scale=None*, *fps=False*)

add_object(*obj*)

Call *set_config*, *set_frame_shape* and *prepare* for the object and add it to the list of objects. :param obj: The object to add.

Returns self

Parameters **obj** (depthai_sdk.visualize.objects.GenericObject) –

Return type *depthai_sdk.visualize.visualizer.Visualizer*

add_bbox(*bbox*, *color=None*, *thickness=None*, *bbox_style=None*, *label=None*)

Add a bounding box to the visualizer.

Parameters

- **bbox** (*depthai_sdk.visualize.bbox.BoundingBox*) – Bounding box.
- **label** (*Optional[str]*) – Label for the detection.
- **thickness** (*Optional[int]*) – Bounding box thickness.
- **color** (*Optional[Tuple[int, int, int]]*) – Bounding box color (RGB).
- **bbox_style** (*Optional[depthai_sdk.visualize.configs.BboxStyle]*) – Bounding box style (one of depthai_sdk.visualize.configs.BboxStyle).

Returns self

Return type *depthai_sdk.visualize.visualizer.Visualizer*

add_detections(*detections*, *normalizer=None*, *label_map=None*, *spatial_points=None*, *label_color=None*, *label_background_color=None*, *label_background_transparency=None*, *is_spatial=False*, *bbox=None*)

Add detections to the visualizer.

Parameters

- **detections** (*List[Union[depthai.ImgDetection, depthai.Tracklet]]*) – List of detections.
- **normalizer** (*Optional[depthai_sdk.visualize.bbox.BoundingBox]*) – Normalizer object.
- **label_map** (*Optional[List[Tuple[str, Tuple]]]*) – List of tuples (label, color).
- **spatial_points** (*Optional[List[depthai.Point3f]]*) – List of spatial points. None if not spatial.
- **label_color** (*Optional[Tuple[int, int, int]]*) – Color for the label.
- **label_background_color** (*Optional[Tuple[int, int, int]]*) – Color for the label background.
- **label_background_transparency** (*Optional[float]*) – Transparency for the label background.
- **is_spatial** – Flag that indicates if the detections are spatial.

- **bbox** (*Optional[Union[numpy.ndarray, Tuple[int, int, int, int]]]*) – Bounding box, if there's a detection inside a bounding box.

Returns self

Return type *depthai_sdk.visualize.visualizer.Visualizer*

add_text(*text*, *coords=None*, *size=None*, *color=None*, *thickness=None*, *outline=True*, *background_color=None*, *background_transparency=0.5*, *bbox=None*, *position=<TextPosition.TOP_LEFT: 0>*, *padding=10*)

Add text to the visualizer.

Parameters

- **text** (*str*) – Text.
- **coords** (*Optional[Tuple[int, int]]*) – Coordinates.
- **size** (*Optional[int]*) – Size of the text.
- **color** (*Optional[Tuple[int, int, int]]*) – Color of the text.
- **thickness** (*Optional[int]*) – Thickness of the text.
- **outline** (*bool*) – Flag that indicates if the text should be outlined.
- **background_color** (*Optional[Tuple[int, int, int]]*) – Background color.
- **background_transparency** (*float*) – Background transparency.
- **bbox** (*Optional[Union[numpy.ndarray, Tuple, depthai_sdk.visualize.bbox.BoundingBox]]*) – Bounding box.
- **position** (*depthai_sdk.visualize.configs.TextPosition*) – Position.
- **padding** (*int*) – Padding.

Returns self

Return type *depthai_sdk.visualize.visualizer.Visualizer*

add_trail(*tracklets*, *label_map*, *bbox=None*)

Add a trail to the visualizer.

Parameters

- **tracklets** (*List[depthai.Tracklet]*) – List of tracklets.
- **label_map** (*List[Tuple[str, Tuple]]*) – List of tuples (label, color).
- **bbox** (*Optional[depthai_sdk.visualize.bbox.BoundingBox]*) – Bounding box.

Returns self

Return type *depthai_sdk.visualize.visualizer.Visualizer*

add_circle(*coords*, *radius*, *color=None*, *thickness=None*)

Add a circle to the visualizer.

Parameters

- **coords** (*Tuple[int, int]*) – Center of the circle.
- **radius** (*int*) – Radius of the circle.
- **color** (*Optional[Tuple[int, int, int]]*) – Color of the circle.

- **thickness** (*Optional[int]*) – Thickness of the circle.

Returns self

Return type *depthai_sdk.visualize.visualizer.Visualizer*

add_line (*pt1*, *pt2*, *color=None*, *thickness=None*)
Add a line to the visualizer.

Parameters

- **pt1** (*Tuple[int, int]*) – Start coordinates.
- **pt2** (*Tuple[int, int]*) – End coordinates.
- **color** (*Optional[Tuple[int, int, int]]*) – Color of the line.
- **thickness** (*Optional[int]*) – Thickness of the line.

Returns self

Return type *depthai_sdk.visualize.visualizer.Visualizer*

add_mask (*mask*, *alpha*)
Add a mask to the visualizer.

Parameters

- **mask** (*numpy.ndarray*) – Mask represented as uint8 numpy array.
- **alpha** (*float*) – Transparency of the mask.

Returns self

drawn (*frame*)
Draw all objects on the frame if the platform is PC. Otherwise, serialize the objects and communicate with the RobotHub application.

Parameters **frame** (*numpy.ndarray*) – The frame to draw on.

Returns np.ndarray if the platform is PC, None otherwise.

Return type Optional[numpy.ndarray]

show (*packet*)
Show the packet on the screen.

serialize (*force_reset=True*)
Serialize all contained objects to JSON.

Parameters **force_reset** (*bool*) – Flag that indicates if the objects should be cleared after serialization.

Returns Stringified JSON.

Return type str

reset ()
Clear all objects.

output (*img_scale=None*, *show_fps=None*)
Configure the output of the visualizer.

Parameters

- **img_scale** (*Optional[float]*) – Scale of the output image.
- **show_fps** (*Optional[bool]*) – Flag that indicates if the FPS should be shown.

Returns self

Return type *depthai_sdk.visualize.visualizer.Visualizer*

stereo(*colorize=None*, *colormap=None*, *wls_filter=None*, *wls_lambda=None*, *wls_sigma=None*)

Parameters

- **colorize** (*Optional[depthai_sdk.visualize.configs.StereoColor]*) –
- **colormap** (*Optional[int]*) –
- **wls_filter** (*Optional[bool]*) –
- **wls_lambda** (*Optional[float]*) –
- **wls_sigma** (*Optional[float]*) –

detections(*thickness=None*, *fill_transparency=None*, *bbox_roundness=None*, *color=None*, *bbox_style=None*, *line_width=None*, *line_height=None*, *hide_label=None*, *label_position=None*, *label_padding=None*)

Configure how bounding boxes will look like. :param thickness: Thickness of the bounding box. :param fill_transparency: Transparency of the bounding box. :param bbox_roundness: Roundness of the bounding box. :param color: Color of the bounding box. :param bbox_style: Style of the bounding box. :param line_width: Width of the bbox horizontal lines CORNERS or ROUNDED_CORNERS style is used. :param line_height: Height of the bbox vertical lines when CORNERS or ROUNDED_CORNERS style is used. :param hide_label: Flag that indicates if the label should be hidden. :param label_position: Position of the label. :param label_padding: Padding of the label.

Returns self

Parameters

- **thickness** (*Optional[int]*) –
- **fill_transparency** (*Optional[float]*) –
- **bbox_roundness** (*Optional[float]*) –
- **color** (*Optional[Tuple[int, int, int]]*) –
- **bbox_style** (*Optional[depthai_sdk.visualize.configs.BboxStyle]*) –
- **line_width** (*Optional[float]*) –
- **line_height** (*Optional[float]*) –
- **hide_label** (*Optional[bool]*) –
- **label_position** (*Optional[depthai_sdk.visualize.configs.TextPosition]*) –
- **label_padding** (*Optional[int]*) –

Return type *depthai_sdk.visualize.visualizer.Visualizer*

text(*font_face=None*, *font_color=None*, *font_transparency=None*, *font_scale=None*, *font_thickness=None*, *font_position=None*, *background_transparency=None*, *background_color=None*, *outline_color=None*, *line_type=None*, *auto_scale=None*)

Configure how text will look like.

Parameters

- **font_face** (*Optional[int]*) – Font face (from cv2).

- **font_color** (*Optional[Tuple[int, int, int]]*) – Font color.
- **font_transparency** (*Optional[float]*) – Font transparency.
- **font_scale** (*Optional[float]*) – Font scale.
- **font_thickness** (*Optional[int]*) – Font thickness.
- **font_position** (*Optional[depthai_sdk.visualize.configs.TextPosition]*) – Font position.
- **background_transparency** (*Optional[float]*) – Text background transparency.
- **background_color** (*Optional[Tuple[int, int, int]]*) – Text background color.
- **outline_color** (*Optional[Tuple[int, int, int]]*) – Outline color.
- **line_type** (*Optional[int]*) – Line type (from cv2).
- **auto_scale** (*Optional[bool]*) – Flag that indicates if the font scale should be automatically adjusted.

Returns self

Return type *depthai_sdk.visualize.visualizer.Visualizer*

tracking(*max_length=None*, *deletion_lost_threshold=None*, *line_thickness=None*, *fading_tails=None*, *show_speed=None*, *line_color=None*, *line_type=None*, *bg_color=None*)

Configure how tracking trails will look like.

Parameters

- **max_length** (*Optional[int]*) – Maximum length of the trail (in pixels).
- **deletion_lost_threshold** (*Optional[int]*) – Number of consequent LOST statuses after which the trail is deleted.
- **line_thickness** (*Optional[int]*) – Thickness of the line.
- **fading_tails** (*Optional[bool]*) – Flag that indicates if the tails should fade.
- **show_speed** (*Optional[bool]*) – Flag that indicates if the speed should be shown.
- **line_color** (*Optional[Tuple[int, int, int]]*) – Color of the line.
- **line_type** (*Optional[int]*) – Type of the line (from cv2).
- **bg_color** (*Optional[Tuple[int, int, int]]*) – Text background color.

Returns self

Return type *depthai_sdk.visualize.visualizer.Visualizer*

segmentation(*mask_alpha=None*)

Parameters **mask_alpha** (*Optional[float]*) –

Return type *depthai_sdk.visualize.visualizer.Visualizer*

property frame_shape

close()

class depthai_sdk.visualize.objects.**GenericObject**(*config=VisConfig(output=OutputConfig(img_scale=1.0, show_fps=False, clickable=True), stereo=StereoConfig(colorize=<StereoColor.RGB: 2>, colormap=array([[[128, 0, 0]], [[132, 0, 0]], [[136, 0, 0]], [[140, 0, 0]], [[144, 0, 0]], [[148, 0, 0]], [[152, 0, 0]], [[156, 0, 0]], [[160, 0, 0]], [[164, 0, 0]], [[168, 0, 0]], [[172, 0, 0]], [[176, 0, 0]], [[180, 0, 0]], [[184, 0, 0]], [[188, 0, 0]], [[192, 0, 0]], [[196, 0, 0]], [[200, 0, 0]], [[204, 0, 0]], [[208, 0, 0]], [[212, 0, 0]], [[216, 0, 0]], [[220, 0, 0]], [[224, 0, 0]], [[228, 0, 0]], [[232, 0, 0]], [[236, 0, 0]], [[240, 0, 0]], [[244, 0, 0]], [[248, 0, 0]], [[252, 0, 0]], [[255, 0, 0]], [[255, 4, 0]], [[255, 8, 0]], [[255, 12, 0]], [[255, 16, 0]], [[255, 20, 0]], [[255, 24, 0]], [[255, 28, 0]], [[255, 32, 0]], [[255, 36, 0]], [[255, 40, 0]], [[255, 44, 0]], [[255, 48, 0]], [[255, 52, 0]], [[255, 56, 0]], [[255, 60, 0]], [[255, 64, 0]], [[255, 68, 0]], [[255, 72, 0]], [[255, 76, 0]], [[255, 80, 0]], [[255, 84, 0]], [[255, 88, 0]], [[255, 92, 0]], [[255, 96, 0]], [[255, 100, 0]], [[255, 104, 0]], [[255, 108, 0]], [[255, 112, 0]], [[255, 116, 0]], [[255, 120, 0]], [[255, 124, 0]], [[255, 128, 0]], [[255, 132, 0]], [[255, 136, 0]], [[255, 140, 0]], [[255, 144, 0]], [[255, 148, 0]], [[255, 152, 0]], [[255, 156, 0]], [[255, 160, 0]], [[255, 164, 0]], [[255, 168, 0]], [[255, 172, 0]], [[255, 176, 0]], [[255, 180, 0]], [[255, 184, 0]], [[255, 188, 0]], [[255, 192, 0]], [[255, 196, 0]], [[255, 200, 0]], [[255, 204, 0]], [[255, 208, 0]], [[255, 212, 0]], [[255, 216, 0]], [[255, 220, 0]], [[255, 224, 0]], [[255, 228, 0]], [[255, 232, 0]], [[255, 236, 0]], [[255, 240, 0]], [[255, 244, 0]], [[255, 248, 0]], [[255, 252, 0]], [[254, 255, 2]], [[250, 255, 6]], [[246, 255, 10]], [[242, 255, 14]], [[238, 255, 18]], [[234, 255, 22]], [[230, 255, 26]], [[226, 255, 30]], [[222, 255, 34]], [[218, 255, 38]], [[214, 255, 42]], [[210, 255, 46]], [[206, 255, 50]], [[202, 255, 54]], [[198, 255, 58]], [[194, 255, 62]], [[190, 255, 66]], [[186, 255, 70]], [[182, 255, 74]], [[178, 255, 78]], [[174, 255, 82]], [[170, 255, 86]], [[166, 255, 90]], [[162, 255, 94]], [[158, 255, 98]], [[154, 255, 102]], [[150, 255,*

Generic object used by visualizer.

set_config(*config*)
Set the configuration for the current object.

Parameters **config** (depthai_sdk.visualize.configs.VisConfig) – instance of VisConfig.

Returns self

Return type *depthai_sdk.visualize.objects.GenericObject*

set_frame_shape(*frame_shape*)
Set the incoming frame shape for the current object.

Parameters **frame_shape** (*Tuple[int, ..]*) – frame shape as a tuple of (height, width, channels).

Returns self

Return type *depthai_sdk.visualize.objects.GenericObject*

prepare()
Prepare necessary data for drawing.

Returns self

Return type *depthai_sdk.visualize.objects.GenericObject*

abstract serialize()
Serialize the object to dict.

Return type dict

add_child(*child*)
Add a child object to the current object.

Parameters **child** (depthai_sdk.visualize.objects.GenericObject) – instance derived from GenericObject.

Returns self

Return type *depthai_sdk.visualize.objects.GenericObject*

property children
Get the children of the current object.

Returns List of children.

class depthai_sdk.visualize.objects.**VisDetections**(*detections*, *normalizer*, *label_map=None*, *label_color=None*, *label_background_color=None*, *label_background_transparency=None*, *spatial_points=None*, *is_spatial=False*, *parent_bbox=None*)
Object that represents detections.

serialize()
Serialize the object to dict.

Return type dict

register_detection(*bbox*, *label*, *color*)
Register a detection.

Parameters

- **bbox** (*Union[Tuple[int, ..], depthai_sdk.visualize.bbox.BoundingBox]*) – Bounding box.
- **label** (*str*) – Label.
- **color** (*Tuple[int, int, int]*) – Color.

Return type None

prepare()

Prepare necessary data for drawing.

Returns self

Return type *depthai_sdk.visualize.objects.VisDetections*

get_detections()

Get detections.

Returns List of tuples (bbox, label, color).

Return type List[Tuple[numpy.ndarray, str, Tuple[int, int, int]]]

class depthai_sdk.visualize.objects.**VisText**(*text, coords=None, size=None, color=None, thickness=None, outline=True, background_color=None, background_transparency=0.5, bbox=None, position=<TextPosition.TOP_LEFT: 0>, padding=10*)

Object that represents a text.

serialize()

Serialize the object to dict.

class depthai_sdk.visualize.objects.**VisLine**(*pt1, pt2, color=None, thickness=None*)

Object that represents a line.

serialize()

Serialize the object to dict.

prepare()

Prepare necessary data for drawing.

Returns self

Return type *depthai_sdk.visualize.objects.VisLine*

class depthai_sdk.visualize.objects.**VisTrail**(*tracklets, label_map, bbox*)

Object that represents a trail.

serialize()

Serialize the object to dict.

prepare()

Prepare necessary data for drawing.

Returns self

Return type *depthai_sdk.visualize.objects.VisTrail*

groupby_tracklet()

Group tracklets by tracklet id.

Returns Dictionary of tracklets grouped by tracklet id.

static get_rect_centroid(*rect*, *w*, *h*)
Get centroid of a rectangle.

Parameters **rect** (*depthai.Rect*) –

Return type Tuple[int, int]

class depthai_sdk.visualize.configs.**TextPosition**(*value*)
Where on frame do we want to print text.

TOP_LEFT = 0

MID_LEFT = 1

BOTTOM_LEFT = 2

TOP_MID = 10

MID = 11

BOTTOM_MID = 12

TOP_RIGHT = 20

MID_RIGHT = 21

BOTTOM_RIGHT = 22

class depthai_sdk.visualize.configs.**BboxStyle**(*value*)
How do we want to draw bounding box.

RECTANGLE = 0

CORNERS = 1

ROUNDED_RECTANGLE = 10

ROUNDED_CORNERS = 11

class depthai_sdk.visualize.configs.**StereoColor**(*value*)
An enumeration.

GRAY = 1

RGB = 2

RGBD = 3

class depthai_sdk.visualize.configs.**OutputConfig**(*img_scale=1.0*, *show_fps=False*, *clickable=True*)
Configuration for output of the visualizer.

img_scale: **float = 1.0**

show_fps: **bool = False**

clickable: **bool = True**

class depthai_sdk.visualize.configs.**StereoConfig**(*colorize: depthai_sdk.visualize.configs.StereoColor = <StereoColor.RGB: 2>*, *colormap: numpy.ndarray = <factory>*, *wls_filter: bool = False*, *wls_lambda: float = 8000*, *wls_sigma: float = 1.5*)

colorize: ***depthai_sdk.visualize.configs.StereoColor*** **= 2**

colormap: **numpy.ndarray**

wls_filter: bool = False

wls_lambda: float = 8000

wls_sigma: float = 1.5

class depthai_sdk.visualize.configs.**DetectionConfig**(*thickness=1, fill_transparency=0.15, box_roundness=0, color=(0, 255, 0), bbox_style=<BboxStyle.RECTANGLE: 0>, line_width=0.5, line_height=0.5, hide_label=False, label_position=<TextPosition.TOP_LEFT: 0>, label_padding=10*)

Configuration for drawing bounding boxes.

thickness: int = 1

fill_transparency: float = 0.15

box_roundness: int = 0

color: Tuple[int, int, int] = (0, 255, 0)

bbox_style: *depthai_sdk.visualize.configs.BboxStyle* = 0

line_width: float = 0.5

line_height: float = 0.5

hide_label: bool = False

label_position: *depthai_sdk.visualize.configs.TextPosition* = 0

label_padding: int = 10

class depthai_sdk.visualize.configs.**TextConfig**(*font_face=0, font_color=(255, 255, 255), font_transparency=0.5, font_scale=1.0, font_thickness=2, font_position=<TextPosition.TOP_LEFT: 0>, background_color=None, background_transparency=0.5, outline_color=(0, 0, 0), line_type=16, auto_scale=True*)

Configuration for drawing labels.

font_face: int = 0

font_color: Tuple[int, int, int] = (255, 255, 255)

font_transparency: float = 0.5

font_scale: float = 1.0

font_thickness: int = 2

font_position: *depthai_sdk.visualize.configs.TextPosition* = 0

background_color: Optional[Tuple[int, int, int]] = None

background_transparency: float = 0.5

outline_color: Tuple[int, int, int] = (0, 0, 0)

line_type: int = 16

auto_scale: bool = True

class depthai_sdk.visualize.configs.**TrackingConfig**(*max_length=500, deletion_lost_threshold=5, line_thickness=1, fading_tails=False, line_color=(255, 255, 255), line_type=16, show_speed=False*)

Configuration for drawing tracking bounding boxes.

max_length: int = 500

deletion_lost_threshold: int = 5

line_thickness: int = 1

fading_tails: bool = False

line_color: Tuple[int, int, int] = (255, 255, 255)

line_type: int = 16

show_speed: bool = False

class depthai_sdk.visualize.configs.**SegmentationConfig**(*mask_alpha=0.5*)

Configuration for drawing segmentation masks.

mask_alpha: float = 0.5

class depthai_sdk.visualize.configs.**CircleConfig**(*thickness=1, color=(255, 255, 255), line_type=16*)

Configuration for drawing circles.

thickness: int = 1

color: Tuple[int, int, int] = (255, 255, 255)

line_type: int = 16

class depthai_sdk.visualize.configs.**VisConfig**(*output=<factory>, stereo=<factory>, detection=<factory>, text=<factory>, tracking=<factory>, circle=<factory>*)

Configuration for visualizer.

output: *depthai_sdk.visualize.configs.OutputConfig*

stereo: *depthai_sdk.visualize.configs.StereoConfig*

detection: *depthai_sdk.visualize.configs.DetectionConfig*

text: *depthai_sdk.visualize.configs.TextConfig*

tracking: *depthai_sdk.visualize.configs.TrackingConfig*

circle: *depthai_sdk.visualize.configs.CircleConfig*

CHAPTER

FIVE

AI MODELS

Through the *NNComponent*, DepthAI SDK abstracts:

1. **AI model sourcing** using blobconverter from Open Model Zoo (OMZ) and DepthAI Model Zoo (DMZ).
2. **AI result decoding** - currently SDK supports on-device decoding for YOLO and MobileNet based results using YoloDetectionNetwork and MobileNetDetectionNetwork nodes.
3. **Decoding** of the `config.json` which **allows an easy deployment of custom AI models** trained using our notebooks and converted using https://tools.luxonis.com.
4. Formatting of the AI model input frame - SDK uses **BGR** color order and **Planar / CHW** (Channel, Height, Width) layout conventions. If model accepts color images, it should accept 3 channels (B,G,R), and if it accepts grayscale images, it should accept 1 channel.

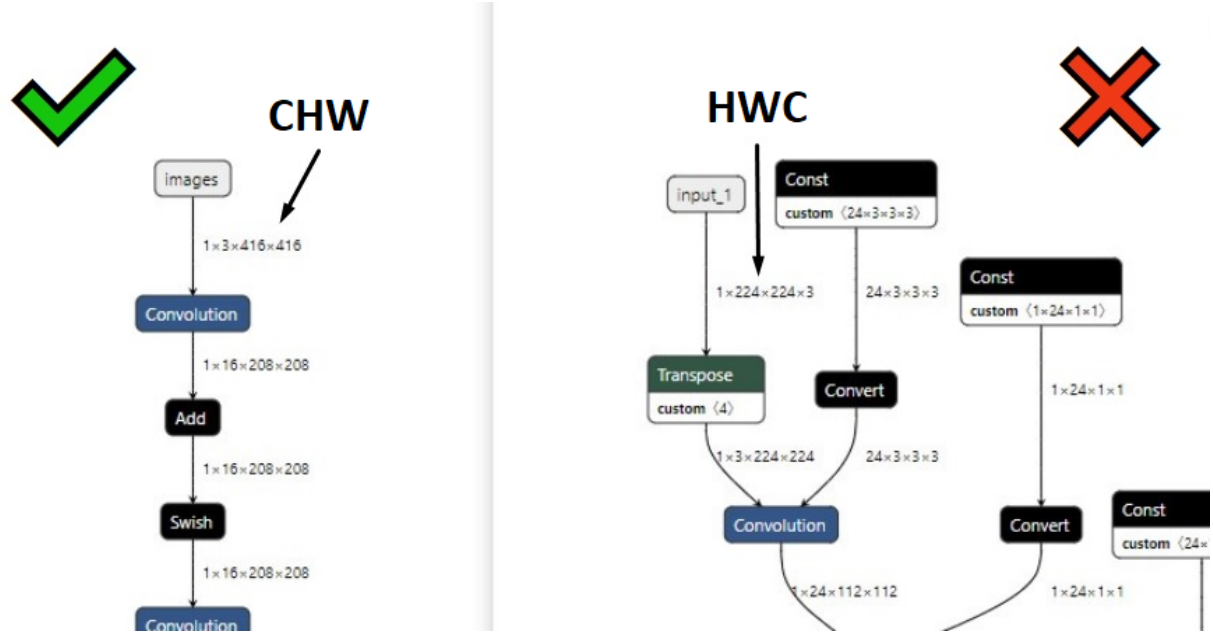


Fig. 1: Netron app allows you to check model's input layout

5. Integration with 3rd party tools/services (Roboflow).

5.1 SDK supported models

With *NNComponent* you can **easily try out a variety of different pre-trained models** by simply changing the model name:

```
  from depthai_sdk import OakCamera

  with OakCamera() as oak:
      color = oak.create_camera('color')
-     nn = oak.create_nn('mobilenet-ssd', color)
+     nn = oak.create_nn('vehicle-detection-0202', color)
      oak.visualize([nn], fps=True)
      oak.start(blocking=True)
```

Both of the models above are supported by this SDK, so they will be downloaded and deployed to the OAK device along with the pipeline.

The following table lists all the models supported by the SDK. The model name is the same as the name used in the *NNComponent* constructor.

Name	Model Source	FPS*
age-gender-recognition-retail-0013	OMZ	33
emotions-recognition-retail-0003	OMZ	33
face-detection-adas-0001	OMZ	18
face-detection-retail-0004	OMZ	33
facemesh_192x192	DMZ	32
facial_landmarks_68_160x160	32	DMZ
human-pose-estimation-0001	OMZ	8
mobilenet-ssd	OMZ	31
mobilenetv2_imagenet_embedder_224x224	DMZ	/
pedestrian-detection-adas-0002	OMZ	19
person-detection-0200	OMZ	14
person-detection-retail-0013	OMZ	15
person-reidentification-retail-0288	OMZ	33
person-vehicle-bike-detection-crossroad-1016	OMZ	12
sbd_mask_classification_224x224	DMZ	64+
vehicle-detection-0202	OMZ	14
vehicle-detection-adas-0002	OMZ	14
vehicle-license-plate-detection-barrier-0106	OMZ	29
yolo-v3-tf	OMZ	3.5
yolo-v3-tiny-tf	OMZ	33
yolov4_coco_608x608	DMZ	1.1
yolov4_tiny_coco_416x416	DMZ	32
yolov5n_coco_416x416	DMZ	32
yolov6n_coco_640x640	DMZ	26
yolov6nr3_coco_640x352	DMZ	32
yolov7tiny_coco_640x352	DMZ	23
yolov7tiny_coco_416x416	DMZ	29
yolov8n_coco_640x352	DMZ	22

⋆ - FPS was measured using only color camera (1080P) and 1 NN using callbacks (without visualization)

CHAPTER

SIX

AUTOMATIC IR POWER CONTROL

Note: This feature is only available on OAK devices with IR lights.

Note: This feature is experimental, please report any issues you encounter to the Luxonis team.

Automatic IR power control is a feature that allows the device to automatically adjust the IR power based on the scene. This is useful for applications where the scene is not always the same, for example when the camera is used in an outdoor environment.

To enable automatic IR power control, you need to use `auto_ir` method that accepts two parameters:

- `auto_mode` - `True` to enable automatic IR power control, `False` to disable it.
- `continuous_mode` - `True` to enable continuous mode, `False` otherwise. Requires `auto_mode` to be enabled.

When **automatic mode** is enabled, the device will automatically adjust the IR power after the startup. The disparity map will be analyzed with different dot projector and illumination settings, and once the best settings are found, the device will use them for the rest of the session. The whole process takes around **25 seconds**.

If **continuous mode** is enabled, the device will continue to search for better settings. In case the scene changes and disparity map quality drops below a certain threshold, the device will automatically adjust the IR power again.

6.1 Usage

The following example shows how to enable automatic IR power control in continuous mode:

```
from depthai_sdk import OakCamera

with OakCamera() as oak:
    left = oak.create_camera('left')
    right = oak.create_camera('right')
    stereo = oak.create_stereo(left=left, right=right)

    # Automatically estimate IR brightness and adjust it continuously
    stereo.set_auto_ir(auto_mode=True, continuous_mode=True)

    oak.visualize([stereo.out.disparity, left])
    oak.start(blocking=True)
```

CHAPTER

SEVEN

CONDITIONAL ACTIONS

DepthAI SDK provides a way to perform actions based on some conditions. For example, you can perform an action when a certain number of objects is detected in the frame. This functionality can be achieved by using Trigger-Action API.

7.1 Overview

Trigger-Action API is a way to define a set of conditions and actions that should be performed when these conditions are met. DepthAI SDK provides a set of predefined conditions and actions, but you can also define your own.

Basic concepts:

- **Trigger** - a condition that should be met to perform an action.
- **Action** - an action that should be performed when a trigger is met.

Note: Trigger-Action API is implemented in the `depthai.trigger_action` module.

7.2 Triggers

The base class for all triggers is *`Trigger`*. In order to create a trigger, you need to use the *`Trigger`* class and pass the following parameters:

- `input` - a component that should be used as a trigger source.
- `condition` - a function that should return `True` or `False` based on the trigger source.
- `cooldown` - defines how often a trigger can be activated (in seconds).

The set of predefined triggers:

- *`DetectionTrigger`* - a trigger that is activated when a certain number of objects is detected in the frame.

7.3 Actions

An action can be represented by either a function or a class derived from `Action` class. The custom action should implement `activate()` and optionally `on_new_packets()` methods.

The set of predefined actions:

- `RecordAction` - records a video of a given duration when a trigger is activated.

7.4 Usage

The following example shows how to create a trigger that is activated when at least 1 person is detected in the frame. When the trigger is activated, it records a 15 seconds video (5 seconds before the trigger is activated and 10 seconds after).

```
from depthai_sdk import OakCamera
from depthai_sdk.trigger_action.actions.record_action import RecordAction
from depthai_sdk.trigger_action.triggers.detection_trigger import DetectionTrigger

with OakCamera() as oak:
    color = oak.create_camera('color', encode='jpeg')
    stereo = oak.create_stereo('400p')

    nn = oak.create_nn('mobilenet-ssd', color)

    trigger = DetectionTrigger(input=nn, min_detections={'person': 1}, cooldown=30)
    action = RecordAction(inputs=[color, stereo.out.disparity], dir_path='./
↪recordings/',
                          duration_before_trigger=5, duration_after_trigger=10)
    oak.trigger_action(trigger=trigger, action=action)

    oak.visualize(nn)
    oak.start(blocking=True)
```

7.5 Reference

class depthai_sdk.trigger_action.**TriggerAction**(*trigger*, *action*)

TriggerAction class represents a single trigger-action pair.

new_packet_trigger(*packet*)

This method is called when a new packet is received from the trigger input stream.

Parameters **packet** (depthai_sdk.classes.packets.FramePacket) – Packet received from the input stream.

Return type None

new_packet_action(*packet*)

This method is called when a new packet is received from the action input streams. Primary purpose of this method is to provide a way to keep a track of the packets.

Parameters **packet** (depthai_sdk.classes.packets.FramePacket) – Packet received from the input stream.

Return type None

class depthai_sdk.trigger_action.**Trigger**(*input*, *condition*, *cooldown*)

Base trigger represents a single trigger that can activate an action.

class depthai_sdk.trigger_action.**Action**(*inputs=None*, *action=None*)

Base action represents a single action that can be activated by a trigger.

activate()

Method that gets called when the action is activated by a trigger.

Return type None

on_new_packets(*packets*)

Callback method that gets called when all packets are synced.

Parameters **packets** (*Dict[str, depthai_sdk.classes.packets.FramePacket]*) – Dictionary of packets received from the input streams.

Return type None

class depthai_sdk.trigger_action.**DetectionTrigger**(*input*, *min_detections*, *cooldown*)

Trigger that is activated when a certain number of detections of a certain label are detected.

condition(*packet*)

Method that checks if the trigger should be activated.

Parameters **packet** (depthai_sdk.classes.packets.DetectionPacket) – DetectionPacket that contains the detections.

Returns True if the trigger should be activated, False otherwise.

Return type bool

class depthai_sdk.trigger_action.**RecordAction**(*inputs*, *dir_path*, *duration_before_trigger*, *duration_after_trigger*, *on_finish_callback=None*)

Action that records video from specified inputs for a specified duration before and after trigger event.

activate()

Method that gets called when the action is activated by a trigger.

on_new_packets(*packets*)

Callback method that gets called when all packets are synced.

Parameters **packets** (*Dict[str, depthai_sdk.classes.packets.FramePacket]*) – Dictionary of packets received from the input streams.

setup(*device*, *xouts*)

Parameters

- **device** (*depthai.Device*) –
- **xouts** (*List[XoutFrames]*) –

CHAPTER

EIGHT

RECORDING

OakCamera allows users to easily **record** video streams so the scene can later be fully replayed (see *Replaying* documentation), including **reconstructing the stereo depth perception**.

The script below will save color, left, and right H265 encoded streams into video files. Frames are synchronized (via timestamps) before being saved.

```
from depthai_sdk import OakCamera, RecordType

with OakCamera() as oak:
    color = oak.create_camera('color', resolution='1080P', fps=20, encode='H265')
    left = oak.create_camera('left', resolution='800p', fps=20, encode='H265')
    right = oak.create_camera('right', resolution='800p', fps=20, encode='H265')

    # Synchronize & save all (encoded) streams
    oak.record([color.out.encoded, left.out.encoded, right.out.encoded], './',␣
↪RecordType.VIDEO)
    # Show color stream
    oak.visualize([color.out.camera], scale=2/3, fps=True)

    oak.start(blocking=True)
```

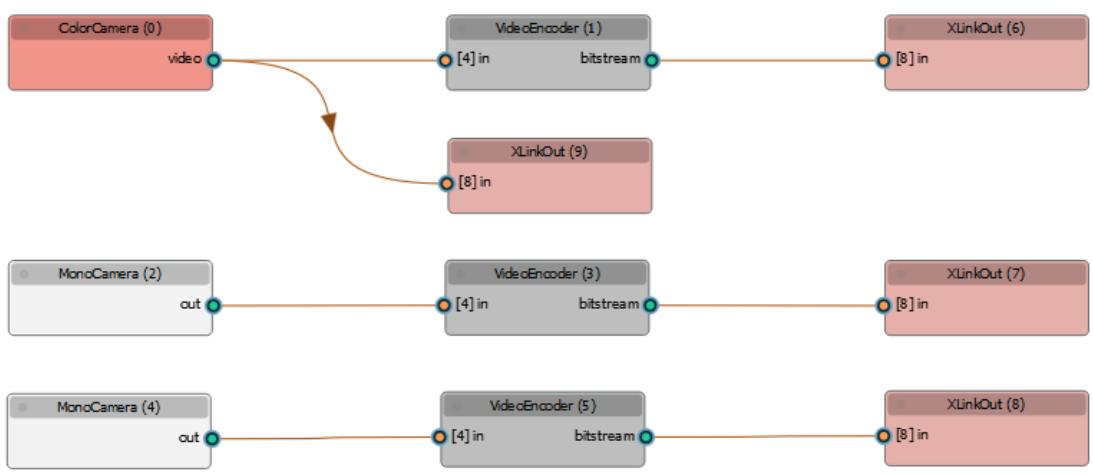


Fig. 1: Recording pipeline of the script above

After 20 seconds we stopped the recording and SDK printed the location of saved files (`./1-18443010D116631200` in our case):

```
Mode                LastWriteTime         Length Name
----                -------------         ------ ----
-a----         10/3/2022  12:50 PM           9281 calib.json
-a----         10/3/2022  12:51 PM       19172908 color.mp4
-a----         10/3/2022  12:51 PM       15137490 left.mp4
-a----         10/3/2022  12:51 PM       15030761 right.mp4
```

This depthai-recording can then be used next time to reconstruct the whole scene using the *Replaying* feature.

8.1 Supported recording types

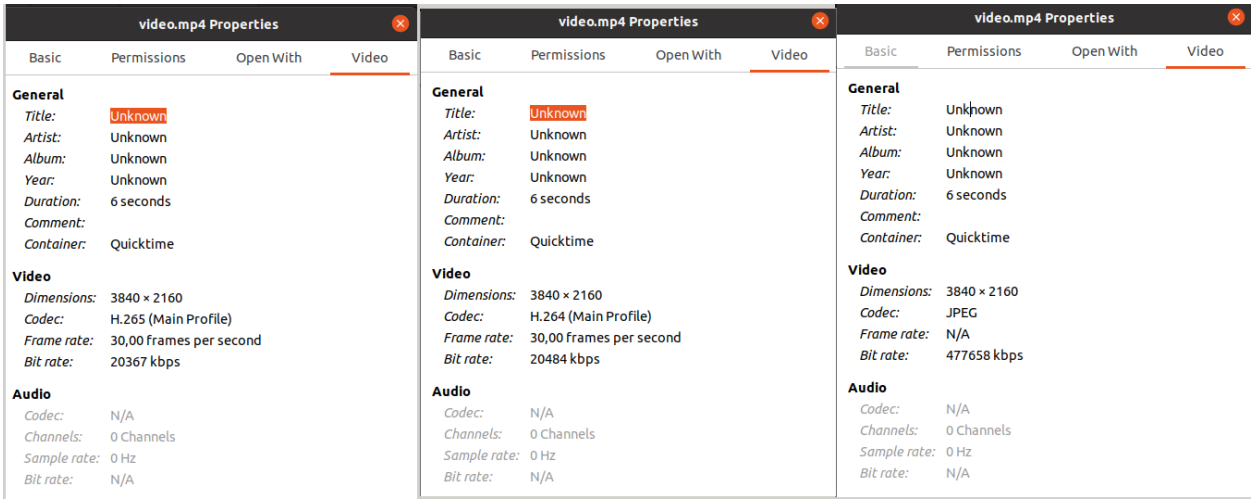
1. *RecordType.VIDEO*
2. *RecordType.BAG*
3. *RecordType.MCAP*

8.1.1 1. Video

This option will write each stream separately to a video file. There are three options for saving these files:

1. If we are saving unencoded frames SDK will use `cv2.VideoWriter` class to save these streams into `.avi` file.
2. If we are saving encoded streams and we have `av` installed (PyAv library) SDK will save encoded streams directly to `.mp4` container. This will allow you to watch videos with a standard video player. There's **no decoding/encoding** (or converting) happening on the host computer and host **CPU/GPU/RAM usage is minimal**. More information here.
3. Otherwise SDK will save encoded streams to files (eg. `color.mjpeg`) and you can use ffmpeg or mkvmerge to containerize the stream so it's viewable by most video players. More information here.

200 frames from 4K color camera using different encoding options (MJPEG, H.264, H.265) using `av`:



8.1.2 2. Rosbag

Currently, we only support recording `depth` to the rosbag (`recording.bag`). In the future we will also support color (which is depth aligned) stream and mono streams. You can open the rosbag with the RealSense Viewer to view the depth:

8.1.3 3. MCAP recording

An alternative to Rosbags are mcap files which can be viewed with Foxglove studio. You can find MCAP recording example here. Currently supported streams:

- MJPEG encoded color/left/right/disparity. Lossless MJPEG/H264/H265 aren't supported by Foxglove Studio.
- Non-encoded color/left/right/disparity/depth frames.
- Pointcloud, enable with `recorder.config_mcap(pointcloud=True)`. It converts depth frame to pointcloud on the host.

Standalone Foxglove studio streaming demo can be found here.

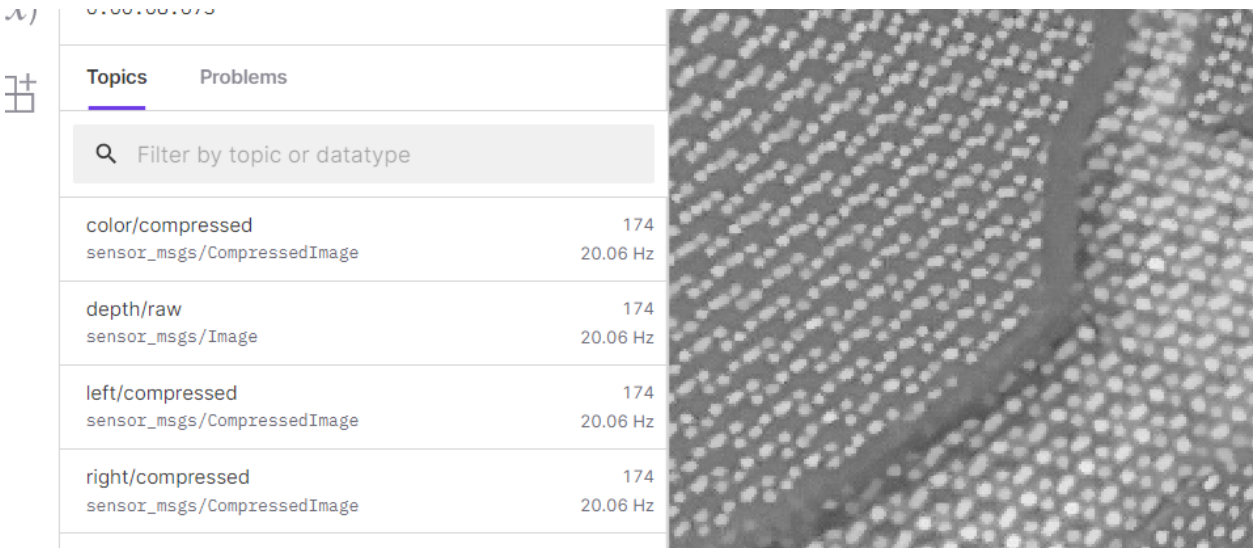


Fig. 2: Available topics in Foxglove Studio from MCAP recorded by mcap-recording.py example

CHAPTER

NINE

REPLAYING

OakCamera allows users to easily use **depthai-recording** instead of the live camera feed to run their pipelines. This feature will send recorded frames to the OAK device. This is useful especially during development, so we can **record a complex scene** only once and replay it many times to fine-tune the pipeline or AI models.

Because *Recording* saves calibration data and can save synchronized left+right mono streams so we can achieve full depth reconstruction.

```
from depthai_sdk import OakCamera

with OakCamera(recording='[PATH/URL/NAME]') as oak:
    # Created CameraComponent/StereoComponent will use streams from the recording
    camera = oak.create_camera('color')
```

9.1 Replaying support

Replaying feature is quite extensible, and supports a variety of different inputs:

1. Single image.
2. Folder with images. Images are getting rotated every 3 seconds. Example here.
3. URL to a video/image.
4. URL to a YouTube video.
5. Path to *depthai-recording*.
6. A name of a *public depthai-recording*.

9.2 Replaying a depthai-recording

When constructing the OakCamera object we can easily replay an existing *depthai-recording*, which results in using XLinkIn nodes instead of ColorCamera / MonoCamera nodes.

Script below will also do depth reconstruction and will display 3D detections coordinates (XYZ) to the frame.

```
  from depthai_sdk import OakCamera

- with OakCamera() as oak:
+ with OakCamera(replay='path/to/folders') as oak:
      color = oak.create_camera('color')
      nn = oak.create_nn('mobilenet-ssd', color, spatial=True)
```

(continues on next page)

(continued from previous page)

```
    oak.visualize(nn.out.main, fps=True)
    oak.start(blocking=True)
```

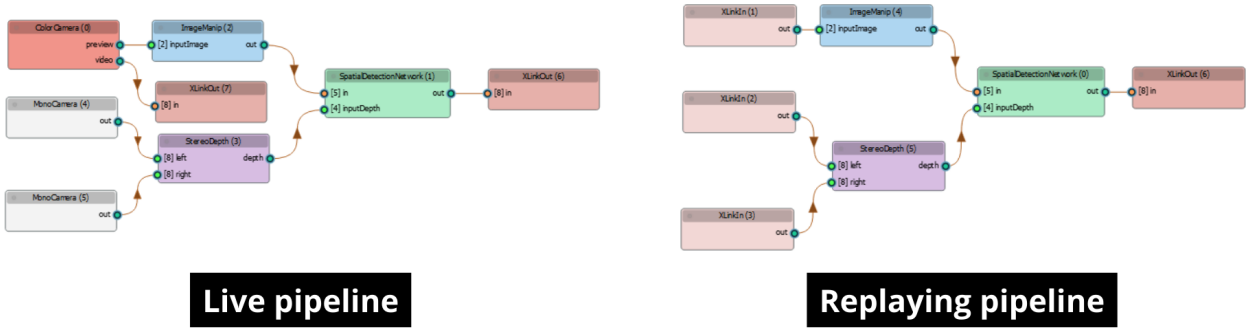


Fig. 1: Live view pipeline uses live camera feeds (MonoCamera, ColorCamera) whereas Replaying pipeline uses XLinkIn nodes to which we send recorded frames.

9.3 Public depthai-recordings

We host several depthai-recordings on our servers that you can easily use in your application, e.g., `OakCamera(recording='cars-california-01')`. Recording will get downloaded & cached on the computer for future use.

The following table lists all available recordings:

Name	Files	Size	Notice
`cars-california-01`	`color.mp4`	21.1 MB	Source video, useful for car detection / license plate recognition
`cars-california-02`	`color.mp4`	27.5 MB	Source video, useful for car detection / license plate recognition
`cars-california-03`	`color.mp4`	19 MB	Source video, useful for license plate recognition and bicylist detection
`cars-tracking-above-01`	`color.mp4`	30.8 MB	Source video, useful for car tracking/counting
`depth-people-counting-01`	`left.mp4`, `right.mp4`, `calib.json`	5.8 MB	Used by depth-people-counting demo
`people-construction-01`	`color.mp4`	5.2 MB	Used by ObjectTracker example and pedestrian reidentification demo
`people-images-01`	5x jpg images	2 MB	Used by people-counting demo
`people-tracking-above-01`	`color.mp4`	3.2 MB	Fisheye top-down view, useful for people tracking/counting. Fast forward/downscaled
`people-tracking-above-02`	`color.mp4`	86.4 MB	Fisheye top-down view, useful for people tracking/counting
`people-tracking-above-03`	`color.mp4`	16.7 MB	Top-down view, used by people-tracker demo
`people-tracking-above-04`	`color.mp4`	5.3 MB	Top-down view at an angle, source video here
`people-tracking-above-05`	`CAM_A.mp4`, `CAM_A.mp4`, `calib.json`	12 MB (35sec)	Top-down view, left+right stereo cameras, demo usage at replay.py

CHAPTER

TEN

CODE SAMPLES

10.1 FFC Camera Visualization

This example shows how to use the *Camera* component to display the camera feed from the FFC camera.

For FFC, the camera board socket must be specified. In our case the cameras are connected to socket A, B and C. After setting the resolution to 1200p and downscaling using ISP to 800p, the camera feed is displayed in a window.

Note: Visualization in current example is done with blocking behavor. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

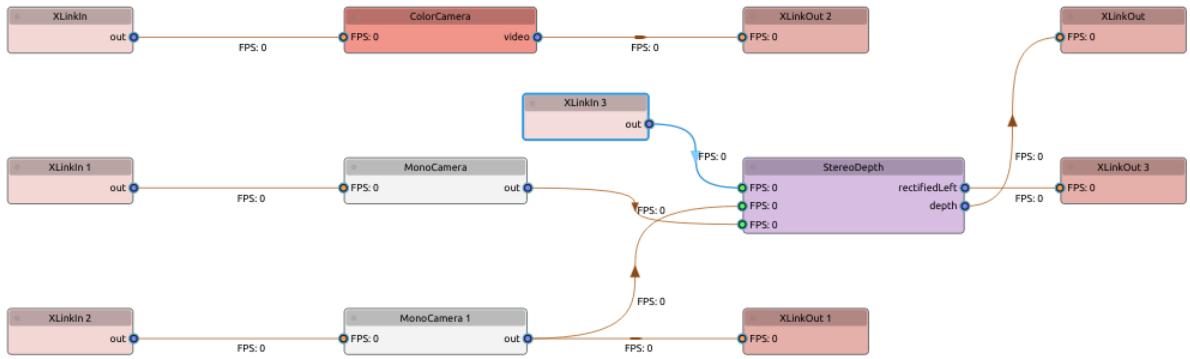
10.1.1 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.1.2 Pipeline



10.1.3 Source Code

Python

Also available on GitHub

```
from depthai_sdk import OakCamera

with OakCamera() as oak:
    cama = oak.create_camera('cama,c', resolution='1200p')
    cama.config_color_camera(isp_scale=(2,3))
    camb = oak.create_camera('camb,c', resolution='1200p')
    camb.config_color_camera(isp_scale=(2,3))
    camc = oak.create_camera('camc,c', resolution='1200p')
    camc.config_color_camera(isp_scale=(2,3))

    stereo = oak.create_stereo(left=camb, right=camc)
    stereo.config_undistortion(M2_offset=0)

    oak.visualize([stereo, camc, cama, stereo.out.rectified_left], fps=True)

    oak.start(blocking=True)
```

10.2 Camera Control

This example shows how to use DepthAI SDK to control the color camera parameters.

```
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
```

10.2.1 Demo

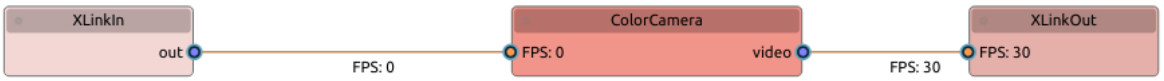
10.2.2 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.2.3 Pipeline



10.2.4 Source Code

Python

Also available on GitHub

```
from depthai_sdk import OakCamera

with OakCamera() as oak:
    color = oak.create_camera('color')
    oak.visualize(color, fps=True, scale=2/3)
    oak.start()

    while oak.running():
        key = oak.poll()
        if key == ord('i'):
            color.control.exposure_time_down()
        elif key == ord('o'):
            color.control.exposure_time_up()
        elif key == ord('k'):
            color.control.sensitivity_down()
        elif key == ord('l'):
            color.control.sensitivity_up()

        elif key == ord('e'): # Switch to auto exposure
            color.control.send_controls({'exposure': {'auto': True}})

```

10.3 Camera Control with NN

This example shows how to set up control of color camera (focus and exposure) to be controlled by NN. The NN is a face detection model which passes detected face bounding box to camera component run auto focus and auto exposure algorithms on.

Note: Visualization in current example is done with blocking behavor. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

10.3.1 Demo

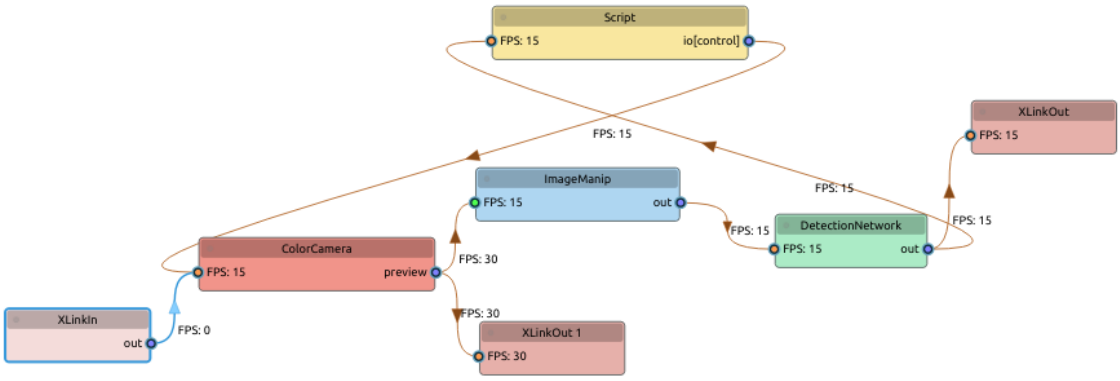
10.3.2 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.3.3 Pipeline



10.3.4 Source Code

Python

Also available on GitHub

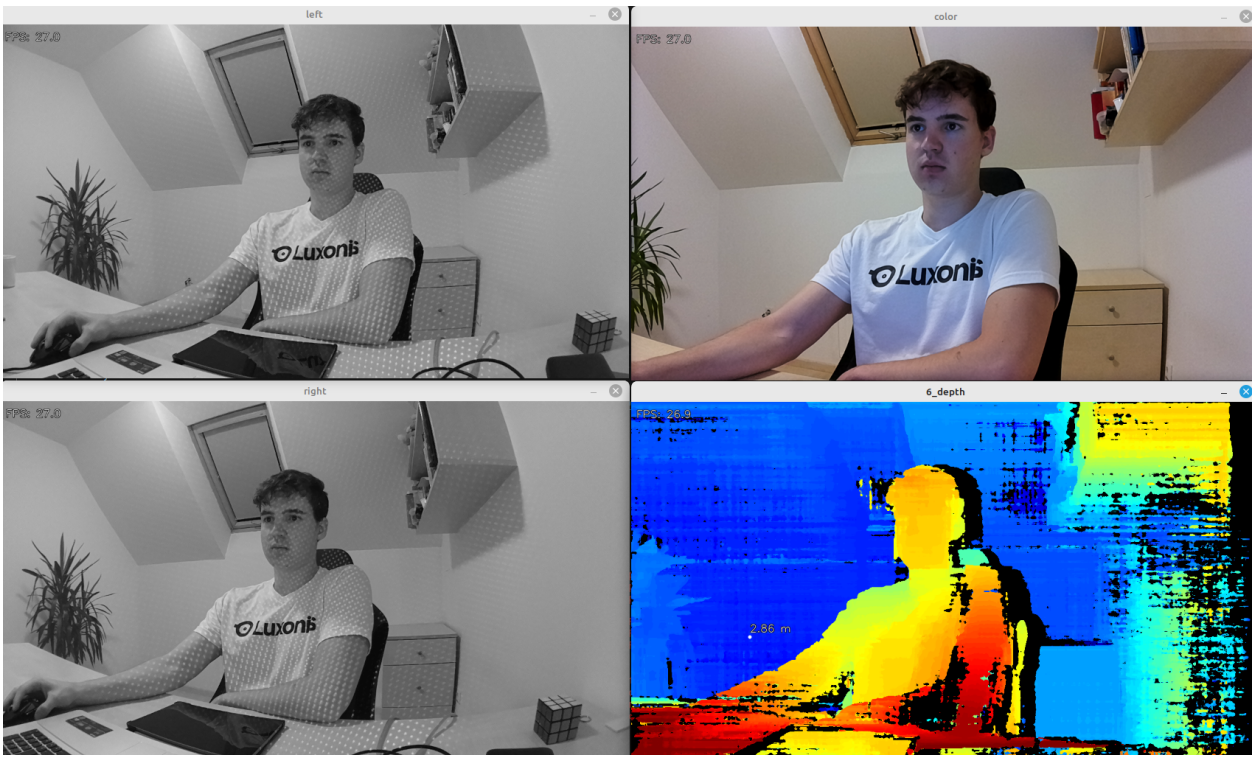
```
from depthai_sdk import OakCamera

with OakCamera() as oak:
    color = oak.create_camera('color')
    face_det = oak.create_nn('face-detection-retail-0004', color)
    # Control the camera's exposure/focus based on the (largest) detected face
    color.control_with_nn(face_det, auto_focus=True, auto_exposure=True, debug=False)

    oak.visualize(face_det, fps=True)
    oak.start(blocking=True)
```

10.4 Camera Preview

This example shows how to set up a pipeline that outputs a a preview for color camera, both mono cameras and their stereo depth. Each frame is displayed using OpenCV in blocking behavour.

Note: Visualization in current example is done with blocking behavor. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

10.4.1 Demo



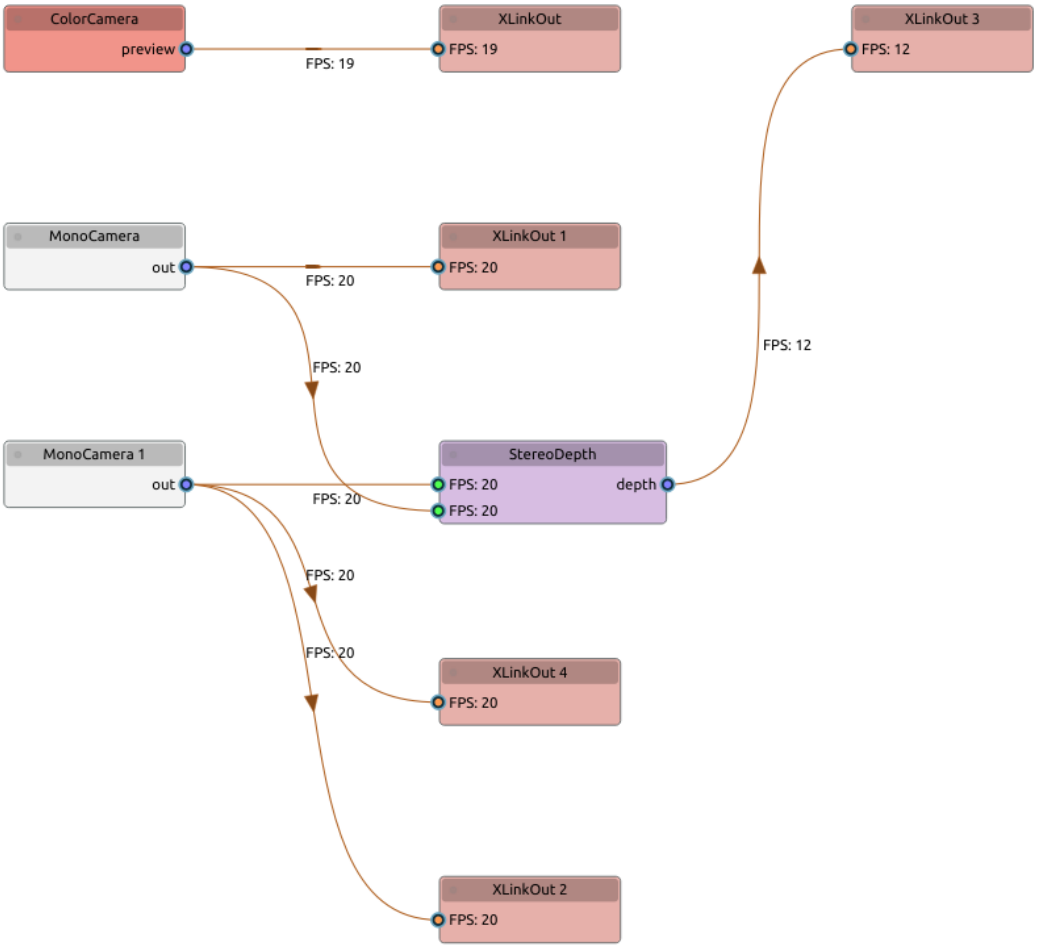
10.4.2 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.4.3 Pipeline



10.4.4 Source Code

Python

Also available on GitHub

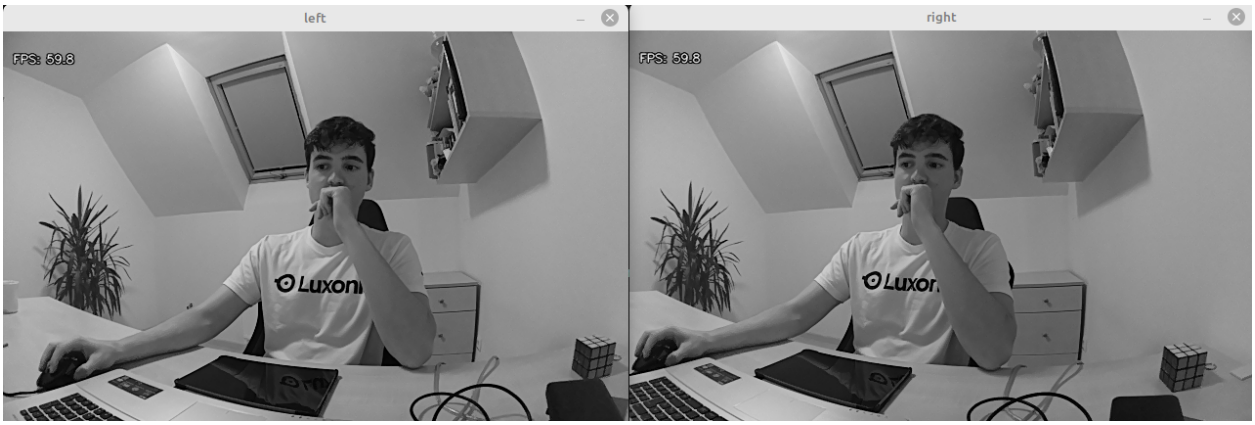
```
from depthai_sdk import OakCamera

with OakCamera() as oak:
    color = oak.create_camera('color')
    left = oak.create_camera('left')
    right = oak.create_camera('right')
    stereo = oak.create_stereo(left=left, right=right)

    oak.visualize([color, left, right, stereo.out.depth], fps=True, scale=2/3)
    oak.start(blocking=True)
```

10.5 Mono Camera Preview

This example shows how to set up a pipeline that outputs a video feed for both mono cameras and sets the resolution to 400p (640x400) and the frame rate to 60 fps.

Note: Visualization in current example is done with blocking behavor. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

10.5.1 Demo



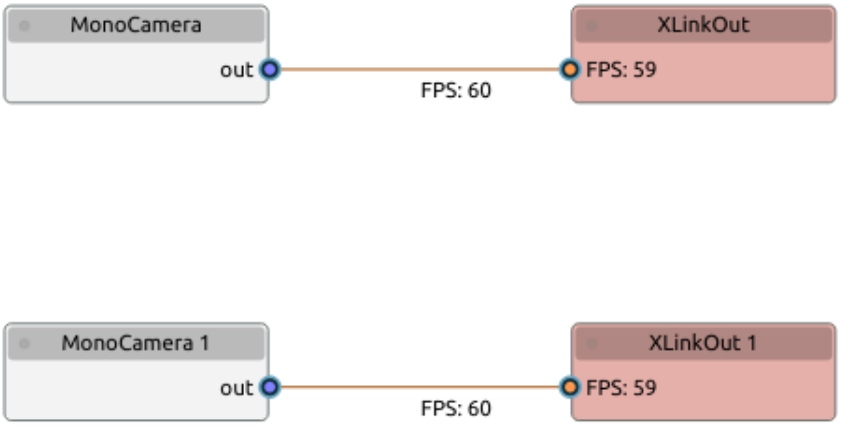
10.5.2 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.5.3 Pipeline



10.5.4 Source Code

Python

Also available on GitHub

```
from depthai_sdk import OakCamera

with OakCamera() as oak:
    left = oak.create_camera('left', resolution='400p', fps=60)
    right = oak.create_camera('right', resolution='400p', fps=60)
    oak.visualize([left, right], fps=True)
    oak.start(blocking=True)
```

10.6 Preview All Cameras

This example shows how to set up a pipeline that outputs a a preview for each camera currently connected (and available) to the device. The preview is displayed in a window on the host machine. If run on OAK-D devices, this example does the same thing as the `sdk_camera_preview` example.

Note: Visualization in current example is done with blocking behavor. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

10.6.1 Demo

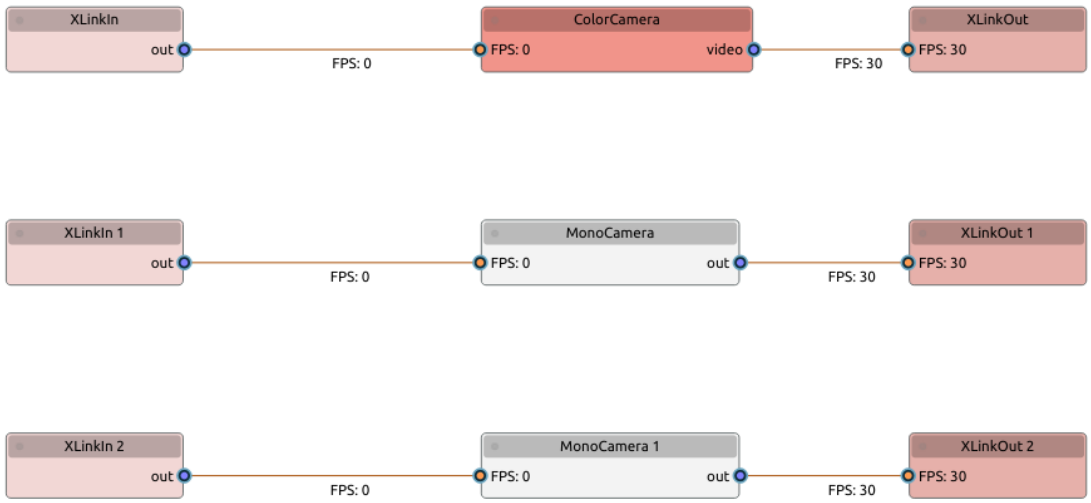
10.6.2 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.6.3 Pipeline



10.6.4 Source Code

Python

Also available on GitHub

```
from depthai_sdk import OakCamera

with OakCamera() as oak:
    cams = oak.create_all_cameras(resolution='max')
    oak.visualize(cams, fps=True)
    oak.start(blocking=True)
```

10.7 RGB and Mono Preview

This example shows how to use the *Camera* component to get RGB and Mono previews. It is similar to the ref:*sdk_camera_preview* example, but lacks the stereo depth visualization.

Note: Visualization in current example is done with blocking behavor. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

10.7.1 Demo

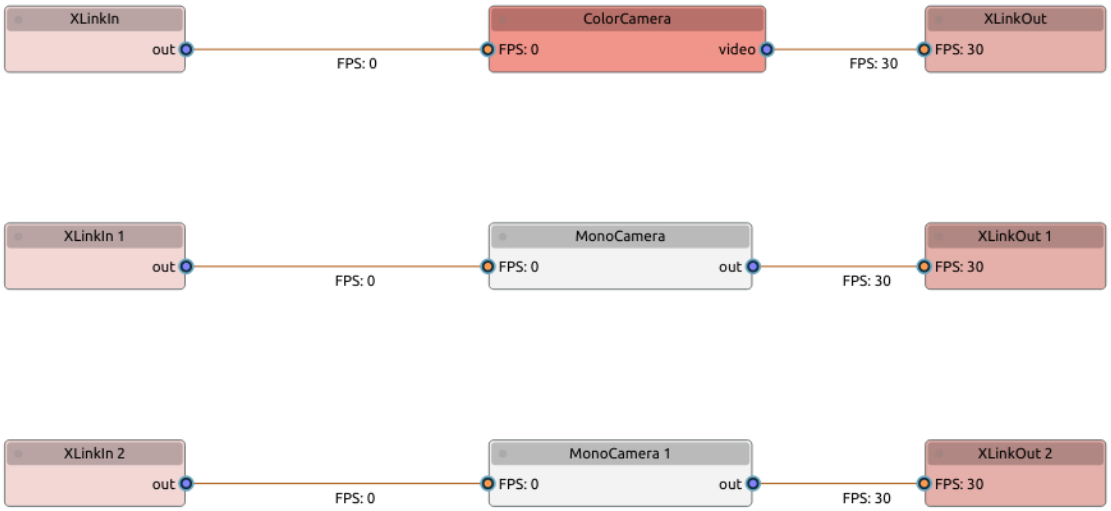
10.7.2 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.7.3 Pipeline



10.7.4 Source Code

Python

Also available on GitHub

```
from depthai_sdk import OakCamera

with OakCamera() as oak:
    color = oak.create_camera('color')
    left = oak.create_camera('left')
    right = oak.create_camera('right')
    oak.visualize([color, left, right], fps=True)
    oak.start(blocking=True)
```

10.8 Camera Rotated Preview

This example showcases how to rotate the preview frames by a desired angle (currently only 90, 180 and 270 degrees are supported).

Note: Visualization in current example is done with blocking behavior. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

10.8.1 Demo

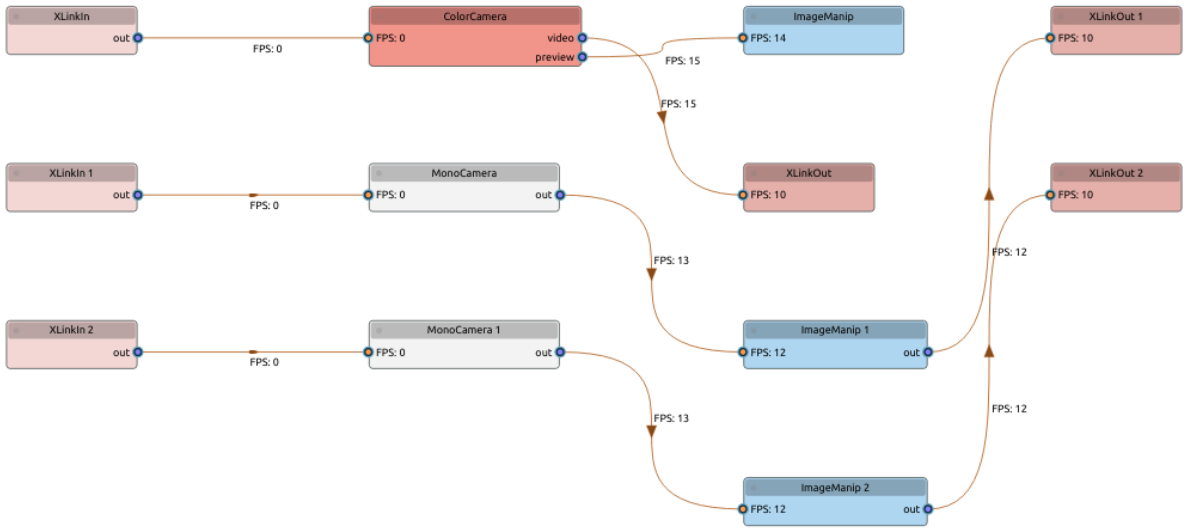
10.8.2 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.8.3 Pipeline



10.8.4 Source Code

Python

Also available on GitHub

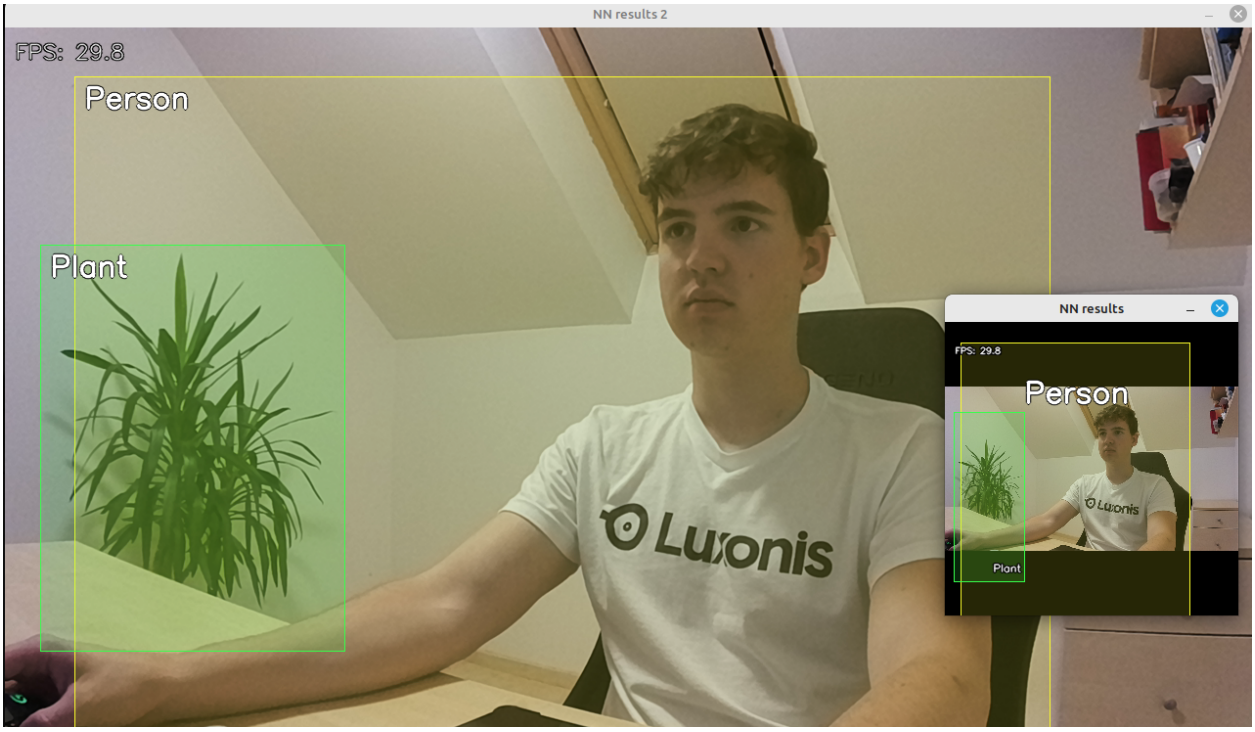
```python
from depthai_sdk import OakCamera

with OakCamera(rotation=90) as oak:
    all_cams = oak.create_all_cameras()
    oak.visualize(all_cams, fps=True)
    oak.start(blocking=True)
```

10.9 API Interoperability Example

This example shows how to bridge the DepthAI API with the SDK. It first creates the color camera and mobilenet neural network and displays the results. With *oak.build()* we build the pipeline which is part of the API. We can then manipulate the pipeline just like we would in the API (e.g. add Xlink connections, scripts, . . .). In this example we manually add a feature tracker since the SDK currently does not support it. We then start the pipeline and display the results.

Note that in this case, the visualizer behavior is non-blocking. This means we need to poll the visualizer in order to get the results.

10.9.1 Demo



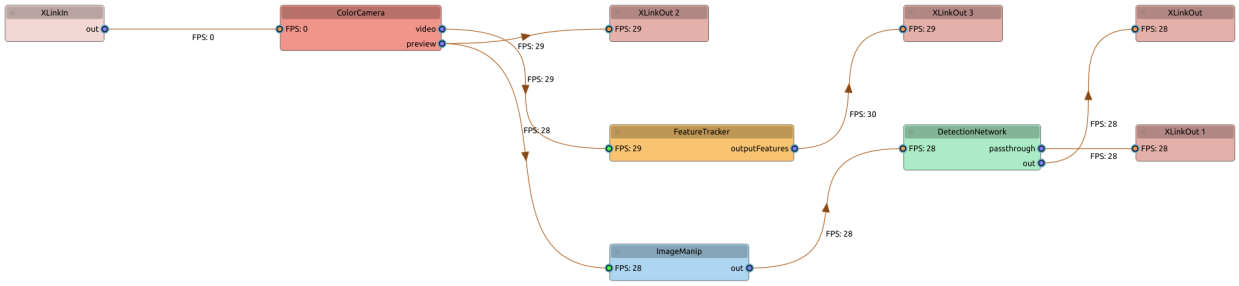
10.9.2 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.9.3 Pipeline



10.9.4 Source Code

Python

Also available on GitHub.

```python
from depthai_sdk import OakCamera
import depthai as dai

with OakCamera() as oak:
    color = oak.create_camera('color')
    nn = oak.create_nn('mobilenet-ssd', color)
    oak.visualize([nn.out.passthrough, nn], fps=True)

    nn.node.setNumInferenceThreads(2) # Configure components' nodes

    features = oak.pipeline.create(dai.node.FeatureTracker) # Create new pipeline␣
↪nodes
    color.node.video.link(features.inputImage)

    out = oak.pipeline.create(dai.node.XLinkOut)
    out.setStreamName('features')
    features.outputFeatures.link(out.input)

    oak.start() # Start the pipeline (upload it to the OAK)

    q = oak.device.getOutputQueue('features') # Create output queue after calling␣
↪start()
    while oak.running():
        if q.has():
            result = q.get()
            print(result)
        # Since we are not in blocking mode, we have to poll oak camera to
        # visualize frames, call callbacks, process keyboard keys, etc.
        oak.poll()
```

10.10 Car Tracking Example

This example shows how to use SDK to run inference on a pre-saved video file and display the results.

Note: Visualization in current example is done with blocking behavor. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

10.10.1 Demo

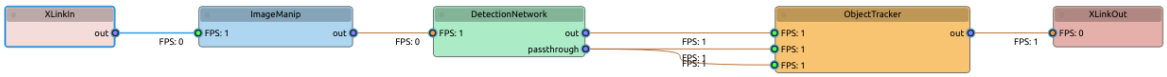
10.10.2 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.10.3 Pipeline



10.10.4 Source Code

Python

Also available on GitHub.

```python
from depthai_sdk import OakCamera, ResizeMode

# Download public depthai-recording
with OakCamera(replay='cars-tracking-above-01') as oak:
    # Create color camera, add video encoder
    color = oak.create_camera('color')

    # Download & run pretrained vehicle detection model and track detections
    nn = oak.create_nn('vehicle-detection-0202', color, tracker=True)

    # Visualize tracklets, show FPS
    visualizer = oak.visualize(nn.out.tracker, fps=True, record_path='./car_tracking.avi')
    visualizer.tracking(line_thickness=5).text(auto_scale=True)
    # Start the app in blocking mode
    # oak.show_graph()
    oak.start(blocking=True)
```

10.11 Collision Avoidance

This example shows how to set up a depth based collision avoidance system for proximity. This can be used with supervised robotic operation where the goal is to limit the robot's speed when a person is detected in front of it.

Note: Visualization in current example is done with blocking behavor. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

10.11.1 Demo

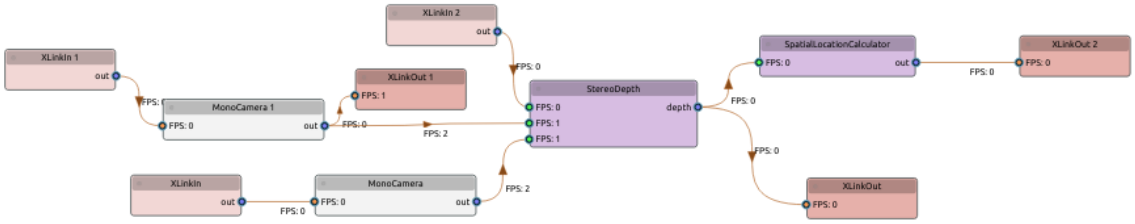
10.11.2 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.11.3 Pipeline



10.11.4 Source Code

Python

Also available on GitHub.

```python
from depthai_sdk import OakCamera
from depthai_sdk.visualize.configs import StereoColor
from depthai_sdk.classes.packets import DisparityDepthPacket
import math
import depthai as dai
```

(continues on next page)

(continued from previous page)

```
import cv2

# User-defined constants
WARNING = 1000 # 1m, orange
CRITICAL = 500 # 50cm, red

slc_data = []

def cb(packet: DisparityDepthPacket):
    global slc_data
    fontType = cv2.FONT_HERSHEY_TRIPLEX

    depthFrameColor = packet.visualizer.draw(packet.frame)

    for depthData in slc_data:
        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)

        coords = depthData.spatialCoordinates
        distance = math.sqrt(coords.x ** 2 + coords.y ** 2 + coords.z ** 2)

        if distance == 0: # Invalid
            continue

        if distance < CRITICAL:
            color = (0, 0, 255)
            cv2.rectangle(depthFrameColor, (xmin, ymin), (xmax, ymax), color, thickness=4)
            cv2.putText(depthFrameColor, "{:.1f}m".format(distance/1000), (xmin + 10, ymin + 20), fontType, 0.5, color)
        elif distance < WARNING:
            color = (0, 140, 255)
            cv2.rectangle(depthFrameColor, (xmin, ymin), (xmax, ymax), color, thickness=2)
            cv2.putText(depthFrameColor, "{:.1f}m".format(distance/1000), (xmin + 10, ymin + 20), fontType, 0.5, color)

    cv2.imshow('0_depth', depthFrameColor)

with OakCamera() as oak:
    stereo = oak.create_stereo('720p')
    # We don't need high fill rate, just very accurate depth, that's why we enable some filters, and
    # set the confidence threshold to 50
    config = stereo.node.initialConfig.get()
    config.postProcessing.brightnessFilter.minBrightness = 0
    config.postProcessing.brightnessFilter.maxBrightness = 255
    stereo.node.initialConfig.set(config)
    stereo.config_postprocessing(colorize=StereoColor.RGBD, colormap=cv2.COLORMAP_BONE)
    stereo.config_stereo(confidence=50, lr_check=True, extended=True)
```

(continues on next page)

(continued from previous page)

```

    oak.visualize([stereo], fps=True, callback=cb)

    slc = oak.pipeline.create(dai.node.SpatialLocationCalculator)
    for x in range(15):
        for y in range(9):
            config = dai.SpatialLocationCalculatorConfigData()
            config.depthThresholds.lowerThreshold = 200
            config.depthThresholds.upperThreshold = 10000
            config.roi = dai.Rect(dai.Point2f((x+0.5)*0.0625, (y+0.5)*0.1), dai.
↪Point2f((x+1.5)*0.0625, (y+1.5)*0.1))
            # TODO: change from median to 10th percentile once supported
            config.calculationAlgorithm = dai.SpatialLocationCalculatorAlgorithm.
↪MEDIAN
            slc.initialConfig.addROI(config)

    stereo.depth.link(slc.inputDepth)

    slc_out = oak.pipeline.create(dai.node.XLinkOut)
    slc_out.setStreamName('slc')
    slc.out.link(slc_out.input)

    oak.start() # Start the pipeline (upload it to the OAK)

    q = oak.device.getOutputQueue('slc') # Create output queue after calling start()
    while oak.running():
        if q.has():
            slc_data = q.get().getSpatialLocations()
        oak.poll()
```

10.12 Speed Calculation Preview

This example showcases the use of callback function inside the visualizer to log speed and draw tracking information.

Note: Visualization in current example is done with blocking behavior. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

10.12.1 Demo

10.12.2 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.12.3 Pipeline

_static/images/pipelines/speed_calculation.png

10.12.4 Source Code

Python

Also available on GitHub.

```python
import cv2

from depthai_sdk import OakCamera
from depthai_sdk.classes.packets import TrackerPacket


def callback(packet: TrackerPacket):
    for obj_id, tracklets in packet.tracklets.items():
        if len(tracklets) != 0:
            tracklet = tracklets[-1]
        if tracklet.speed is not None:
            print(f'Speed for object {obj_id}: {tracklet.speed:.02f} m/s, {tracklet.speed_kmph:.02f} km/h, {tracklet.speed_mph:.02f} mph')

    frame = packet.visualizer.draw(packet.decode())
    cv2.imshow('Speed estimation', frame)


with OakCamera() as oak:
    color = oak.create_camera('color')
    stereo = oak.create_stereo('800p')
    stereo.config_stereo(subpixel=False, lr_check=True)

```

(continues on next page)

(continued from previous page)

```
    nn = oak.create_nn('face-detection-retail-0004', color, spatial=stereo, tracker=True)
    nn.config_tracker(calculate_speed=True)

    visualizer = oak.visualize(nn.out.tracker, callback=callback, fps=True)
    visualizer.tracking(show_speed=True).text(auto_scale=True)

    oak.start(blocking=True)
```

10.13 Switch Between Models

This example shows how to switch between models on the fly. It uses script node to alter pipeline flow (either to use the yolo model or the mobilenet model).

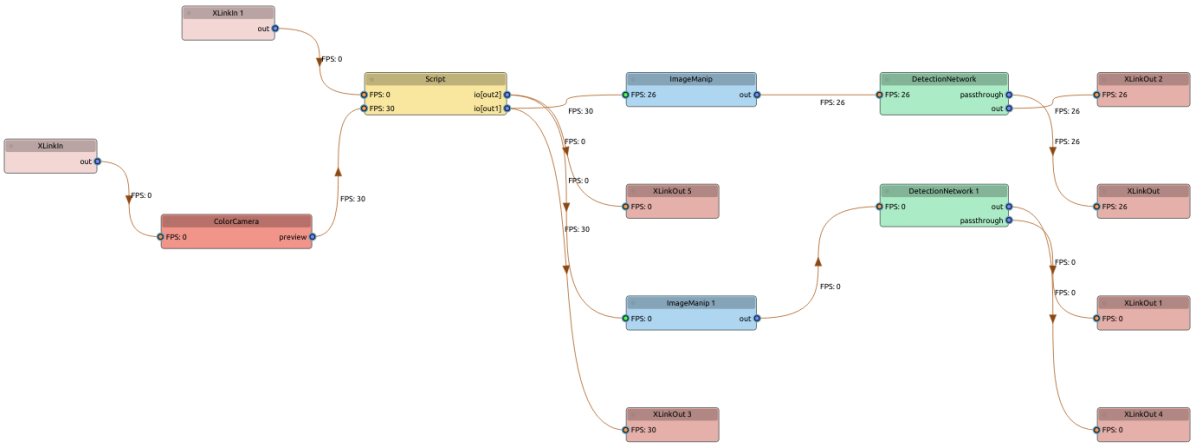
10.13.1 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.13.2 Pipeline



10.13.3 Source Code

Python

Also available on GitHub.

```python
from depthai_sdk import OakCamera
from depthai_sdk.classes.packets import DetectionPacket
import depthai as dai
import cv2

# We use callback, so we only have cv2 window for both models
def cb(packet: DetectionPacket):
    frame = packet.visualizer.draw(packet.frame)
    cv2.imshow('Frame', frame)

with OakCamera() as oak:
    color = oak.create_camera('color')

    script = oak.pipeline.create(dai.node.Script)
    # When Script node receives a message from the host it will switch streaming␣
↪frames to another NN model
    script.setScript("""
    i = 0
    outputs = ['out1', 'out2']

    while True:
        frame = node.io['frames'].get()

        switch = node.io['switch'].tryGet()
        if switch is not None:
            i += 1
            if len(outputs) <= i:
                i = 0

        node.io[outputs[i]].send(frame)
    """)
    color.stream.link(script.inputs['frames'])

    # We can have multiple models here, not just 2 object detection models
    nn1 = oak.create_nn('yolov6nr3_coco_640x352', input=script.outputs['out1'])
    nn1.config_nn(resize_mode='stretch') # otherwise, BB mappings will be incorrect
    nn2 = oak.create_nn('mobilenet-ssd', input=script.outputs['out2'])
    nn2.config_nn(resize_mode='stretch') # otherwise, BB mappings will be incorrect

    # We will send "switch" message via XLinkIn
    xin = oak.pipeline.create(dai.node.XLinkIn)
    xin.setStreamName('switch')
    xin.out.link(script.inputs['switch'])

    # We don't want syncing, we just want either of the model packets in the callback
    oak.visualize([nn1, nn2], fps=True, callback=cb)

    oak.visualize([nn1.out.passthrough, nn2.out.passthrough], fps=True)

    # oak.show_graph()

    oak.start()
```

(continues on next page)

(continued from previous page)

```
    qin = oak.device.getInputQueue('switch')

    while True:
        key = oak.poll()
        if key == ord('s'):
            print('Switching NN model')
            qin.send(dai.Buffer())
        elif key == ord('q'):
            break
```

10.14 Sync Multiple Outputs

This example shows how to apply software syncing to different outputs of the OAK device. In this example, the color stream is synced with two NeuralNetworks and passthrough.

Note: Visualization in current example is done with blocking behavior. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

10.14.1 Demo

```
jaka@X750LN:~/Desktop/depthai/depthai_sdk/examples$ python3 mixed/sync_multiple_outputs.py
synced! {'color': <depthai_sdk.classes.packets.FramePacket object at 0x7f5d3d6e4790>, 'face-detection_out;face-detection_preview': <depthai
_sdk.classes.packets.DetectionPacket object at 0x7f5d3d6e4490>, 'mobilenet_out;mobilenet_passthrough': <depthai_sdk.classes.packets.Detecti
onPacket object at 0x7f5d3d6e40a0>, 'mobilenet_out;mobilenet_preview': <depthai_sdk.classes.packets.DetectionPacket object at 0x7f5d3d6e407
0>}
synced! {'color': <depthai_sdk.classes.packets.FramePacket object at 0x7f5d3d6e47f0>, 'face-detection_out;face-detection_preview': <depthai
_sdk.classes.packets.DetectionPacket object at 0x7f5d3d6e44c0>, 'mobilenet_out;mobilenet_passthrough': <depthai_sdk.classes.packets.Detecti
onPacket object at 0x7f5d3d6e4160>, 'mobilenet_out;mobilenet_preview': <depthai_sdk.classes.packets.DetectionPacket object at 0x7f5d3d6e446
0>}
synced! {'color': <depthai_sdk.classes.packets.FramePacket object at 0x7f5d3d6e4700>, 'face-detection_out;face-detection_preview': <depthai
_sdk.classes.packets.DetectionPacket object at 0x7f5d3d6e4430>, 'mobilenet_out;mobilenet_passthrough': <depthai_sdk.classes.packets.Detecti
onPacket object at 0x7f5d3d6e45e0>, 'mobilenet_out;mobilenet_preview': <depthai_sdk.classes.packets.DetectionPacket object at 0x7f5d3d6e434
0>}
synced! {'color': <depthai_sdk.classes.packets.FramePacket object at 0x7f5d3d6e4610>, 'face-detection_out;face-detection_preview': <depthai
_sdk.classes.packets.DetectionPacket object at 0x7f5d3d6e4280>, 'mobilenet_out;mobilenet_passthrough': <depthai_sdk.classes.packets.Detecti
onPacket object at 0x7f5d3d6e45e0>, 'mobilenet_out;mobilenet_preview': <depthai_sdk.classes.packets.DetectionPacket object at 0x7f5d3d6e416
0>}
```

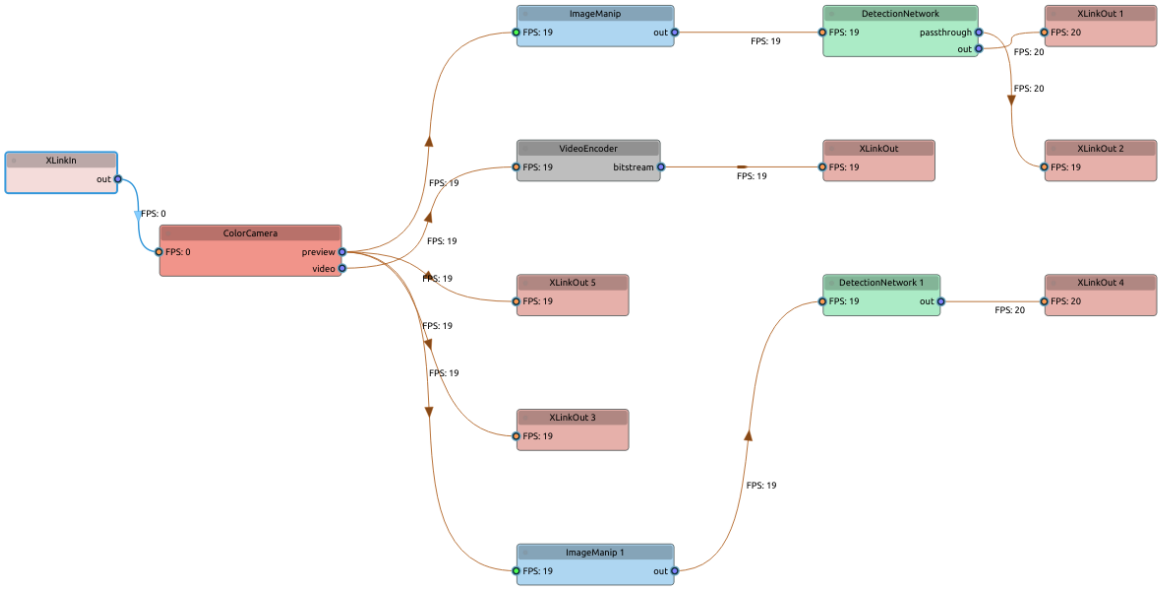
10.14.2 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.14.3 Pipeline



10.14.4 Source Code

Python

Also available on GitHub

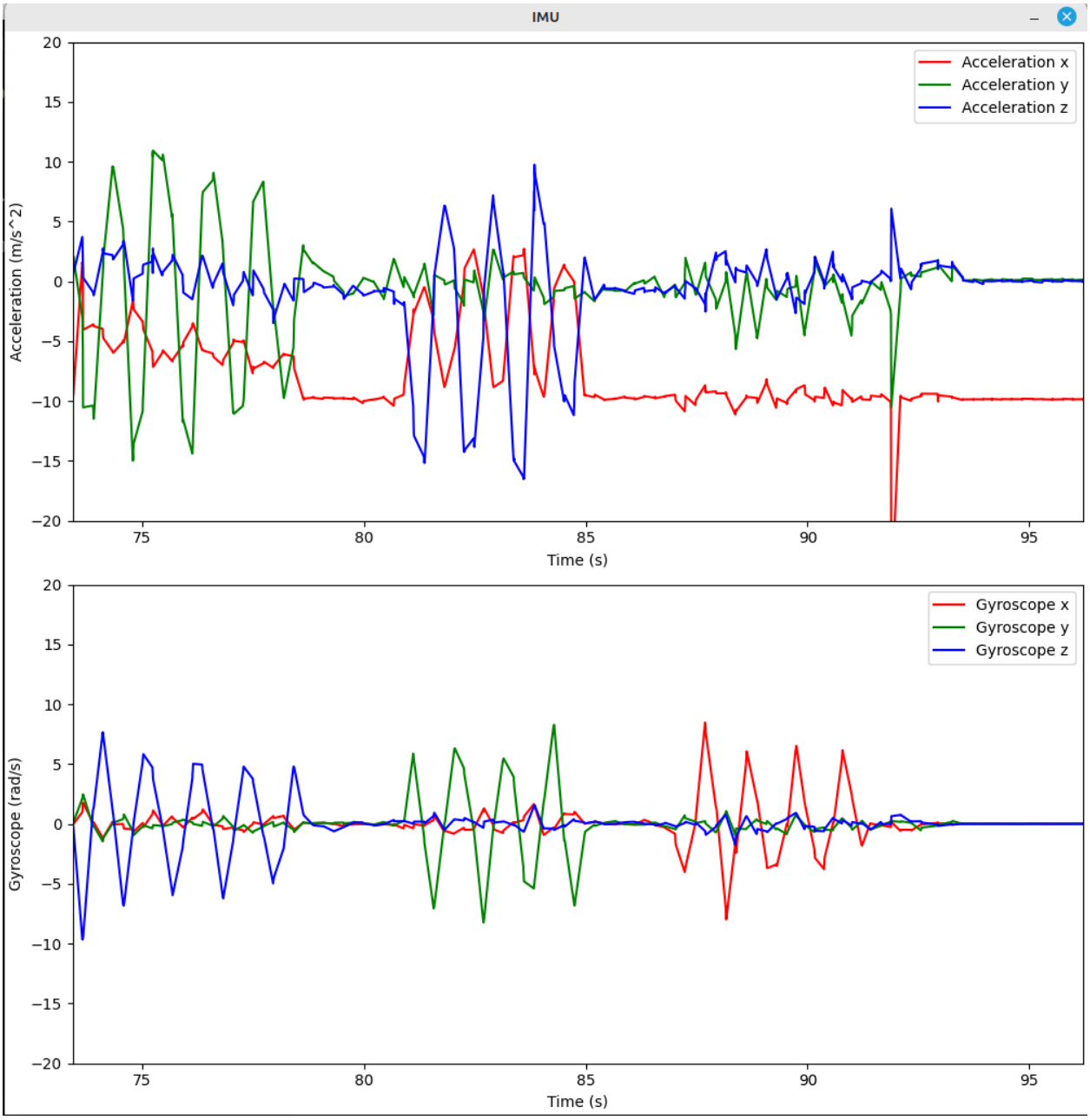
```python
from typing import Dict

from depthai_sdk import OakCamera

with OakCamera() as oak:
    color = oak.create_camera('color', encode='h264')
    nn = oak.create_nn('mobilenet-ssd', color)
    nn2 = oak.create_nn('face-detection-retail-0004', color)

    def cb(msgs: Dict):
        print('====== New synced packets! ======')
        for name, packet in msgs.items():
            print(f"Packet '{name}' with timestamp:", packet.get_timestamp(), 'Seq number:', packet.get_sequence_num(), 'Object', packet)

    oak.callback([nn.out.passthrough, nn.out.encoded, nn2.out.encoded], cb) \
        .configure_syncing(enable_sync=True, threshold_ms=30)
    # oak.show_graph()

    oak.start(blocking=True)
```

10.15 IMU Demonstration

This example showcases how to use the integrated IMU sensor on the OAK-D board with the Depthai sdk. In our example we set the IMU to output data at 400Hz, and batch size to 5. This means we get 5 IMU readings every 12.5ms (2.5ms per reading * 5). We then print out the IMU data to the console.

Note: Visualization in current example is done with blocking behavor. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

10.15.1 Demo


IMU
Acceleration x
Acceleration y
Acceleration z
Acceleration (m/s^2)
Time (s)
Gyroscope x
Gyroscope y
Gyroscope z
Gyroscope (rad/s)
Time (s)

10.15.2 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.15.3 Pipeline

10.15.4 Source Code

Python

Also available on GitHub.

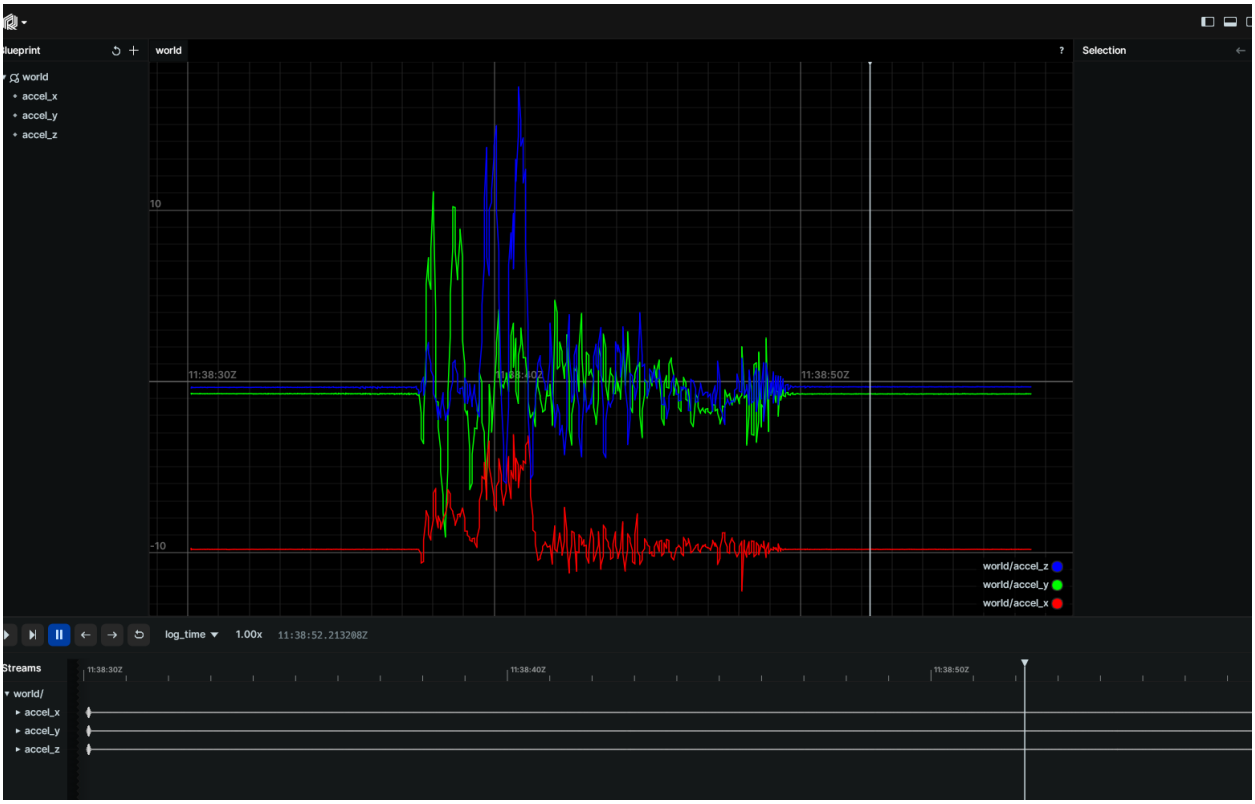
```python
from depthai_sdk import OakCamera

with OakCamera() as oak:
    imu = oak.create_imu()
    imu.config_imu(report_rate=400, batch_report_threshold=5)
    # DepthAI viewer should open, and IMU data can be viewed on the right-side panel,
    # under "Stats" tab (right of the "Device Settings" tab).
    oak.visualize(imu.out.main)
    oak.start(blocking=True)
```

10.16 IMU Rerun Demonstration

This example showcases how to use the integrated IMU sensor on the OAK-D board. In this example, the displaying is done with Rerun (the same core as our DepthAI Viewer).

Note: Visualization in current example is done with blocking behavor. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

10.16.1 Demo



10.16.2 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.16.3 Pipeline

10.16.4 Source Code

Python

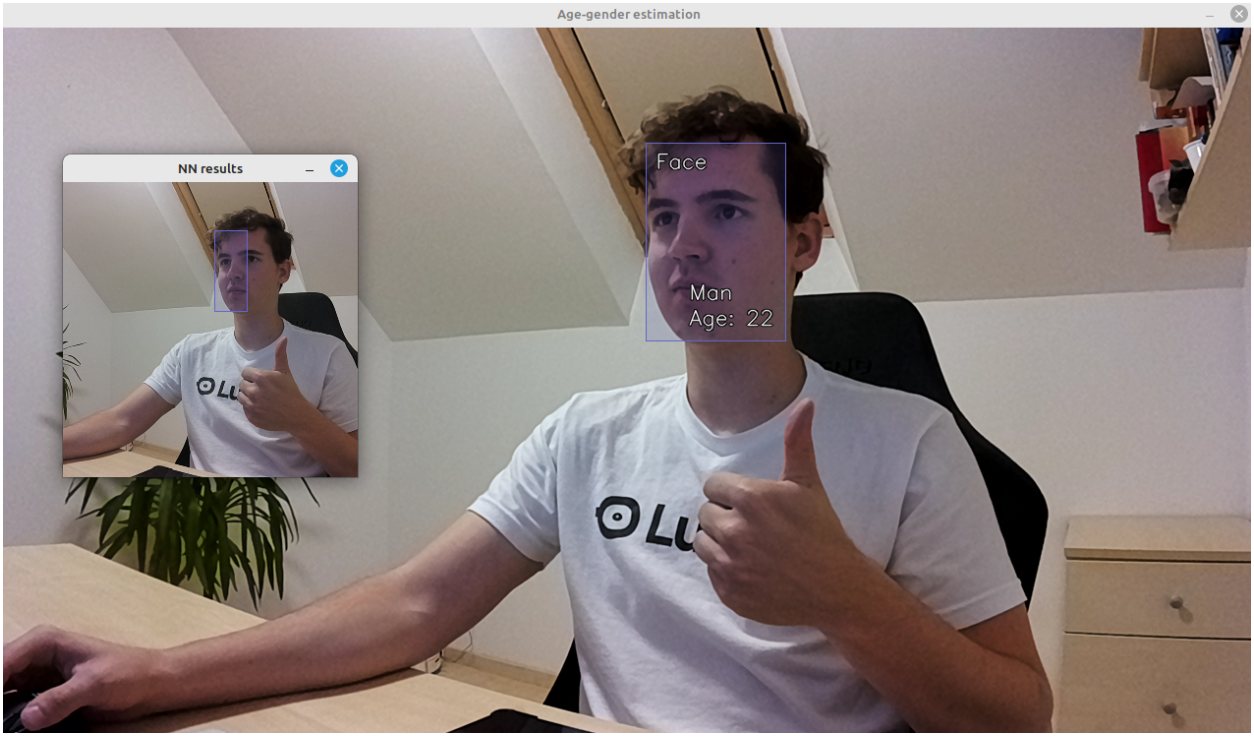
Also available on GitHub.

10.17 Age-Gender Inference

This example showcases the usage of multi-stage neural network pipeline to make age and gender inference on a video frame.

Note: Visualization in current example is done with blocking behavior. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

10.17.1 Demo



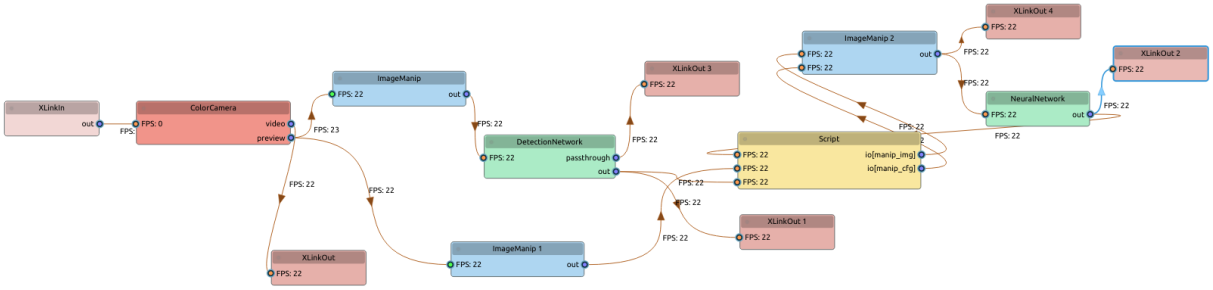
10.17.2 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.17.3 Pipeline



10.17.4 Source Code

Python

Also available on GitHub.

```python
import cv2
import numpy as np

from depthai_sdk import OakCamera
from depthai_sdk.classes import TwoStagePacket
from depthai_sdk.visualize.configs import TextPosition


def callback(packet: TwoStagePacket):
    visualizer = packet.visualizer
    for det, rec in zip(packet.detections, packet.nnData):
        age = int(float(np.squeeze(np.array(rec.getLayerFp16('age_conv3')))) * 100)
        gender = np.squeeze(np.array(rec.getLayerFp16('prob')))
        gender_str = "Woman" if gender[0] > gender[1] else "Man"

        visualizer.add_text(f'{gender_str}\nAge: {age}',
                            bbox=packet.bbox.get_relative_bbox(det.bbox),
                            position=TextPosition.BOTTOM_RIGHT)

    frame = visualizer.draw(packet.frame)
    cv2.imshow('Age-gender estimation', frame)


with OakCamera() as oak:
```

(continues on next page)

(continued from previous page)

```
    color = oak.create_camera('color')
    det = oak.create_nn('face-detection-retail-0004', color)
    det.config_nn(resize_mode='crop')

    age_gender = oak.create_nn('age-gender-recognition-retail-0013', input=det)
    # age_gender.config_multistage_nn(show_cropped_frames=True) # For debugging

    # Visualize detections on the frame. Don't show the frame but send the packet
    # to the callback function (where it will be displayed)
    oak.visualize(age_gender, callback=callback)
    oak.visualize(det.out.passthrough)
    oak.visualize(age_gender.out.twostage_crops)

    # oak.show_graph() # Show pipeline graph, no need for now
    oak.start(blocking=True)  # This call will block until the app is stopped (by␣
↪pressing 'Q' button)
```

10.18 Custom Decode Function

This example showcases the usage of custom decoding functions for the neural network component. More info is available inside the function itself.

Note: Visualization in current example is done with blocking behavor. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

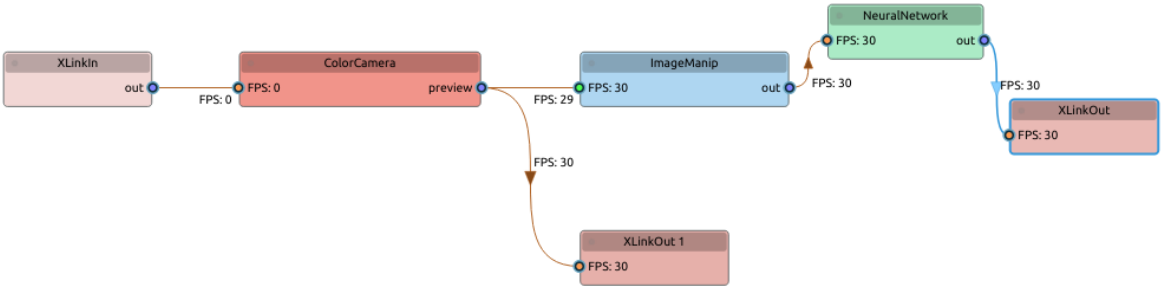
10.18.1 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.18.2 Pipeline



10.18.3 Source Code

Python

Also available on GitHub.

```python
import blobconverter
import numpy as np
import depthai as dai
from depthai_sdk import OakCamera
from depthai_sdk.classes import Detections


def decode(nn_data: dai.NNData) -> Detections:
    """
    Custom decode function for the NN component. Decode function has to accept NNData argument.
    The return type should preferably be a class that inherits from depthai_sdk.classes.GenericNNOutput,
    which support visualization. But this is not required, i.e. the function can return arbitrary type.

    The decoded output can be accessed from the packet object in the callback function via packet.img_detections.
    """
    layer = nn_data.getFirstLayerFp16()
    results = np.array(layer).reshape((1, 1, -1, 7))
    dets = Detections(nn_data)
    for result in results[0][0]:
        if result[2] > 0.3:
            label = int(result[1])
            conf = result[2]
            bbox = result[3:]
            det = dai.ImgDetection()
            det.confidence = conf
            det.label = label
            det.xmin = bbox[0]
            det.ymin = bbox[1]
            det.xmax = bbox[2]
            det.ymax = bbox[3]
```

(continues on next page)

(continued from previous page)

```
            dets.detections.append(det)

    return dets

with OakCamera() as oak:
    color = oak.create_camera('color')

    nn_path = blobconverter.from_zoo(name='person-detection-0200', version='2021.4',␣
↪shaves=6)
    nn = oak.create_nn(nn_path, color, decode_fn=decode)

    oak.visualize(nn)
    oak.start(blocking=True)
```

10.19 Deeplabv3 Person Segmentation

This example showcases the implementation of deepLabv3 person segmentation model with DepthAI SDK.

Note: Visualization in current example is done with blocking behavor. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

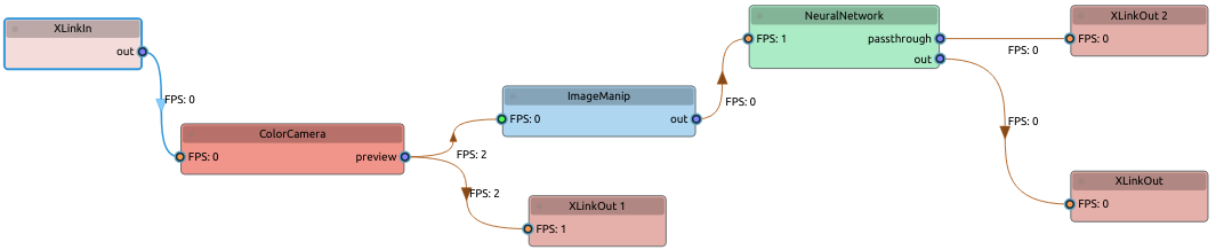
10.19.1 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.19.2 Pipeline



10.19.3 Source Code

One thing worth noting is the resize mode option. Because inference is done on a color camera which has a 16:9 aspect ratio, and the model expects a 1:1 aspect ratio, we need to resize the input frame to fit the model. This is done in three ways:

- letterbox - resize the frame to fit the model, and pad the rest with black pixels
- crop - crop the frame to fit the model
- stretch - stretch the frame to fit the model

More information at Maximizing FOV.

Python

Also available on GitHub

10.20 Emotion Recognition

This example showcases the implementation of two stage neural network pipeline, where the first stage is a face detection network, and the second stage is an emotion recognition model.

Note: Visualization in current example is done with blocking behavior. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

10.20.1 Demo

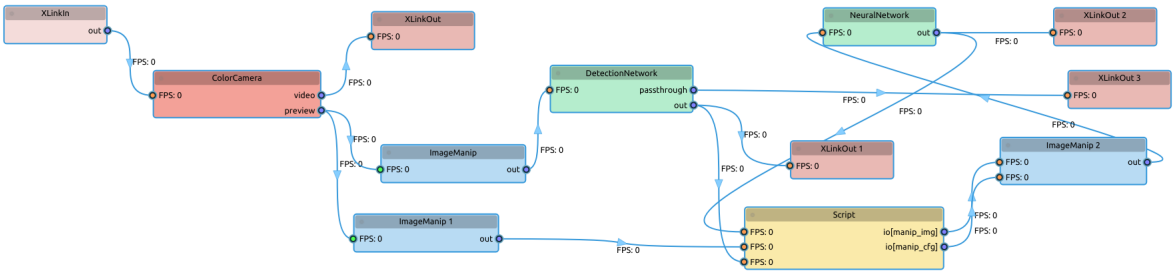
10.20.2 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.20.3 Pipeline



10.20.4 Source Code

Python

Also available on GitHub.

```python
import cv2
import numpy as np

from depthai_sdk import OakCamera
from depthai_sdk.classes import TwoStagePacket
from depthai_sdk.visualize.configs import TextPosition

emotions = ['neutral', 'happy', 'sad', 'surprise', 'anger']


def callback(packet: TwoStagePacket):
    visualizer = packet.visualizer

    for det, rec in zip(packet.detections, packet.nnData):
        emotion_results = np.array(rec.getFirstLayerFp16())
        emotion_name = emotions[np.argmax(emotion_results)]

        visualizer.add_text(emotion_name,
                            bbox=packet.bbox.get_relative_bbox(det.bbox),
                            position=TextPosition.BOTTOM_RIGHT)

    visualizer.draw(packet.frame)
    cv2.imshow(packet.name, packet.frame)


with OakCamera() as oak:
    color = oak.create_camera('color')
    det = oak.create_nn('face-detection-retail-0004', color)
    # Passthrough is enabled for debugging purposes
    det.config_nn(resize_mode='crop')

    emotion_nn = oak.create_nn('emotions-recognition-retail-0003', input=det)
    # emotion_nn.config_multistage_nn(show_cropped_frames=True) # For debugging

    # Visualize detections on the frame. Also display FPS on the frame. Don't show␣
↪the frame but send the packet
```

(continues on next page)

(continued from previous page)

```
    # to the callback function (where it will be displayed)
    oak.visualize(emotion_nn, callback=callback, fps=True)
    oak.visualize(det.out.passthrough)
    # oak.show_graph() # Show pipeline graph, no need for now
    oak.start(blocking=True)  # This call will block until the app is stopped (by␣
↪pressing 'Q' button)
```

10.21 Face Detection RGB

This example shows how to run face detection on RGB camera input using SDK.

For running the same face detection on mono camera, see *Face Detection Mono*.

Note: Visualization in current example is done with blocking behavior. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

10.21.1 Demo

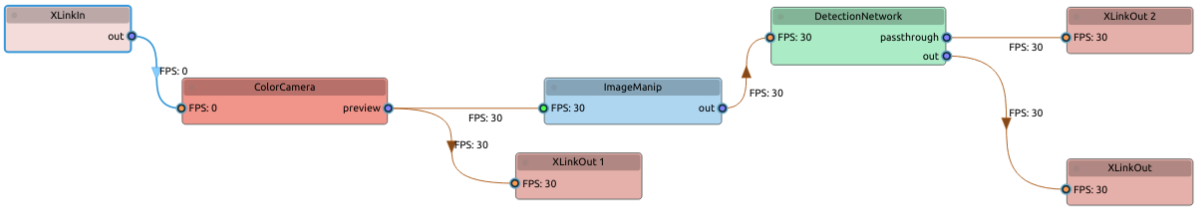
10.21.2 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.21.3 Pipeline



10.21.4 Source Code

Python

Also available on GitHub.

```python
from depthai_sdk import OakCamera

with OakCamera() as oak:
    color = oak.create_camera('color')
    nn = oak.create_nn('face-detection-retail-0004', color)
    oak.visualize([nn.out.main, nn.out.passthrough], scale=2/3, fps=True)
    oak.start(blocking=True)
```

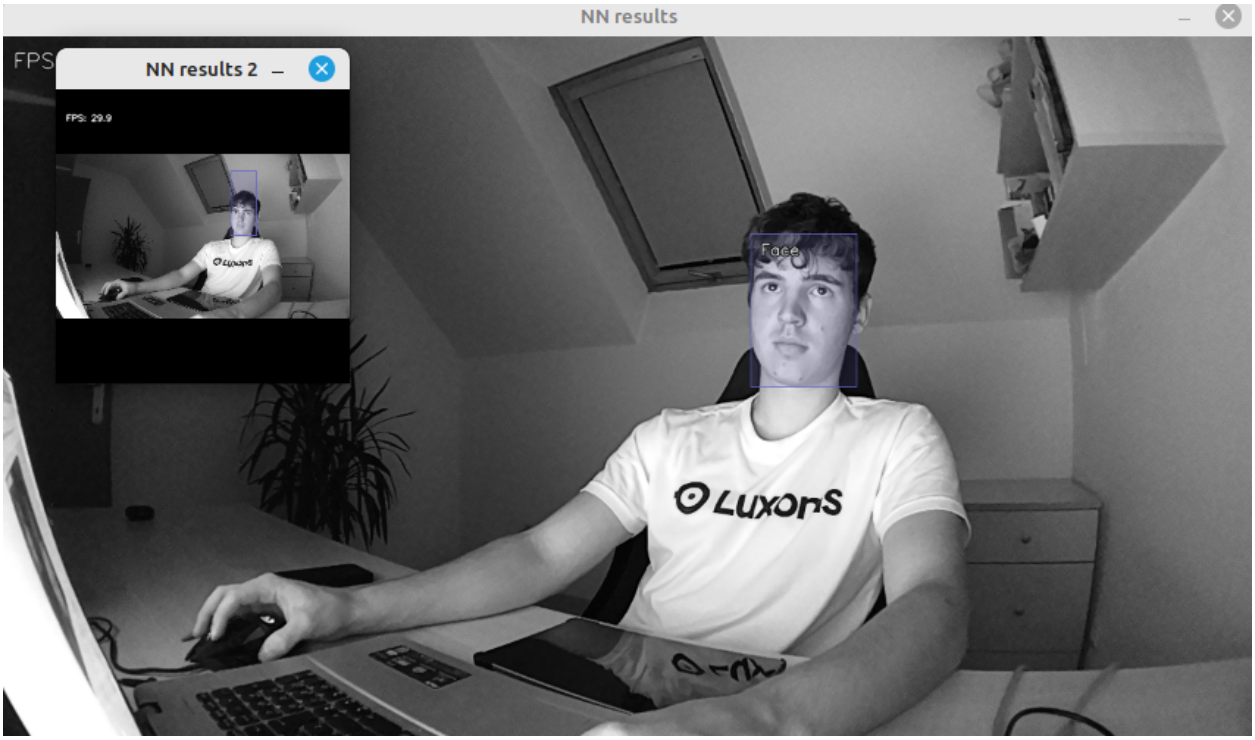
10.22 Face Detection Mono

This example shows how to run face detection on Mono camera input using SDK.

For running the same face detection on RGB camera, see *Face Detection RGB*.

Note: Visualization in current example is done with blocking behavior. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

10.22.1 Demo



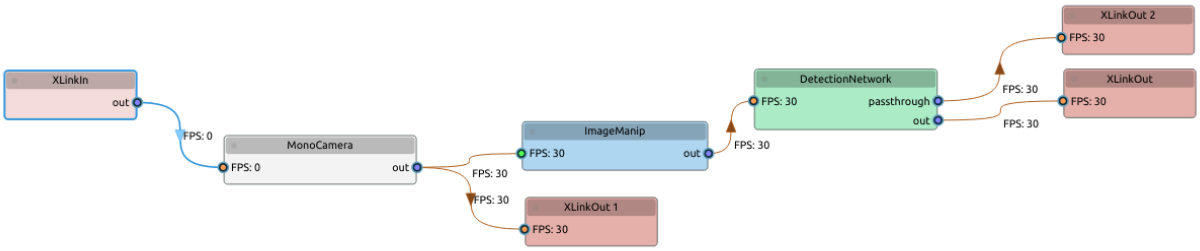
10.22.2 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.22.3 Pipeline



10.22.4 Source Code

Python

Also available on GitHub.

```
from depthai_sdk import OakCamera

with OakCamera() as oak:
    left = oak.create_camera('left')
    nn = oak.create_nn('face-detection-retail-0004', left)
    oak.visualize([nn.out.main, nn.out.passthrough], scale=2/3, fps=True)
    oak.start(blocking=True)
```

10.23 Human Pose Estimation

This example showcases the implementation of a human pose estimation network using the DepthAI SDK.

Note: Visualization in current example is done with blocking behavior. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

10.23.1 Demo

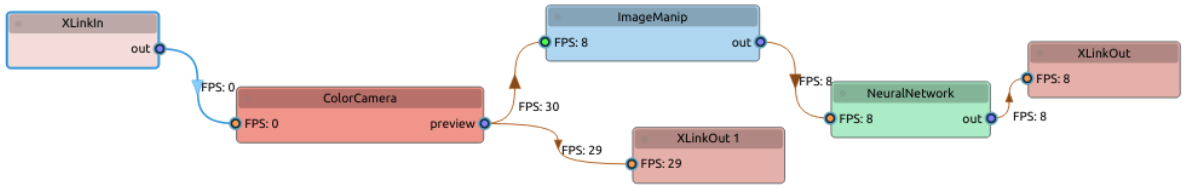
10.23.2 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.23.3 Pipeline



10.23.4 Source Code

Python

Also available on GitHub.

```
from depthai_sdk import OakCamera

with OakCamera() as oak:
    color = oak.create_camera('color')
    # List of models that are supported out-of-the-box by the SDK:
    # https://docs.luxonis.com/projects/sdk/en/latest/features/ai_models/#sdk-
↪supported-models
    human_pose_nn = oak.create_nn('human-pose-estimation-0001', color)

    oak.visualize(human_pose_nn)
    oak.start(blocking=True)
```

10.24 MobileNet Encoded

This example shows how to run an encoded RGB stream through a neural network and display the encoded results.

Note: Visualization in current example is done with blocking behavor. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

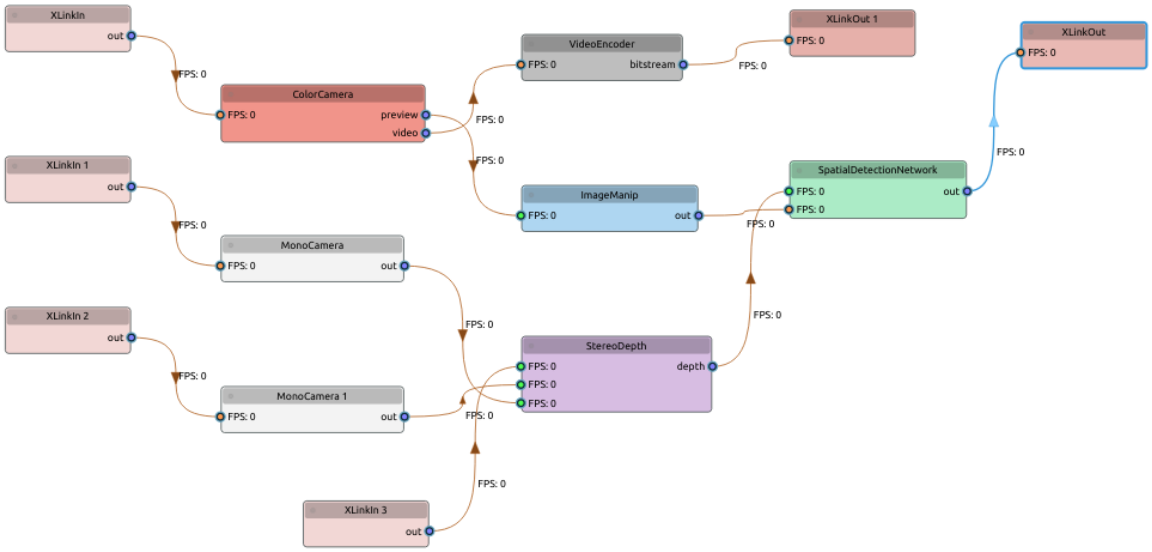
10.24.1 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.24.2 Pipeline



10.24.3 Source Code

Python

Also available on GitHub.

```
from depthai_sdk import OakCamera

with OakCamera() as oak:
    color = oak.create_camera('color', encode='mjpeg', fps=10)

    nn = oak.create_nn('mobilenet-ssd', color, spatial=True)  # spatial flag␣
↪indicates that we want to get spatial data

    oak.visualize([nn.out.encoded])  # Display encoded output
    oak.start(blocking=True)
```

10.25 Neural Network Component

This example shows how to run run a color camera stream through a YoloV7 model and display the results on the host.

For additional models, check: models supported by SDK

Note: Visualization in current example is done with blocking behavior. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

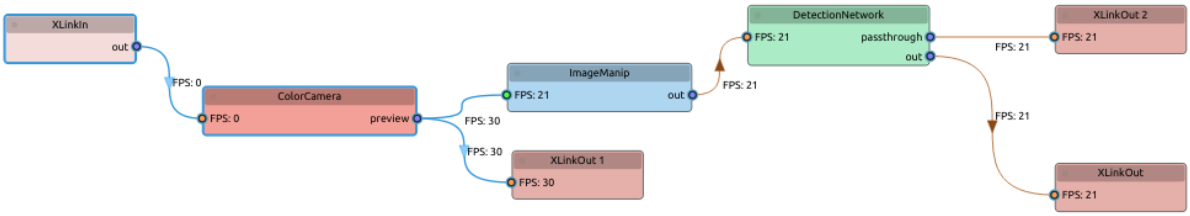
10.25.1 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.25.2 Pipeline



10.25.3 Source Code

Python

Also available on GitHub.

```python
from depthai_sdk import OakCamera

with OakCamera() as oak:
    color = oak.create_camera('color')
    # List of models that are supported out-of-the-box by the SDK:
    # https://docs.luxonis.com/projects/sdk/en/latest/features/ai_models/#sdk-supported-models
    nn = oak.create_nn('yolov5n_coco_416x416', color)
    nn.config_nn(resize_mode='stretch')
    oak.visualize([nn.out.main], fps=True)
    oak.visualize(nn.out.passthrough)
    oak.start(blocking=True)
```

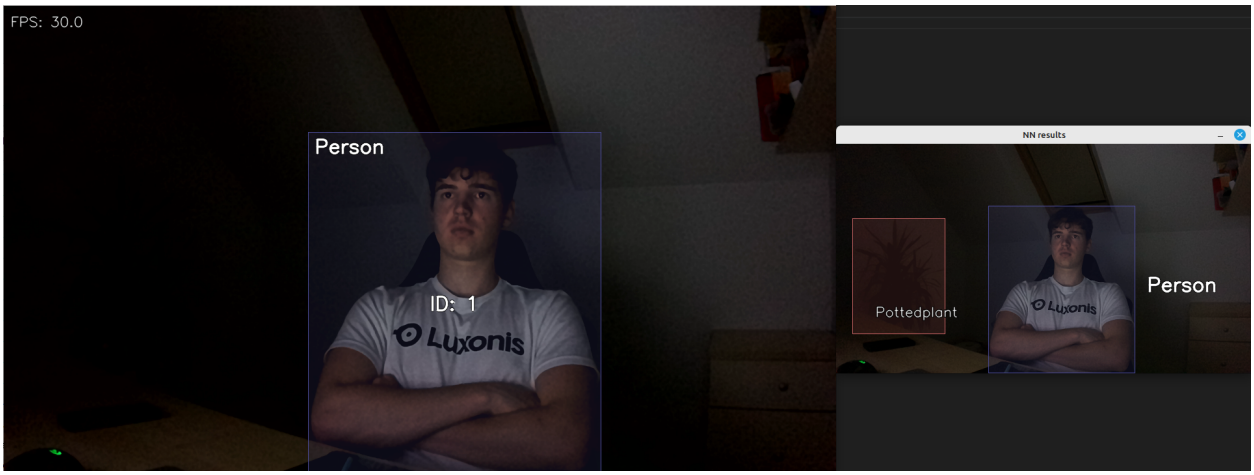
10.26 Object Tracking

This example showcases the usage of object tracking in Depthai SDK.

For more information about tracker configuration, please refer to config tracker reference.

Note: Visualization in current example is done with blocking behavor. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

10.26.1 Demo



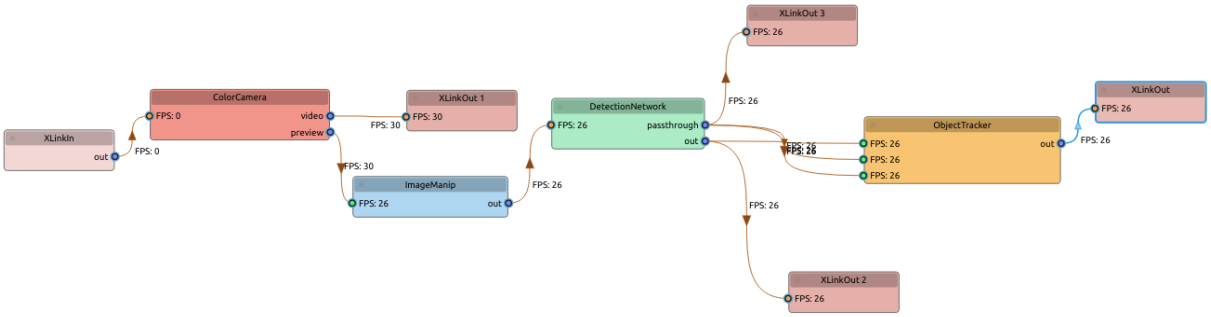
10.26.2 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.26.3 Pipeline



10.26.4 Source Code

Python

Also available on GitHub.

```python
from depthai_sdk import OakCamera
import depthai as dai

with OakCamera() as oak:
    color = oak.create_camera('color')
    # List of models that are supported out-of-the-box by the SDK:
    # https://docs.luxonis.com/projects/sdk/en/latest/features/ai_models/#sdk-supported-models
    nn = oak.create_nn('yolov6nr3_coco_640x352', color, tracker=True)

    nn.config_nn(resize_mode='stretch')
    nn.config_tracker(
        tracker_type=dai.TrackerType.ZERO_TERM_COLOR_HISTOGRAM,
        track_labels=[0], # Track only 1st object from the object map. If unspecified, track all object types
        # track_labels=['person'] # Track only people (for coco datasets, person is 1st object in the map)
        assignment_policy=dai.TrackerIdAssignmentPolicy.SMALLEST_ID,
        max_obj=10, # Max objects to track, which can improve performance
        threshold=0.1 # Tracker threshold
    )

    oak.visualize([nn.out.tracker], fps=True)
    oak.visualize(nn.out.passthrough)
    oak.start(blocking=True)
```

10.27 Roboflow Integration

This example showcases the usage of the ROBOFLOW platform to train a custom object detection model and use it with DepthAI SDK.

Note: Visualization in current example is done with blocking behavor. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

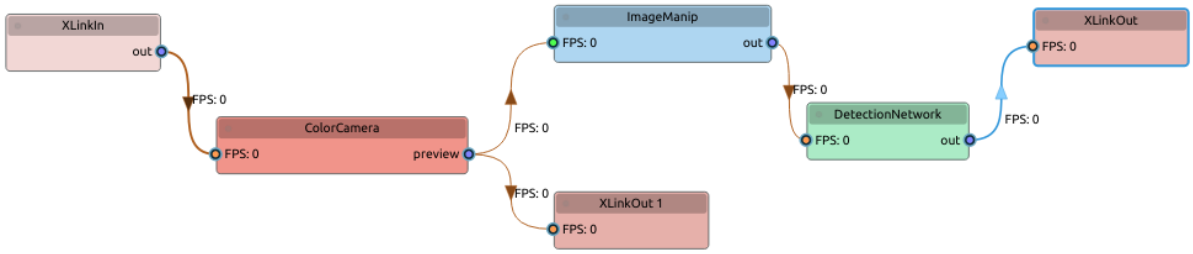
10.27.1 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.27.2 Pipeline



10.27.3 Source Code

Python

Also available on GitHub.

```python
from depthai_sdk import OakCamera

# Download & deploy a model from Roboflow universe:
# # https://universe.roboflow.com/david-lee-d0rhs/american-sign-language-letters/dataset/6

with OakCamera() as oak:
    color = oak.create_camera('color')
    model_config = {
        'source': 'roboflow', # Specify that we are downloading the model from Roboflow
```

(continues on next page)

(continued from previous page)

```
        'model':'american-sign-language-letters/6',
        'key':'181b0f6e43d59ee5ea421cd77f6d9ea2a4b059f8' # Fake API key, replace with␣
↪your own!
    }
    nn = oak.create_nn(model_config, color)
    oak.visualize(nn, fps=True)
    oak.start(blocking=True)
```

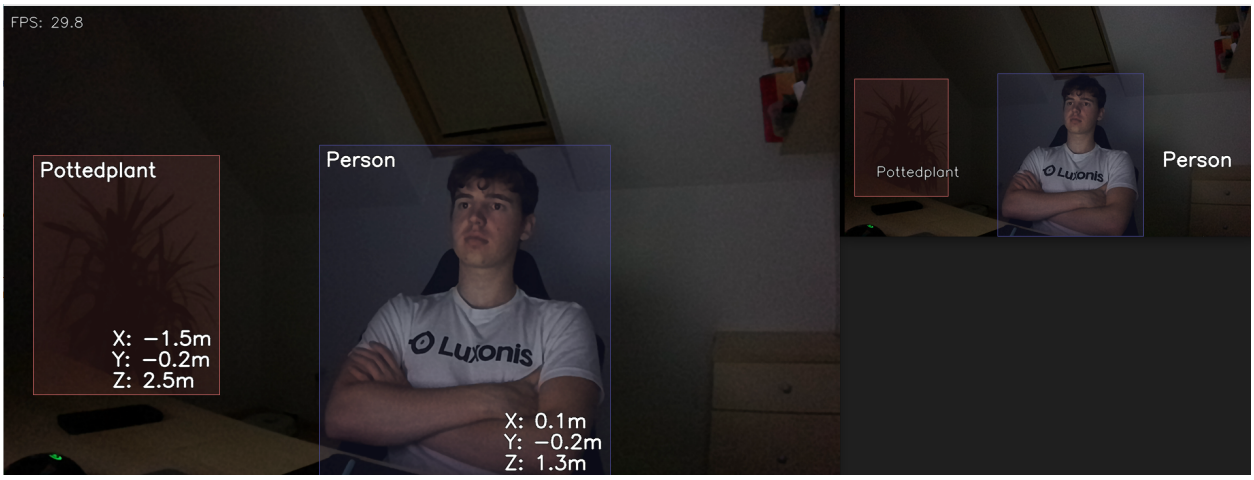
10.28 Spatial Detection

This example showcases the usage of spatial detection using MobileNet-SSD neural network.

For more information about spatial configuration (thresholds, averaging), please refer to config spatial reference.

Note: Visualization in current example is done with blocking behavor. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

10.28.1 Demo



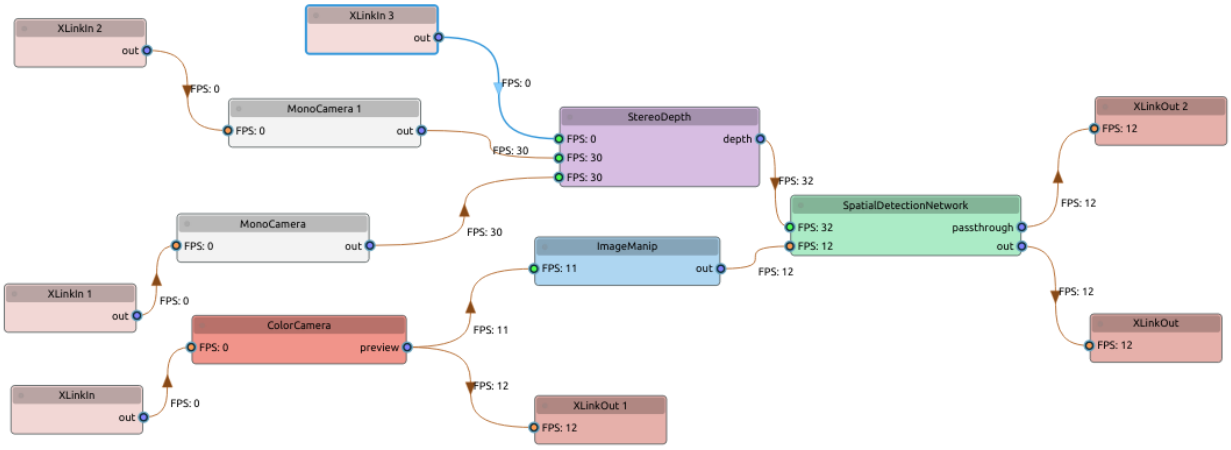
10.28.2 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.28.3 Pipeline



10.28.4 Source Code

Python

Also available on GitHub

```python
from depthai_sdk import OakCamera
import depthai as dai

with OakCamera() as oak:
    color = oak.create_camera('color')
    # List of models that are supported out-of-the-box by the SDK:
    # https://docs.luxonis.com/projects/sdk/en/latest/features/ai_models/#sdk-supported-models
    nn = oak.create_nn('yolov6nr3_coco_640x352', color, spatial=True)

    nn.config_spatial(
        bb_scale_factor=0.5, # Scaling bounding box before averaging the depth in that ROI
        lower_threshold=300, # Discard depth points below 30cm
        upper_threshold=10000, # Discard depth pints above 10m
        # Average depth points before calculating X and Y spatial coordinates:
        calc_algo=dai.SpatialLocationCalculatorAlgorithm.AVERAGE
    )

    oak.visualize(nn.out.main, fps=True)
    oak.visualize([nn.out.passthrough, nn.out.spatials])
    oak.start(blocking=True)
```

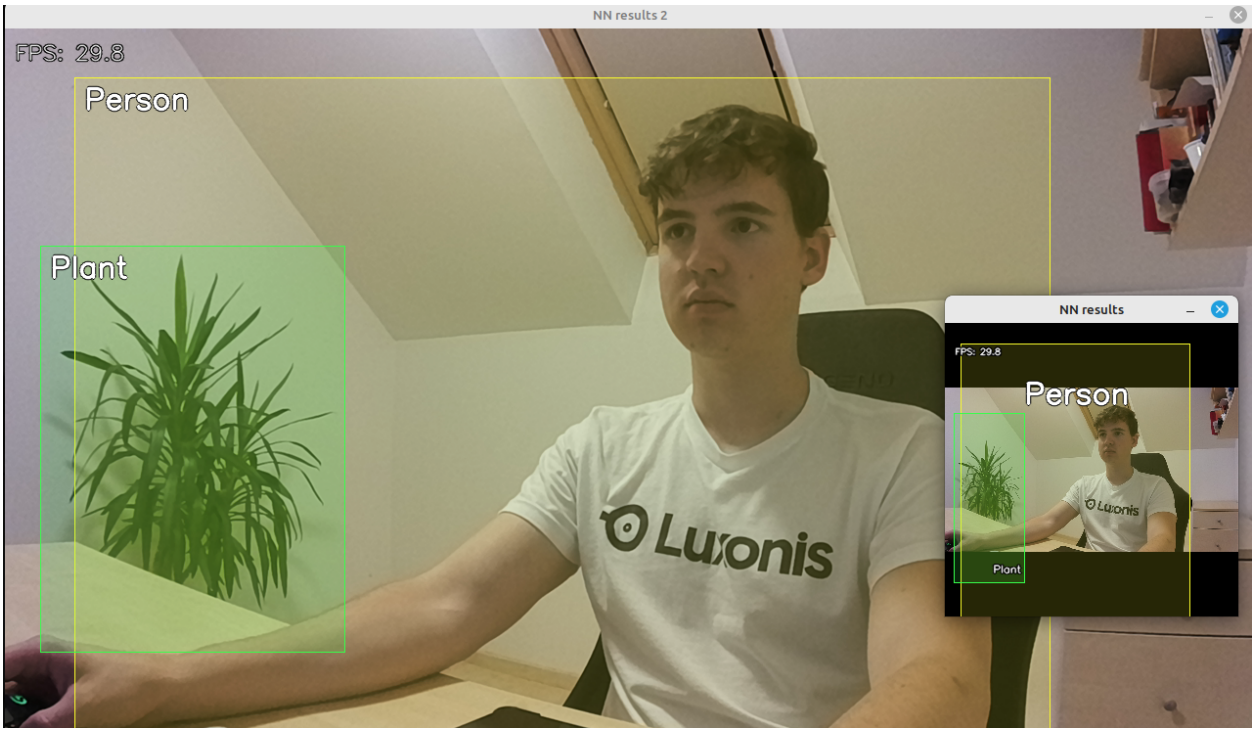
10.29 YOLO SDK

This example showcases the implementation of Yolov3 object detection network with DepthAI SDK.

For more information about tracker configuration, please refer to config tracker reference.

Note: Visualization in current example is done with blocking behavor. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

10.29.1 Demo



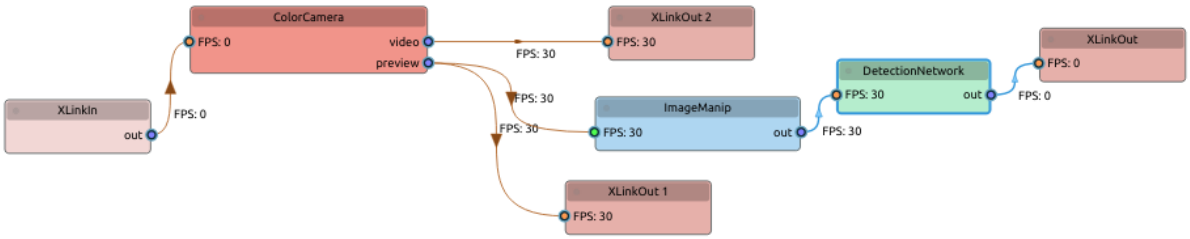
10.29.2 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.29.3 Pipeline



10.29.4 Source Code

Python

Also available on GitHub.

```
from depthai_sdk import OakCamera

with OakCamera() as oak:
    color = oak.create_camera('color')
    nn = oak.create_nn('yolo-v3-tf', color)
    oak.visualize([nn, color], scale=2 / 3, fps=True)  # 1080P -> 720P
    # oak.show_graph()
    oak.start(blocking=True)
```

10.30 Pointcloud Demo

This example shows how to create and display pointclouds with DepthAI SDK.

Note: Visualization in current example is done with blocking behavior. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

10.30.1 Demo

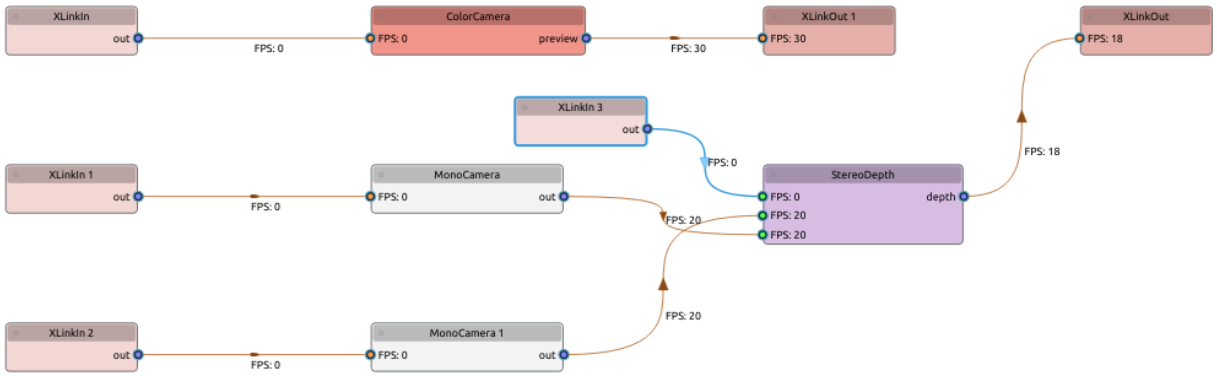
10.30.2 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.30.3 Pipeline



10.30.4 Source Code

Python

Also available on GitHub.

```python
from depthai_sdk import OakCamera

with OakCamera() as oak:
    color = oak.camera('color')
    stereo = oak.create_stereo()
    stereo.config_stereo(align=color)
    pcl = oak.create_pointcloud(stereo=stereo, colorize=color)
    oak.visualize(pcl, visualizer='depthai-viewer')
    oak.start(blocking=True)
```

10.31 Encode Multiple Streams

This example showcases how to encode video from the camera and save it to a file. Possible encodings are: `H264`, `H265` and `MJPEG`.

Note: Visualization in current example is done with blocking behavor. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

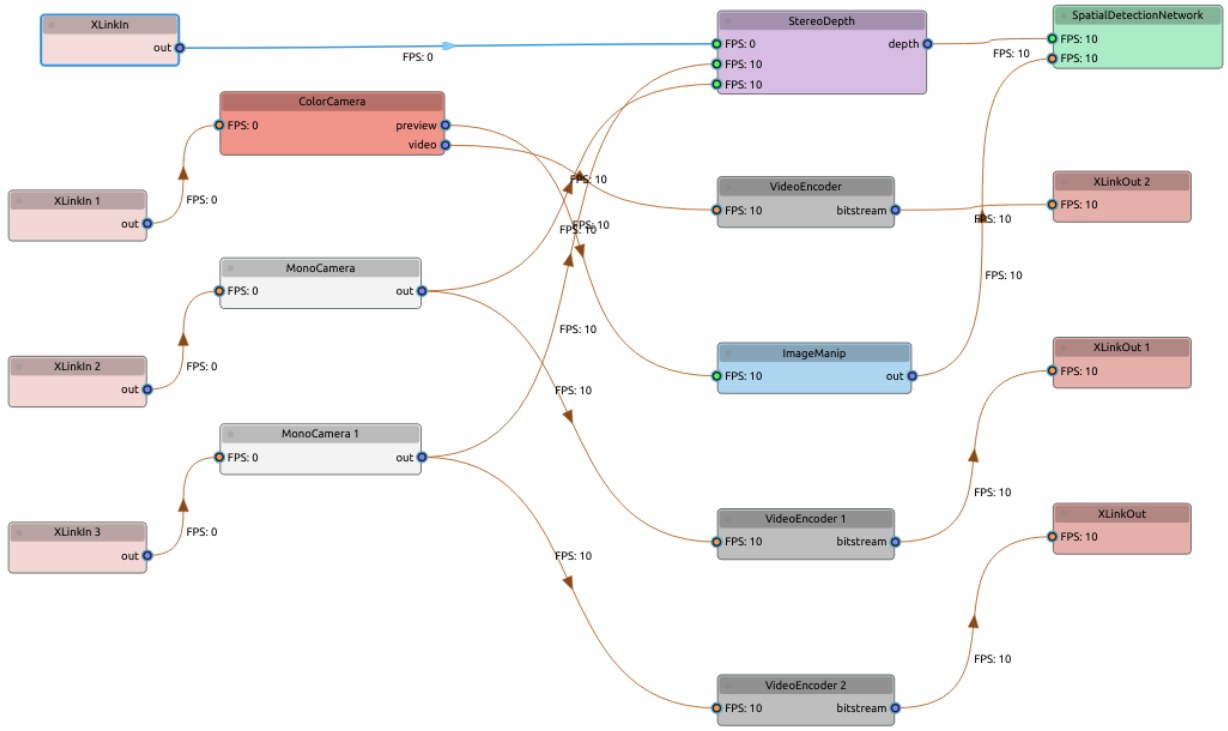
10.31.1 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.31.2 Pipeline



10.31.3 Source Code

Python

Also available on GitHub.

```
from depthai_sdk import OakCamera, RecordType

with OakCamera() as oak:
    color = oak.create_camera('color', resolution='1080P', fps=10, encode='H265')
    left = oak.create_camera('left', resolution='800p', fps=10, encode='H265')
    right = oak.create_camera('right', resolution='800p', fps=10, encode='H265')

    stereo = oak.create_stereo(left=left, right=right)
    nn = oak.create_nn('mobilenet-ssd', color, spatial=stereo)

    # Sync & save all (encoded) streams
    oak.record([color.out.encoded, left.out.encoded, right.out.encoded], './record', RecordType.VIDEO) \
        .configure_syncing(enable_sync=True, threshold_ms=50)

    oak.visualize([color.out.encoded], fps=True)

    oak.start(blocking=True)
```

10.32 Preview Encoder

This example shows how to use the callback function to write MJPEG encoded frames from color camera to a file.

Note: Visualization in current example is done with blocking behavior. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

10.32.1 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.32.2 Pipeline

10.32.3 Source Code

Python

Also available on GitHub.

```
from pathlib import Path

from depthai_sdk import OakCamera
from depthai_sdk.recorders.video_writers.av_writer import AvWriter

fourcc = 'h264' # Can be 'mjpeg', 'h264', or 'hevc'

rec = AvWriter(Path('./'), 'color', fourcc=fourcc)

def save_raw_mjpeg(packet):
    rec.write(packet.msg)

with OakCamera() as oak:
    color = oak.create_camera('color', encode=fourcc, fps=20)

    # Stream encoded video packets to host. For visualization, we decode them
    # on the host side, and for callback we write encoded frames directly to disk.
    oak.visualize(color.out.encoded, scale=2 / 3, fps=True)
    oak.callback(color.out.encoded, callback=save_raw_mjpeg)

    oak.start(blocking=True)

rec.close()
```

10.33 MCAP Recording

This example showcases the use of SDK to save to MCAP file format. The MCAP file contains color as well as both left and right mono cameras and their inferred depth map.

Note: Visualization in current example is done with blocking behavor. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

10.33.1 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.33.2 Source Code

Python

Also available on GitHub.

```python
from depthai_sdk import OakCamera, RecordType

with OakCamera() as oak:
    color = oak.create_camera('color', resolution='1080P', fps=30, encode='MJPEG')
    color.config_color_camera(isp_scale=(2, 3)) # 720P
    left = oak.create_camera('left', resolution='400p', fps=30)
    right = oak.create_camera('right', resolution='400p', fps=30)
    stereo = oak.create_stereo(left=left, right=right)

    # Sync & save all streams
    recorder = oak.record([color.out.encoded, left, right, stereo.out.depth], './', RecordType.MCAP)
    # recorder.config_mcap(pointcloud=True)
    oak.visualize(left)
    oak.start(blocking=True)
```

10.34 MCAP IMU Recording

This example showcases how to record IMU data along with depth and save both in an MCAP file.

Note: Visualization in current example is done with blocking behavior. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

10.34.1 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.34.2 Source Code

Python

Also available on GitHub.

```
from depthai_sdk import OakCamera, RecordType

with OakCamera() as oak:
    left = oak.create_camera('left', resolution='400p', fps=30)
    right = oak.create_camera('right', resolution='400p', fps=30)
    stereo = oak.create_stereo(left=left, right=right)

    imu = oak.create_imu()
    imu.config_imu(report_rate=500, batch_report_threshold=5)

    # Note that for MCAP recording, user has to have ROS installed
    recorder = oak.record([imu, stereo.out.depth], './', RecordType.MCAP)

    oak.visualize([left, stereo])
    oak.start(blocking=True)
```

10.35 Hardcode Recording Duration

This example shows how to record a video for a fixed duration of time.

Note: Visualization in current example is done with blocking behavior. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

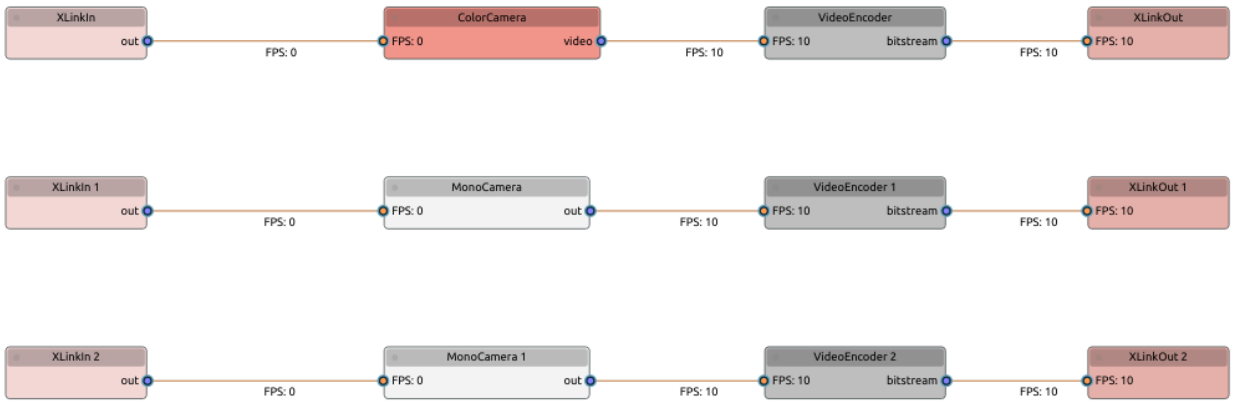
10.35.1 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.35.2 Pipeline



10.35.3 Source Code

Python

Also available on GitHub.

```python
from depthai_sdk import OakCamera, RecordType
import time

with OakCamera() as oak:
    color = oak.create_camera('color', resolution='1080P', fps=10, encode='H265')
    left = oak.create_camera('left', resolution='800p', fps=10, encode='H265')
    right = oak.create_camera('right', resolution='800p', fps=10, encode='H265')

    # Sync & save all (encoded) streams
    oak.record([color.out.encoded, left.out.encoded, right.out.encoded], './record')
    oak.start()
    start_time = time.monotonic()
    while oak.running():
        if time.monotonic() - start_time > 5:
            break
        oak.poll()
```

10.36 ROSBAG Recording

This example showcases the use of SDK to save color, mono, depth and IMU data to a ROSBAG file. This can be useful for recording data for later use, or for testing purposes.

Note: Visualization in current example is done with blocking behavior. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

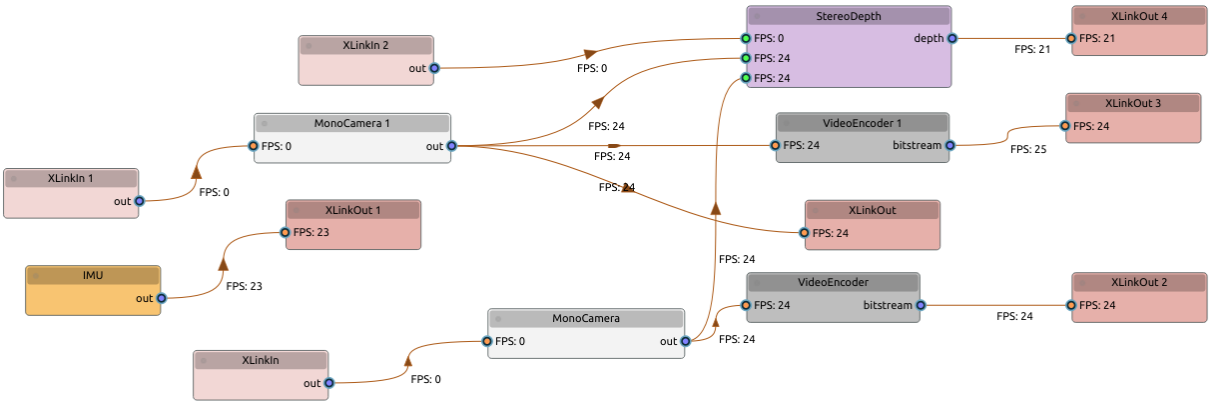
10.36.1 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.36.2 Pipeline



10.36.3 Source Code

Python

Also available on GitHub.

```python
from depthai_sdk import OakCamera, RecordType

with OakCamera() as oak:
    color = oak.create_camera('color', encode='jpeg', fps=30)
    left = oak.create_camera('left', resolution='800p', encode='jpeg', fps=30)
    right = oak.create_camera('right', resolution='800p', encode='jpeg', fps=30)
    stereo = oak.create_stereo(left=left, right=right)
    stereo.config_stereo(align=color)
    imu = oak.create_imu()
    imu.config_imu(report_rate=400, batch_report_threshold=5)

    # DB3 / ROSBAG. ROSBAG doesn't require having ROS installed, while DB3 does.
    record_components = [left.out.encoded, color.out.encoded, right.out.encoded, stereo.out.depth, imu]
    oak.record(record_components, 'record', record_type=RecordType.ROSBAG)

    # Visualize only color stream
    oak.visualize(color.out.encoded)
    oak.start(blocking=True)
```

10.37 Stereo Recording

This example shows how to record disparity map to a file.

Note: Visualization in current example is done with blocking behavor. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

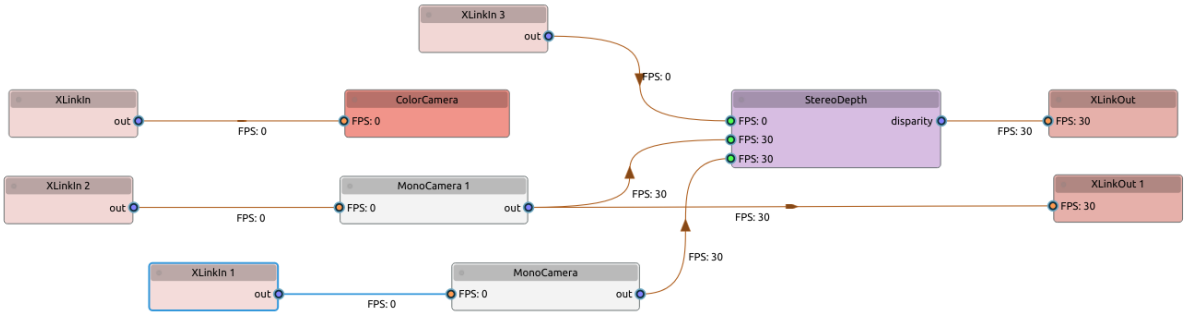
10.37.1 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.37.2 Pipeline



10.37.3 Source Code

Python

Also available on GitHub.

```python
import cv2

from depthai_sdk import OakCamera
from depthai_sdk.visualize.configs import StereoColor

with OakCamera() as oak:
    color = oak.create_camera('color', resolution='1080p', fps=30)
    stereo = oak.create_stereo('400p', fps=30)

    stereo.config_postprocessing(
```

(continues on next page)

(continued from previous page)

```
        colorize=StereoColor.RGB,
        colormap=cv2.COLORMAP_JET
    )

    stereo.config_wls(
        wls_level='high'  # options: 'low', 'medium', 'high'
    )

    # Record RGB and disparity to records folder
    # Record doesn't work with visualize so the config is ignored
    # oak.record([color.out.main, stereo.out.disparity], 'records')

    # Record depth only
    oak.visualize(stereo.out.disparity, record_path='disparity.mp4')

    oak.start(blocking=True)
```

10.38 Object counting on images

This example cycles through a folder of images and counts the number of objects (people in our case) in each image. It displays the count number on the top of the image. It cycles through each image every 3 seconds, but you can change that with:

```
with OakCamera('path/to/folder') as oak:
  oak.replay.set_fps(0.5) # For switching cycling through image every 2 seconds
  # ...
```

Note: Visualization in current example is done with blocking behavor. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

10.38.1 Demo

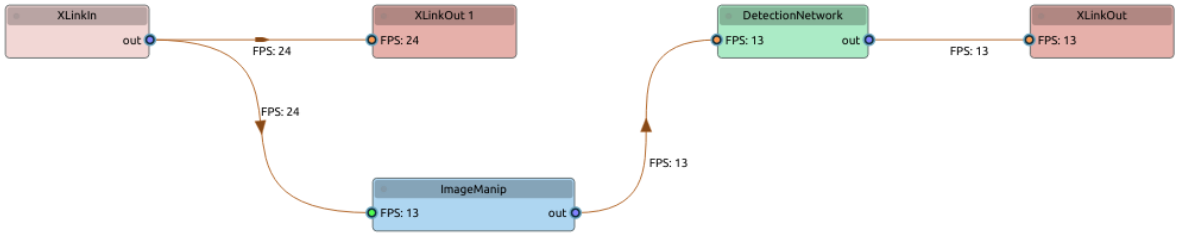
10.38.2 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.38.3 Pipeline



10.38.4 Source Code

Python

Also available on GitHub.

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

import cv2

from depthai_sdk import OakCamera
from depthai_sdk.classes import DetectionPacket
from depthai_sdk.visualize.configs import TextPosition


def callback(packet: DetectionPacket):
    visualizer = packet.visualizer
    num = len(packet.img_detections.detections)
    print('New msgs! Number of people detected:', num)

    visualizer.add_text(f"Number of people: {num}", position=TextPosition.TOP_MID)
    visualizer.draw(packet.frame)
    cv2.imshow(f'frame {packet.name}', packet.frame)


with OakCamera(replay='people-images-01') as oak:
    color = oak.create_camera('color')
    nn = oak.create_nn('person-detection-retail-0013', color)
    oak.replay.set_fps(0.5)

    oak.visualize(nn, callback=callback)
    # oak.show_graph()
    oak.start(blocking=True)
```

10.39 Looped Replay

This example shows how to run replay in a loop. This means the device won't close when the replay file ends.

Note: Visualization in current example is done with blocking behavor. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

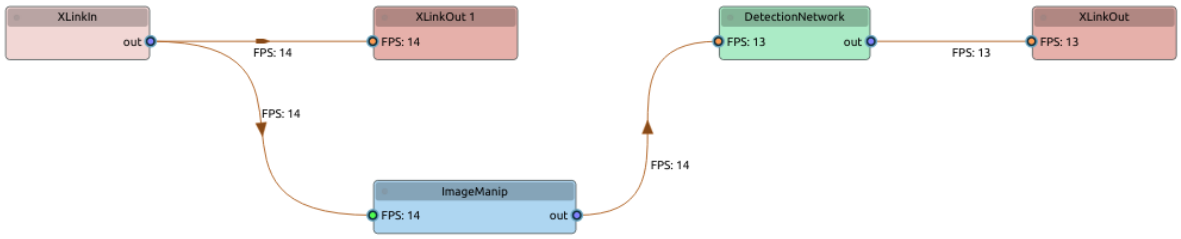
10.39.1 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.39.2 Pipeline



10.39.3 Source Code

Python

Also available on GitHub.

```python
from depthai_sdk import OakCamera

with OakCamera(replay='https://www.youtube.com/watch?v=Y1jTEyb3wiI') as oak:
    oak.replay.set_loop(True)  # <--- Enable looping of the video, so it will never end

    color = oak.create_camera('color')
    nn = oak.create_nn('vehicle-detection-0202', color)
    oak.visualize(nn, fps=True)
    oak.start(blocking=True)
```

10.40 People Tracker on Video Replay

This example shows how to run the people tracker pipeline on a video file.

Note: Visualization in current example is done with blocking behavor. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

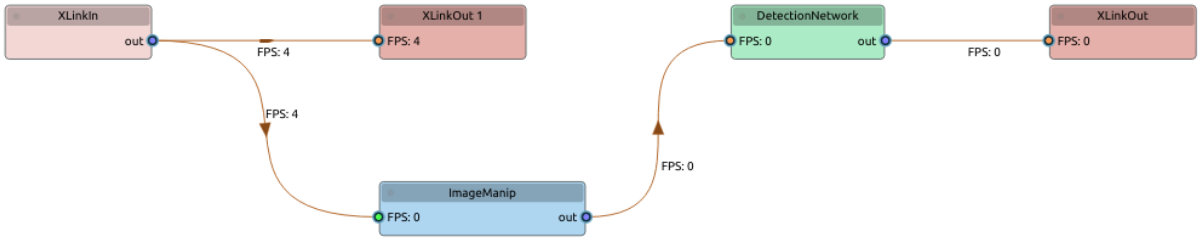
10.40.1 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.40.2 Pipeline



10.40.3 Source Code

Python

Also available on GitHub.

```python
from depthai_sdk import OakCamera, ResizeMode

with OakCamera(replay="people-tracking-above-02") as oak:
    color = oak.create_camera('color')
    nn = oak.create_nn('person-detection-0200', color)
    nn.config_nn(resize_mode=ResizeMode.LETTERBOX)
    oak.visualize([color, nn], fps=True)  # 1080P -> 720P
    oak.start(blocking=True)
```

10.41 Face Detection Inference on Downloaded Image

This example shows how to run the face detection neural network model on a downloaded image from a specified url.

Note: Visualization in current example is done with blocking behavor. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

10.41.1 Demo

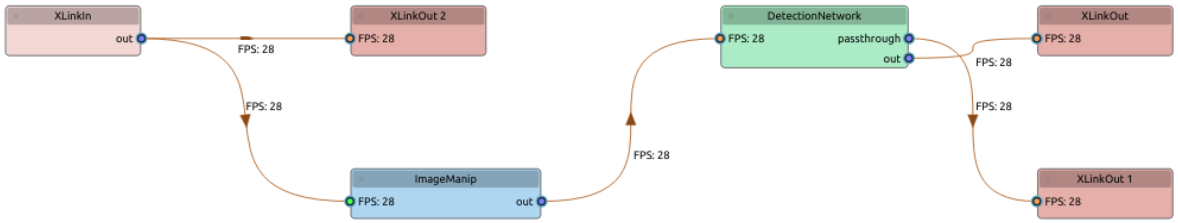
10.41.2 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.41.3 Pipeline



10.41.4 Source Code

Python

Also available on GitHub.

```python
from depthai_sdk import OakCamera

with OakCamera(replay='https://images.pexels.com/photos/3184398/pexels-photo-3184398.jpeg?w=800&h=600') as oak:
    color = oak.create_camera('color')
    nn = oak.create_nn('face-detection-retail-0004', color)
    oak.visualize([nn.out.passthrough, nn])
    oak.start(blocking=True)
```

10.42 Vehicle Detection on a Youtube Video

This example shows how to run the vehicle detection neural network model on a downloaded Youtube video.

Note: Visualization in current example is done with blocking behavior. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

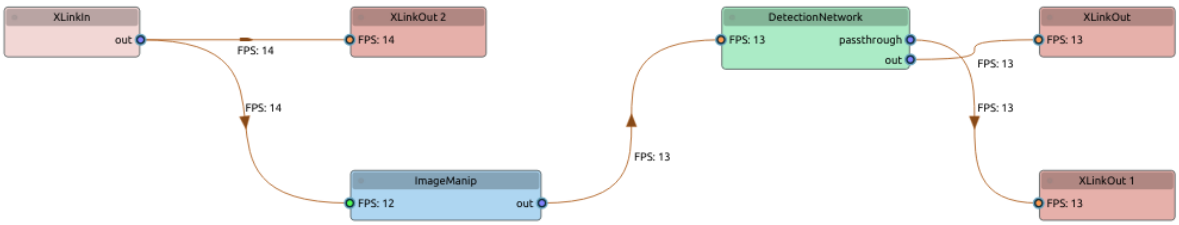
10.42.1 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.42.2 Pipeline



10.42.3 Source Code

Python

Also available on GitHub.

```
from depthai_sdk import OakCamera

with OakCamera(replay='https://www.youtube.com/watch?v=Y1jTEyb3wiI') as oak:
    color = oak.create_camera('color')
    nn = oak.create_nn('vehicle-detection-0202', color)
    oak.visualize([nn.out.passthrough], fps=True)
    oak.visualize(nn, scale=2 / 3, fps=True)
    oak.start(blocking=True)
```

10.43 Stereo Preview

This example shows how to display WLS filtered disparity map using OpenCV.

Note: Visualization in current example is done with blocking behavor. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

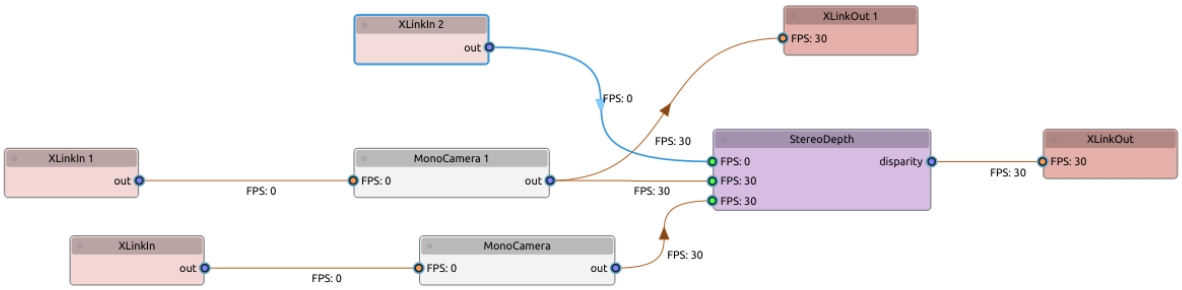
10.43.1 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.43.2 Pipeline



10.43.3 Source Code

Python

Also available on GitHub.

```python
import cv2

from depthai_sdk import OakCamera
from depthai_sdk.components.stereo_component import WLSLevel
from depthai_sdk.visualize.configs import StereoColor

with OakCamera() as oak:
    stereo = oak.create_stereo('800p', fps=30)

    # Configure postprocessing (done on host)
    stereo.config_postprocessing(colorize=StereoColor.RGBD, colormap=cv2.COLORMAP_MAGMA)
    stereo.config_wls(wls_level=WLSLevel.MEDIUM)  # WLS filtering, use for smoother results

    oak.visualize(stereo.out.depth)
    oak.start(blocking=True)
```

10.44 Auto IR Brightness

This example shows how to use the automatic IR brightness feature of the DepthAI Stereo Camera. The function `set_auto_ir(auto_mode=True)` enables/disables auto IR dot projector and flood brightness. If enabled, it selects the best IR brightness level automatically.

Can be set to continious mode, which will continuously adjust the IR brightness. Set to `False` by default and which will automatically adjust the IR brightness only at device bootup.

Note: Visualization in current example is done with blocking behavior. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

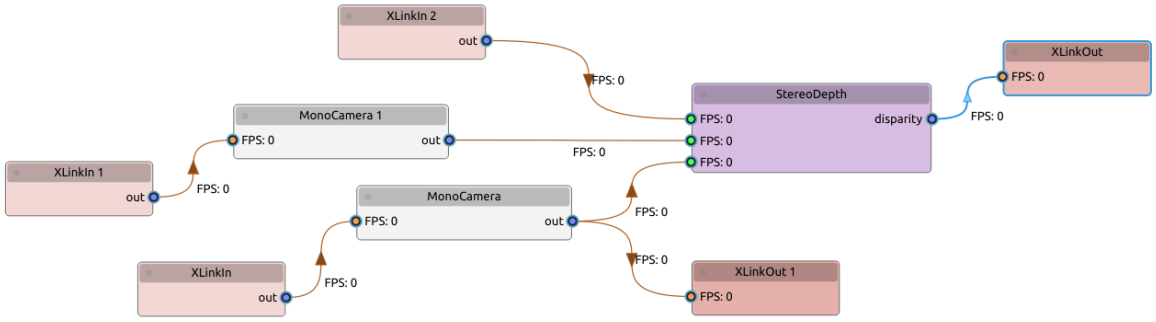
10.44.1 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.44.2 Pipeline



10.44.3 Source Code

Python

Also available on GitHub.

```python
from depthai_sdk import OakCamera

with OakCamera() as oak:
    left = oak.create_camera('left')
    right = oak.create_camera('right')
    stereo = oak.create_stereo(left=left, right=right)

    # Automatically estimate IR brightness and adjust it continuously
    stereo.set_auto_ir(auto_mode=True, continuous_mode=True)

    oak.visualize([stereo.out.disparity, left])
    oak.start(blocking=True)
```

10.45 Stereo Control

This example shows how to change stereo parameter such as confidence threshold, median filter and decimating factor on the fly.

```
Control:                  key[dec/inc]    min..max
Confidence threshold:          I   O      1....255

Switches:
'K' - Switch median filter
'1' - Switch to decimation factor 1
'2' - Switch to decimation factor 2
'3' - Switch to decimation factor 3
```

Note: Visualization in current example is done with blocking behavor. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

10.45.1 Demo

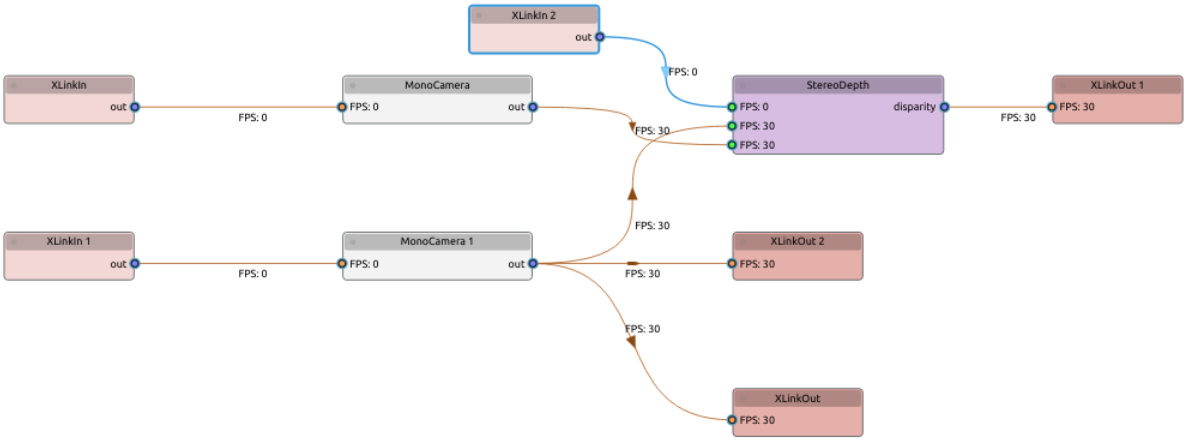
10.45.2 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.45.3 Pipeline



10.45.4 Source Code

Python

Also available on GitHub.

```python
from depthai_sdk import OakCamera

with OakCamera() as oak:
    left = oak.create_camera('left')
    right = oak.create_camera('right')
    stereo = oak.create_stereo(left=left, right=right)
    stereo.config_stereo(lr_check=True)

    oak.visualize([right, stereo.out.disparity], fps=True)
    oak.start()

    while oak.running():
        key = oak.poll()

        if key == ord('i'):
            stereo.control.confidence_threshold_down()
        if key == ord('o'):
            stereo.control.confidence_threshold_up()
        if key == ord('k'):
            stereo.control.switch_median_filter()

        if key == ord('1'):
            stereo.control.send_controls({'postprocessing': {'decimation': {'factor': 1}}})
        if key == ord('2'):
            stereo.control.send_controls({'postprocessing': {'decimation': {'factor': 2}}})
        if key == ord('3'):
            stereo.control.send_controls({'postprocessing': {'decimation': {'factor': 3}}})
```

10.46 Stereo Encoding

This example shows how to encode disparity map and display it using OpenCV.

Note: Visualization in current example is done with blocking behavor. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

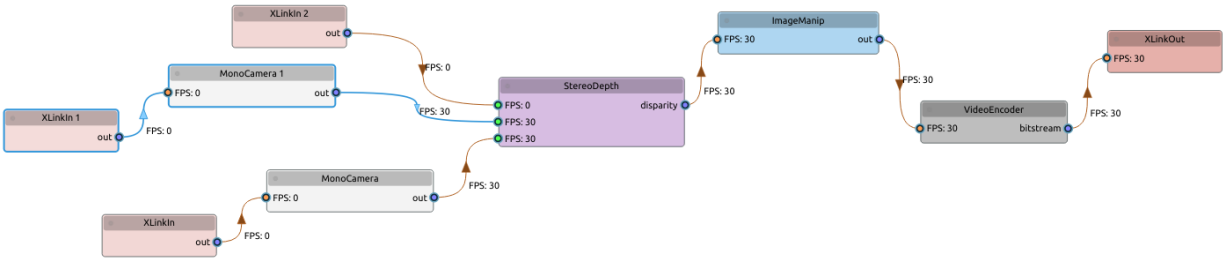
10.46.1 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.46.2 Pipeline



10.46.3 Source Code

Python

Also available on GitHub.

```python
from depthai_sdk import OakCamera
import depthai as dai


with OakCamera() as oak:
    stereo = oak.create_stereo('800p', fps=30, encode='h264')

    # Set on-device output colorization, works only for encoded output
    stereo.set_colormap(dai.Colormap.JET)

    oak.visualize(stereo.out.encoded, fps=True)
    oak.start(blocking=True)
```

10.47 ROS Publishing

This example shows how to use DepthAI SDK to create a ROS Publisher for left, right, color and IMU streams.

Note: Visualization in current example is done with blocking behavor. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

10.47.1 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.47.2 Source Code

Python

Also available on GitHub.

```python
from depthai_sdk import OakCamera

with OakCamera() as oak:
    color = oak.create_camera('color', resolution='1080p', encode='jpeg', fps=30)
    color.config_color_camera(isp_scale=(2,3))
    left = oak.create_camera('left', resolution='400p', encode='jpeg',fps=30)
    right = oak.create_camera('right', resolution='400p', encode='jpeg',fps=30)
    imu = oak.create_imu()
    imu.config_imu(report_rate=400, batch_report_threshold=5)

    oak.ros_stream([left, right, color, imu])
    # oak.visualize(left)
    oak.start(blocking=True)
```

10.48 Custom Trigger Action

This example shows how to set custom action to be triggered when a certain event occurs. In this case, we will trigger an action when a person is detected in the frame. The action will save the exact frame to a file.

Note: Visualization in current example is done with blocking behavor. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

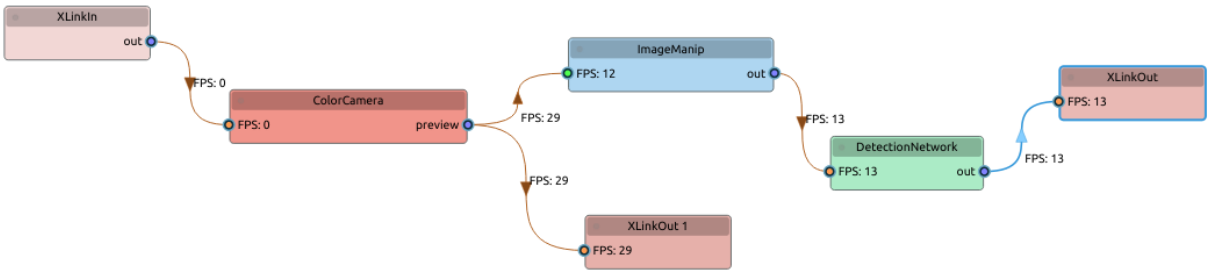
10.48.1 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.48.2 Pipeline



10.48.3 Source Code

Python

Also available on GitHub.

```python
from pathlib import Path
from typing import Dict

import cv2

from depthai_sdk import OakCamera, FramePacket
from depthai_sdk.trigger_action import Action, DetectionTrigger


class MyAction(Action):
    """
    Saves the latest frame from the input stream to a file.
    """

    def __init__(self, inputs, dir_path):
        super().__init__(inputs)

        self.dir_path = Path(dir_path)
        self.dir_path.mkdir(parents=True, exist_ok=True)

        self.latest_packets = None

    def activate(self):
        print('+', self.latest_packets)
```

(continues on next page)

(continued from previous page)

```
        if self.latest_packets:
            for stream_name, packet in self.latest_packets.items():
                print(f'Saving {stream_name} to {self.dir_path / f"{stream_name}.jpg"}
↪')
                cv2.imwrite(str(self.dir_path / f'{stream_name}.jpg'), packet.frame)

    def on_new_packets(self, packets: Dict[str, FramePacket]) -> None:
        self.latest_packets = packets


with OakCamera() as oak:
    color = oak.create_camera('color', '1080p')
    nn = oak.create_nn('mobilenet-ssd', color)

    oak.trigger_action(
        trigger=DetectionTrigger(input=nn, min_detections={'person': 1}, cooldown=30),
        action=MyAction(inputs=[nn], dir_path='./images/')  # `action` can be Callable␣
↪as well
    )

    oak.start(blocking=True)
```

10.49 Custom Trigger

This example shows how to set custom trigger condition in DepthAI SDK. The trigger condition returns a boolean value if the condition is met. In this case the trigger will start a recording of disparity stream when all depth values are below 1 meter.

Note: Visualization in current example is done with blocking behavior. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

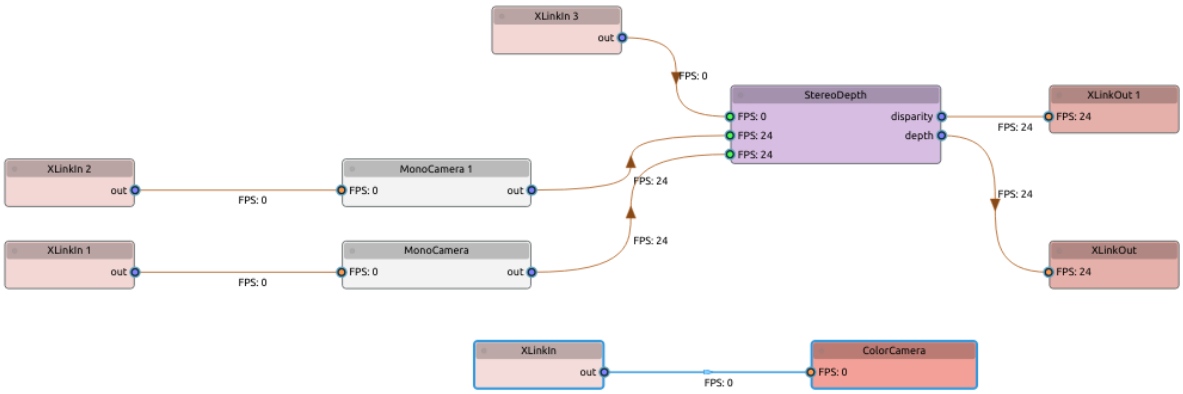
10.49.1 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.49.2 Pipeline



10.49.3 Source Code

Python

Also available on GitHub.

```
import numpy as np

from depthai_sdk import OakCamera
from depthai_sdk.trigger_action import Trigger
from depthai_sdk.trigger_action.actions import RecordAction


def my_condition(packet) -> bool:
    """
    Returns true if all depth values are within 1m range.
    """
    frame = packet.frame
    required_range = 1000  # mm --> 5m

    frame = frame[frame > 0]  # remove invalid depth values
    frame = frame[(frame > np.percentile(frame, 1)) & (frame < np.percentile(frame,
↪99))]

    min_depth = np.min(frame)
    max_depth = np.max(frame)

    if min_depth < required_range < max_depth:
        return True

    return False


with OakCamera() as oak:
    color = oak.create_camera('color', fps=30)
```

(continues on next page)

(continued from previous page)

```
    stereo = oak.create_stereo('800p')
    stereo.config_stereo(align=color)

    trigger = Trigger(input=stereo.out.depth, condition=my_condition, cooldown=30)
    action = RecordAction(
        inputs=[stereo.out.disparity], dir_path='./recordings/',
        duration_before_trigger=5, duration_after_trigger=5
    )

    oak.trigger_action(trigger=trigger, action=action)
    oak.start(blocking=True)
```

10.50 Person Record

This example shows how to set up a trigger with a RecordAction to record both color and disparity frames when a condition is met.

Note: Visualization in current example is done with blocking behavor. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

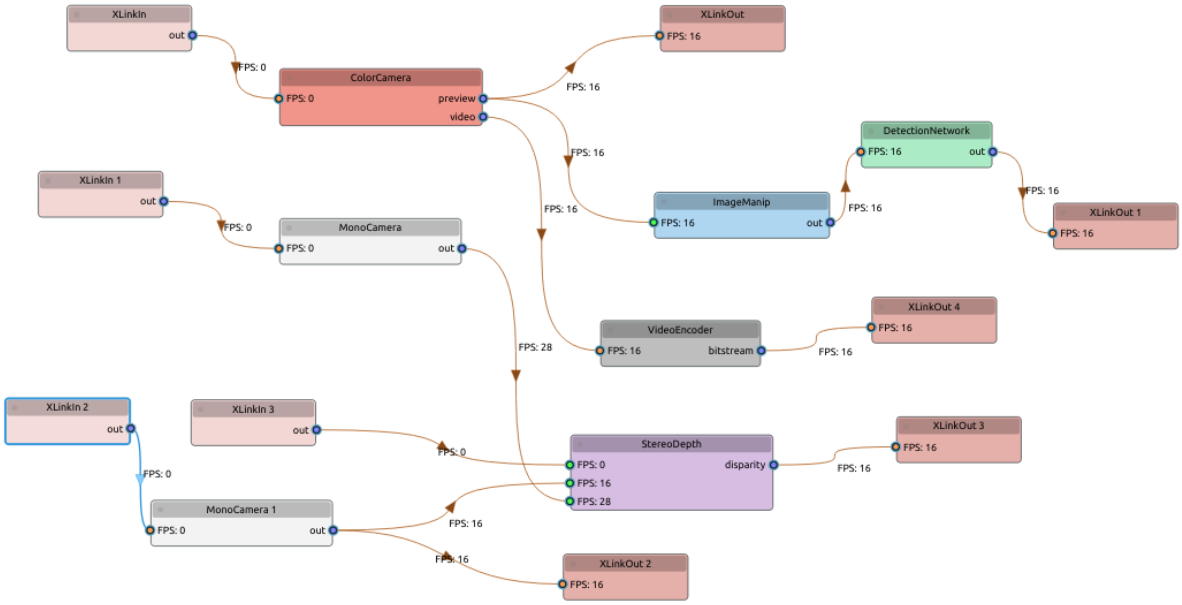
10.50.1 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.50.2 Pipeline



10.50.3 Source Code

Python

Also available on GitHub.

```python
from depthai_sdk import OakCamera
from depthai_sdk.trigger_action.actions.record_action import RecordAction
from depthai_sdk.trigger_action.triggers.detection_trigger import DetectionTrigger

with OakCamera() as oak:
    color = oak.create_camera('color', encode='jpeg')
    stereo = oak.create_stereo('400p')

    nn = oak.create_nn('mobilenet-ssd', color)

    trigger = DetectionTrigger(input=nn, min_detections={'person': 1}, cooldown=30)
    action = RecordAction(inputs=[color, stereo.out.disparity], dir_path='./recordings/',
                          duration_before_trigger=5, duration_after_trigger=10)
    oak.trigger_action(trigger=trigger, action=action)

    oak.visualize(nn)
    oak.start(blocking=True)
```

10.51 Visualizer Demo

This example shows how to use the visualizer component to display the detection results and configure the style of text and tracker.

Note: Visualization in current example is done with blocking behavor. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

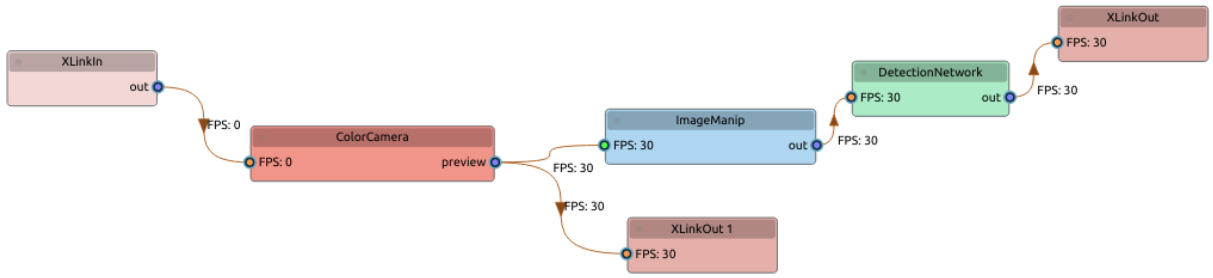
10.51.1 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.51.2 Pipeline



10.51.3 Source Code

Python

Also available on GitHub.

```python
from depthai_sdk import OakCamera
from depthai_sdk.visualize.configs import BboxStyle, TextPosition

with OakCamera() as oak:
    camera = oak.create_camera('color')
    det = oak.create_nn('face-detection-retail-0004', camera)
    # Record visualized video into a mp4 file
    visualizer = oak.visualize(det.out.main, record_path='./test.mp4')
    # Chained methods for setting visualizer parameters
    visualizer.detections(  # Detection-related parameters
        color=(0, 255, 0),
```

(continues on next page)

(continued from previous page)

```
        thickness=2,
        bbox_style=BboxStyle.RECTANGLE,  # Options: RECTANGLE, CORNERS, ROUNDED_
↪RECTANGLE, ROUNDED_CORNERS
        label_position=TextPosition.MID,
    ).text(  # Text-related parameters
        font_color=(255, 255, 0),
        auto_scale=True
    ).output(  # General output parameters
        show_fps=True,
    ).tracking(  # Tracking-related parameters
        line_thickness=5
    )

    oak.start(blocking=True)
```

10.52 Visualizer Callback Function

This example demonstrates the use of a callback function to customize the visualization of detection results.

Note: Visualization in current example is done with blocking behavor. This means that the program will halt at `oak.start()` until the window is closed. This is done to keep the example simple. For more advanced usage, see *Blocking behavior* section.

10.52.1 Demo

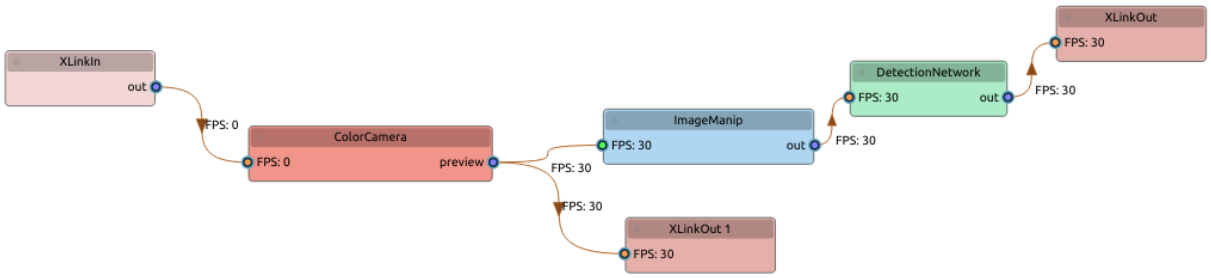
10.52.2 Setup

Please run the install script to download all required dependencies. Please note that this script must be ran from git context, so you have to download the depthai repository first and then run the script

```
git clone https://github.com/luxonis/depthai.git
cd depthai/
python3 install_requirements.py
```

For additional information, please follow our *installation guide*.

10.52.3 Pipeline



10.52.4 Source Code

Python

Also available on GitHub.

```python
import cv2

from depthai_sdk import OakCamera
from depthai_sdk.classes import DetectionPacket
from depthai_sdk.visualize.visualizer_helper import FramePosition, VisualizerHelper


def callback(packet: DetectionPacket):
    visualizer = packet.visualizer
    print('Detections:', packet.img_detections.detections)
    VisualizerHelper.print(packet.frame, 'BottomRight!', FramePosition.BottomRight)
    frame = visualizer.draw(packet.frame)
    cv2.imshow('Visualizer', frame)


with OakCamera() as oak:
    color = oak.create_camera('color')
    nn = oak.create_nn('mobilenet-ssd', color)

    oak.visualize([nn], fps=True, callback=callback)
    oak.start(blocking=True)
```

Code samples are used for automated testing. They are also a great starting point for the DepthAI SDK, as different component functionalities are presented with code.

ColorCamera

- *FFC Camera Visualization* - Preview FFC Cameras
- *Camera Control* - Demonstrates RGB camera control from the host
- *Camera Preview* - Preview color, right, left and depth frames
- *Camera Control with NN* - Control camera (focus, exposure) with NN detections
- *Mono Camera Preview* - Preview mono cameras with manual 400p resolution
- *Preview All Cameras* - Preview all cameras connected to the OAK device
- *RGB and Mono Preview* - Preview RGB and mono cameras
- *Camera Rotated Preview* - Demonstrates how to rotate the camera previews

Mixed

- *API Interoperability Example* - Demonstrates interoperability between the DepthAI API and the SDK
- *Car Tracking Example* - Demonstrates how to run inference on a pre-saved video
- *Collision Avoidance* - Demonstrates how to run collision avoidance
- *Speed Calculation Preview* - Demonstrates how to calculate speed of detected objects in the frame`
- *Switch Between Models* - Demonstrates how to switch between models
- *Sync Multiple Outputs* - Demonstrates how to sync multiple outputs

IMU

- *IMU Demonstration* - Demonstrates how to use and display the IMU
- *IMU Rerun Demonstration* - Demonstrates how use and display the IMU in Rerun

NN

- *Age-Gender Inference* - Demonstrates age-gender inference
- *Custom Decode Function* - Demonstrates custom decoding function
- *Emotion Recognition* - Demonstrates emotion recognition
- *Face Detection RGB* - Run face detection on RGB camera
- *Face Detection Mono* - Run face detection on mono camera
- *Human Pose Estimation* - Run human pose estimation inference
- *MobileNet Encoded* - Pass encoded color stream to the NN (MobileNet)
- *Neural Network Component* - Run color camera stream through NN (YoloV7)
- *Object Tracking* - Tracking objects in the frame
- *Roboflow Integration* - Demonstrates how to use Roboflow platform to train a custom model
- *Spatial Detection* - Perform spatial detection with at MobileNet model
- *YOLO SDK* - Run YoloV3 inference on the color camera stream

Pointcloud

- *Pointcloud Demo* - Preview pointcloud with rerun viewer

Recording

- *Encode Multiple Streams* - Demonstrates how to encode multiple (color, left, right) streams and save them to a file
- *Preview Encoder* - Record color camera stream and save it as mjpeg
- *MCAP Recording* - Record color, left, right and depth streams and save them to a MCAP
- *MCAP IMU Recording* - Record IMU and depth streams and save them to a MCAP
- *Hardcode Recording Duration* - Record color camera stream for a specified duration
- *ROSBAG Recording* - Record IMU, left, right and depth streams and save them to a ROSBAG
- *Stereo Recording* - Records disparity stream

Replay

- *Object counting on images* - Count number of objects on a folder of images (cycle through images every 3 sec)
- *People Tracker on Video Replay* - Run people tracker on a pre-saved video
- *Face Detection Inference on Downloaded Image* - Run face detection on a downloaded image
- *Vehicle Detection on a Youtube Video* - Run vehicle detection on a Youtube video stream
- *Looped Replay* - Replay a pre-saved video in a loop

Stereo

- *Stereo Preview* - Display WLS filtered disparity map
- *Auto IR Brightness* - Demonstrates the use of auto IR brightness function
- *Stereo Control* - Demonstrates stereo control (median filter, decimation factor, confidence threshold) from the host
- *Stereo Encoding* - Demonstrates how to encode stereo stream and visualize it

Streaming

- *ROS Publishing* - Publish color, left, right and IMU streams to ROS

Trigger Action

- *Custom Trigger Action* - Demonstrates how to set a custom trigger action
- *Custom Trigger* - Demonstrates how to set a custom trigger
- *Person Record* - Demonstrates how to record a person when a person is detected

Visualizer

- *Visualizer Demo* - Demonstrates how to use the visualizer
- *Visualizer Callback Function* - Demonstrates how to set the visualizer callback function

CHAPTER

ELEVEN

API REFERENCE

Classes:

ArgsParser()	
ResizeMode(value)	If NN input frame is in different aspect ratio than what the model expect, we have 3 different modes of operation.
OakCamera([device, usb_speed, replay, . . .])	OakCamera improves ease of use when developing apps for OAK devices.
AbstractReader()	
BboxStyle(value)	How do we want to draw bounding box.
CircleConfig([thickness, color, line_type])	Configuration for drawing circles.
DetectionConfig([thickness, . . .])	Configuration for drawing bounding boxes.
FramePacket(name, msg)	Contains only dai.ImgFrame message and cv2 frame, which is used by visualization logic.
IMUPacket(name, packet[, rotation])	
IntEnum(value)	Enum where members are also (and must be) ints
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.
OutputConfig([img_scale, show_fps, clickable])	Configuration for output of the visualizer.
Path(*args, **kwargs)	PurePath subclass that can make system calls.
PreviewDecoder()	
Previews(value)	Enum class, assigning preview name with decode function.
Queue([maxsize])	Create a queue object with a given maximum size.
Record(path, record_type)	This class records depthai streams from OAK cameras into different formats.
RecordType(value)	An enumeration.
Recorder()	
Replay(path)	
ReplayStream()	
SegmentationConfig([mask_alpha])	Configuration for drawing segmentation masks.
StereoColor(value)	An enumeration.
StereoConfig(colorize, colormap, wls_filter, . . .)	
TextConfig([font_face, font_color, . . .])	Configuration for drawing labels.
TextPosition(value)	Where on frame do we want to print text.
Thread([group, target, name, args, kwargs, . . .])	A class that represents a thread of control.
TrackingConfig([max_length, . . .])	Configuration for drawing tracking bounding boxes.

continues on next page

Table 1 – continued from previous page

TurboJPEG([lib_path])	A Python wrapper of libjpeg-turbo for decoding and encoding JPEG image.
VisConfig([output, stereo, detection, text, . . .])	Configuration for visualizer.
Visualizer([scale, fps])	
XoutFrames(frames[, fourcc])	Stream of frames.
partial	partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords.

Functions:

set_logging_level(level)	Set the logging level for the DepthAI SDK logger.
get_config_field(key)	Get sentry status from config file.
cosDist(a, b)	Calculates cosine distance - https://en.wikipedia.org/wiki/Cosine_similarity
createBlankFrame(width, height[, rgb_color])	Create new image(numpy array) filled with certain color in RGB
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
dataclass([cls, init, repr, eq, order, . . .])	Returns the same class as was passed in, with dunder methods added based on the fields defined in the class.
downloadContent(url)	
downloadRecording(name, keys)	
downloadYTVideo(video[, output_dir])	Downloads a video from YouTube and returns the path to video.
field(*[, default, default_factory, init, . . .])	Return an object to identify dataclass fields.
frameNorm(frame, bbox)	Mapps bounding box coordinates (0..1) to pixel values on frame
getAvailableRecordings()	Get available (online) depthai-recordings.
getDeviceInfo([deviceId, debug])	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
getLocalRecording(recording)	
isUrl(source)	
isYoutubeLink(source)	
loadModule(path)	Loads module from specified path.
merge(source, destination)	Utility function to merge two dictionaries
monotonic()	Monotonic clock, cannot go backward.
report_crash_dump(device)	Report crash dump to Sentry if sentry is enabled and crash dump is available.
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
set_sentry_status([status])	Set sentry status in config file.
showProgress(curr, max)	Print progressbar to stdout.

continues on next page

Table 2 – continued from previous page

toPlanar(arr[, shape])	Converts interleaved frame into planar
toTensorResult(packet)	Converts NN packet to dict, with each key being output tensor name and each value being correctly reshaped and converted results array

Exceptions:

CrashDumpException

class depthai_sdk.**ArgsParser**

Bases: *object*

Methods:

parseArgs([parser])	Creates Argument parser for common OAK device configuration

static parseArgs(*parser=None*)

Creates Argument parser for common OAK device configuration

Parameters **parser** (*argparse.ArgumentParser, optional*) – Use an existing parser. By default it creates a new ArgumentParser.

Return type Dict[str, Any]

class depthai_sdk.**ResizeMode**(*value*)

Bases: *enum.IntEnum*

If NN input frame is in different aspect ratio than what the model expect, we have 3 different modes of operation. Full documentation: https://docs.luxonis.com/projects/api/en/latest/tutorials/maximize_fov/

Attributes:

LETTERBOX
STRETCH
CROP
FULL_CROP

Methods:

parse(mode)

LETTERBOX = 0

STRETCH = 1

CROP = 2

FULL_CROP = 3

static parse(*mode*)

Parameters **mode** (*Union[str, depthai_sdk.classes.enum.ResizeMode]*) –

Return type depthai_sdk.classes.enum.ResizeMode

`depthai_sdk.set_logging_level(level)`

Set the logging level for the DepthAI SDK logger.

class `depthai_sdk.OakCamera(device=None, usb_speed=None, replay=None, rotation=0, config=None, args=True)`

Bases: `object`

OakCamera improves ease of use when developing apps for OAK devices.

It abstracts DepthAI API pipeline building, different camera permutations, stream recording/replaying, it adds debugging features, does AI model handling, message syncing & visualization, and much more.

It was designed with interoperability with depthai API in mind.

Methods:

`__init__`([device, usb_speed, replay, ...])	Initializes OakCamera
`camera`(source[, resolution, fps, encode])	Creates Camera component.
`create_camera`(source[, resolution, fps, encode])	Deprecated, use camera() instead.
`all_cameras`([resolution, fps, encode])	Creates Camera component for each camera sensors on the OAK camera.
`create_all_cameras`([resolution, fps, encode])	Deprecated, use all_cameras() instead.
`create_nn`(model, input[, nn_type, tracker, ...])	Creates Neural Network component.
`stereo`([resolution, fps, left, right, encode])	Create Stereo camera component.
`create_stereo`([resolution, fps, left, ...])	Deprecated, use stereo() instead.
`create_imu`()	Create IMU component
`create_pointcloud`([stereo, colorize])	
`config_pipeline`([xlink_chunk, calib, ...])	Configures DepthAI pipeline.
`close`()	
`start`([blocking])	Start the application - upload the pipeline to the OAK device.
`running`()	Check if camera is running.
`poll`()	Poll events; cv2.waitKey, send controls to OAK (if controls are enabled), update, check syncs.
`sync`(outputs, callback[, visualize])	
`record`(outputs, path[, record_type])	Record component outputs.
`show_graph`()	Shows DepthAI Pipeline graph, which can be useful when debugging.
`visualize`(output[, record_path, scale, fps, ...])	Visualize component output(s).
`queue`(output[, max_size])	Create a queue for the component output(s).
`callback`(output, callback[, main_thread])	Create a callback for the component output(s).
`ros_stream`(output)	Publish component output(s) to ROS streams.
`trigger_action`(trigger, action)	

Attributes:

`sensors`	Returns list of all sensors added to the pipeline.

__init__(*device=None, usb_speed=None, replay=None, rotation=0, config=None, args=True*)

Initializes OakCamera

Parameters

- **device** (*str, optional*) – OAK device we want to connect to, either MxId, IP, or

USB port

- **usb_speed** (*str, optional*) – USB speed we want to use. Defaults to 'auto'.
- **replay** (*str, optional*) – Replay a depthai-recording - either local path, or from depthai-recordings repo
- **rotation** (*int, optional*) – Rotate the camera output by this amount of degrees, 0 by default, 90, 180, 270 are supported.
- **args** (*None, bool, Dict*) – Use user defined arguments when constructing the pipeline
- **config** (*Optional[depthai.Device.Config]*) –

camera (*source*, *resolution=None*, *fps=None*, *encode=None*)

Creates Camera component. This abstracts ColorCamera/MonoCamera nodes and supports mocking the camera when recording is passed during OakCamera initialization. Mocking the camera will send frames from the host to the OAK device (via XLinkIn node).

Parameters

- **source** (*str / dai.CameraBoardSocket*) – Camera source
- **resolution** (*str/SensorResolution*) – Sensor resolution of the camera.
- **fps** (*float*) – Sensor FPS
- **encode** (*bool/str/Profile*) – Whether we want to enable video encoding (accessible via cameraComponent.out_encoded). If True, it will use MJPEG

Return type depthai_sdk.components.camera_component.CameraComponent

create_camera (*source*, *resolution=None*, *fps=None*, *encode=None*)

Deprecated, use camera() instead.

Creates Camera component. This abstracts ColorCamera/MonoCamera nodes and supports mocking the camera when recording is passed during OakCamera initialization. Mocking the camera will send frames from the host to the OAK device (via XLinkIn node).

Parameters

- **source** (*str / dai.CameraBoardSocket*) – Camera source
- **resolution** (*str/SensorResolution*) – Sensor resolution of the camera.
- **fps** (*float*) – Sensor FPS
- **encode** (*bool/str/Profile*) – Whether we want to enable video encoding (accessible via cameraComponent.out_encoded). If True, it will use MJPEG

Return type depthai_sdk.components.camera_component.CameraComponent

all_cameras (*resolution=None*, *fps=None*, *encode=None*)

Creates Camera component for each camera sensors on the OAK camera.

Parameters

- **resolution** (*str/SensorResolution*) – Sensor resolution of the camera.
- **fps** (*float*) – Sensor FPS
- **encode** (*bool/str/Profile*) – Whether we want to enable video encoding (accessible via cameraComponent.out_encoded). If True, it will use MJPEG

Return type List[depthai_sdk.components.camera_component.CameraComponent]

create_all_cameras(*resolution=None*, *fps=None*, *encode=None*)

Deprecated, use all_cameras() instead.

Creates Camera component for each camera sensors on the OAK camera.

Parameters

- **resolution** (*str/SensorResolution*) – Sensor resolution of the camera.
- **fps** (*float*) – Sensor FPS
- **encode** (*bool/str/Profile*) – Whether we want to enable video encoding (accessible via cameraComponent.out_encoded). If True, it will use MJPEG

Return type List[depthai_sdk.components.camera_component.CameraComponent]

create_nn(*model*, *input*, *nn_type=None*, *tracker=False*, *spatial=None*, *decode_fn=None*)

Creates Neural Network component.

Parameters

- **model** (*str / Path*) – str for SDK supported model or Path to custom model's json/blob
- **input** (*CameraComponent/NNComponent*) – Input to the model. If NNComponent (detector), it creates 2-stage NN
- **nn_type** (*str*) – Type of the network (yolo/mobilenet) for on-device NN result decoding (only needed if blob path was specified)
- **tracker** (*bool*) – Enable object tracker, if model is object detector (yolo/mobilenet)
- **spatial** (*Union[None, bool, depthai_sdk.components.stereo_component.StereoComponent]*) – Calculate 3D spatial coordinates, if model is object detector (yolo/mobilenet) and depth stream is available
- **decode_fn** (*Optional[Callable]*) – Custom decoding function for the model's output

Return type depthai_sdk.components.nn_component.NNComponent

stereo(*resolution=None*, *fps=None*, *left=None*, *right=None*, *encode=None*)

Create Stereo camera component. If left/right cameras/component aren't specified they will get created internally.

Parameters

- **resolution** (*str/SensorResolution*) – If monochrome cameras aren't already passed, create them and set specified resolution
- **fps** (*float*) – If monochrome cameras aren't already passed, create them and set specified FPS
- **left** (*CameraComponent/dai.node.MonoCamera*) – Pass the camera object (component/node) that will be used for stereo camera.
- **right** (*CameraComponent/dai.node.MonoCamera*) – Pass the camera object (component/node) that will be used for stereo camera.
- **encode** (*bool/str/Profile*) – Whether we want to enable video encoding (accessible via StereoComponent.out.encoded). If True, it will use h264 codec.

Return type depthai_sdk.components.stereo_component.StereoComponent

create_stereo(*resolution=None*, *fps=None*, *left=None*, *right=None*, *name=None*, *encode=None*)

Deprecated, use stereo() instead.

Create Stereo camera component. If left/right cameras/component aren't specified they will get created internally.

Parameters

- **resolution** (*str/SensorResolution*) – If monochrome cameras aren't already passed, create them and set specified resolution
- **fps** (*float*) – If monochrome cameras aren't already passed, create them and set specified FPS
- **left** (*CameraComponent/dai.node.MonoCamera*) – Pass the camera object (component/node) that will be used for stereo camera.
- **right** (*CameraComponent/dai.node.MonoCamera*) – Pass the camera object (component/node) that will be used for stereo camera.
- **encode** (*bool/str/Profile*) – Whether we want to enable video encoding (accessible via StereoComponent.out.encoded). If True, it will use h264 codec.
- **name** (*Optional[str]*) –

Return type depthai_sdk.components.stereo_component.StereoComponent

create_imu()

Create IMU component

Return type depthai_sdk.components.imu_component.IMUComponent

create_pointcloud(*stereo=None*, *colorize=None*)

Parameters

- **stereo** (*Union[None, depthai_sdk.components.stereo_component.StereoComponent, depthai.node.StereoDepth, depthai.Node.Output]*) –
- **colorize** (*Union[None, depthai_sdk.components.camera_component.CameraComponent, depthai.node.MonoCamera, depthai.node.ColorCamera, depthai.Node.Output, bool]*) –

Return type depthai_sdk.components.pointcloud_component.PointcloudComponent

config_pipeline(*xlink_chunk=None*, *calib=None*, *tuning_blob=None*, *openvino_version=None*)

Configures DepthAI pipeline. @param xlink_chunk: Chunk size of XLink messages. 0 can result in lower latency @param calib: Calibration data to be uploaded to OAK @param tuning_blob: Camera tuning blob @param openvino_version: Force specific OpenVINO version

Parameters

- **xlink_chunk** (*Optional[int]*) –
- **calib** (*Optional[depthai.CalibrationHandler]*) –
- **tuning_blob** (*Optional[str]*) –
- **openvino_version** (*Union[None, str, depthai.OpenVINO.Version]*) –

close()

start (*blocking=False*)
Start the application - upload the pipeline to the OAK device. :param blocking: Continuously loop and call oak.poll() until program exits :type blocking: bool

running ()
Check if camera is running. :returns: True if camera is running, False otherwise.

Return type bool

poll ()
Poll events; cv2.waitKey, send controls to OAK (if controls are enabled), update, check syncs.

Returns: key pressed from cv2.waitKey, or None if

Return type Optional[int]

sync (*outputs*, *callback*, *visualize=False*)

Parameters

- **outputs** (*Union[depthai_sdk.components.component.ComponentOutput, List[depthai_sdk.components.component.ComponentOutput]]*) –
- **callback** (*Callable*) –

record (*outputs*, *path*, *record_type=<RecordType.VIDEO: 1>*)
Record component outputs. This handles syncing multiple streams (eg. left, right, color, depth) and saving them to the computer in desired format (raw, mp4, mcap, bag..).

Parameters

- **outputs** (*Component/Component output*) – Component output(s) to be recorded.
- **path** (*str*) – Folder path where to save these streams.
- **record_type** (*depthai_sdk.record.RecordType*) – Record type.

Return type depthai_sdk.classes.packet_handlers.RecordPacketHandler

show_graph ()
Shows DepthAI Pipeline graph, which can be useful when debugging. You must call this BEFORE calling the oak.start()!

visualize (*output*, *record_path=None*, *scale=None*, *fps=False*, *callback=None*, *visualizer='opencv'*)
Visualize component output(s). This handles output streaming (OAK->host), message syncing, and visualizing.

Parameters

- **output** (*Component/Component output*) – Component output(s) to be visualized. If component is passed, SDK will visualize its default output (out()).
- **record_path** (*Optional[str]*) – Path where to store the recording (visualization window name gets appended to that path), supported formats: mp4, avi.
- **scale** (*Optional[float]*) – Scale the output window by this factor.
- **fps** – Whether to show FPS on the output window.
- **callback** (*Optional[Callable]*) – Instead of showing the frame, pass the Packet to the callback function, where it can be displayed.
- **visualizer** (*str*) – Which visualizer to use. Options: 'opencv', 'depthai-viewer', 'robothub'.

Return type *depthai_sdk.visualize.visualizer.Visualizer*

queue(*output*, *max_size=30*)

Create a queue for the component output(s). This handles output streaming (OAK->Host) and message syncing.

Parameters

- **output** (*Union[depthai_sdk.components.component.ComponentOutput, depthai_sdk.components.component.Component, List]*) – Component output(s) to be visualized. If component is passed, SDK will visualize its default output.
- **max_size** (*int*) – Maximum queue size for this output.

Return type depthai_sdk.classes.packet_handlers.QueuePacketHandler

callback(*output*, *callback*, *main_thread=False*)

Create a callback for the component output(s). This handles output streaming (OAK->Host) and message syncing.

Parameters

- **output** (*Union[List, Callable, depthai_sdk.components.component.Component]*) – Component output(s) to be visualized. If component is passed, SDK will visualize its default output.
- **callback** (*Callable*) – Handler function to which the Packet will be sent.
- **main_thread** – Whether to run the callback in the main thread. If False, it will call the callback in a separate thread, so some functions (eg. cv2.imshow) won't work.

Return type depthai_sdk.classes.packet_handlers.CallbackPacketHandler

ros_stream(*output*)

Publish component output(s) to ROS streams.

Parameters **output** (*Union[List, depthai_sdk.components.component.ComponentOutput, depthai_sdk.components.component.Component]*) –

Return type depthai_sdk.classes.packet_handlers.RosPacketHandler

trigger_action(*trigger*, *action*)

Parameters

- **trigger** (*depthai_sdk.trigger_action.triggers.abstract_trigger.Trigger*) –
- **action** (*Union[depthai_sdk.trigger_action.actions.abstract_action.Action, Callable]*) –

Return type None

property sensors

Returns list of all sensors added to the pipeline.

depthai_sdk.**get_config_field**(*key*)

Get sentry status from config file.

Returns True if sentry is enabled, False otherwise.

Return type bool

Parameters **key** (*str*) –

class depthai_sdk.**AbstractReader**

Bases: abc.ABC

Methods:

read()	Read a frame (or multiple frames) from the reader.
getStreams()	
getShape(name)	Returns (width, height)
get_socket(name)	Returns socket
get_message_size(name)	@param name: Stream name @return: Number of bytes for that this message contains
close()	
disableStream(name)	@param name: Name of the stream to be disabled

abstract read()

Read a frame (or multiple frames) from the reader. @return: Single np.ndarray, or dict of frames and their names. None if frames weren't read or there was an error.

Return type Dict[str, numpy.ndarray]

abstract getStreams()

Return type List[str]

abstract getShape(*name*)

Returns (width, height)

Parameters **name** (*str*) –

Return type Tuple[int, int]

get_socket(*name*)

Returns socket

Parameters **name** (*str*) –

abstract get_message_size(*name*)

@param name: Stream name @return: Number of bytes for that this message contains

Parameters **name** (*str*) –

Return type int

abstract close()

abstract disableStream(*name*)

@param name: Name of the stream to be disabled

Parameters **name** (*str*) –

class depthai_sdk.**BboxStyle**(*value*)

Bases: enum.IntEnum

How do we want to draw bounding box.

Attributes:

RECTANGLE
CORNERS
ROUNDED_RECTANGLE
ROUNDED_CORNERS

RECTANGLE = 0

CORNERS = 1

ROUNDED_RECTANGLE = 10

ROUNDED_CORNERS = 11

class depthai_sdk.**CircleConfig**(*thickness=1*, *color=(255, 255, 255)*, *line_type=16*)

Bases: object

Configuration for drawing circles.

Attributes:

thickness
color
line_type

Methods:

__init__([thickness, color, line_type])	Initialize self.

thickness: int = 1

color: Tuple[int, int, int] = (255, 255, 255)

line_type: int = 16

__init__(*thickness=1*, *color=(255, 255, 255)*, *line_type=16*)

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

Parameters

- **thickness** (*int*) –
- **color** (*Tuple[int, int, int]*) –
- **line_type** (*int*) –

Return type None

exception depthai_sdk.**CrashDumpException**

Bases: Exception

class depthai_sdk.**DetectionConfig**(*thickness=1*, *fill_transparency=0.15*, *box_roundness=0*, *color=(0, 255, 0)*, *bbox_style=<BboxStyle.RECTANGLE: 0>*, *line_width=0.5*, *line_height=0.5*, *hide_label=False*, *label_position=<TextPosition.TOP_LEFT: 0>*, *label_padding=10*)

Bases: object

Configuration for drawing bounding boxes.

Attributes:

thickness
fill_transparency
box_roundness
color
bbox_style

continues on next page

Table 13 – continued from previous page

line_width
line_height
hide_label
label_position
label_padding

Methods:

__init__([thickness, fill_transparency, . . .])	Initialize self.

thickness: int = 1

fill_transparency: float = 0.15

box_roundness: int = 0

color: Tuple[int, int, int] = (0, 255, 0)

bbox_style: *depthai_sdk.visualize.configs.BboxStyle* = 0

line_width: float = 0.5

line_height: float = 0.5

hide_label: bool = False

label_position: *depthai_sdk.visualize.configs.TextPosition* = 0

label_padding: int = 10

__init__(*thickness=1, fill_transparency=0.15, box_roundness=0, color=(0, 255, 0), bbox_style=<BboxStyle.RECTANGLE: 0>, line_width=0.5, line_height=0.5, hide_label=False, label_position=<TextPosition.TOP_LEFT: 0>, label_padding=10*)

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

Parameters

- **thickness** (*int*) –
- **fill_transparency** (*float*) –
- **box_roundness** (*int*) –
- **color** (*Tuple[int, int, int]*) –
- **bbox_style** (depthai_sdk.visualize.configs.BboxStyle) –
- **line_width** (*float*) –
- **line_height** (*float*) –
- **hide_label** (*bool*) –
- **label_position** (depthai_sdk.visualize.configs.TextPosition) –
- **label_padding** (*int*) –

Return type None

class depthai_sdk.**FramePacket**(*name, msg*)

Bases: depthai_sdk.classes.packets.BasePacket

Contains only dai.ImgFrame message and cv2 frame, which is used by visualization logic.

Methods:

__init__(name, msg)	Initialize self.
get_timestamp()	
get_sequence_num()	
set_decode_codec(get_codec)	
decode()	
get_size()	

Attributes:

frame

__init__(*name*, *msg*)

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

Parameters

- **name** (*str*) –
- **msg** (*depthai.ImgFrame*) –

property frame

get_timestamp()

Return type datetime.timedelta

get_sequence_num()

Return type int

set_decode_codec(*get_codec*)

Parameters **get_codec** (*Callable*) –

decode()

Return type Optional[numpy.ndarray]

get_size()

Return type Tuple[int, int]

class depthai_sdk.**IMUPacket**(*name*, *packet*, *rotation=None*)

Bases: depthai_sdk.classes.packets.BasePacket

Methods:

__init__(name, packet[, rotation])	Initialize self.
get_imu_vals()	Returns imu values in a tuple.
get_timestamp()	
get_sequence_num()	

__init__(*name*, *packet*, *rotation=None*)

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

Parameters **packet** (*depthai.IMUPacket*) –

get_imu_vals()

Returns imu values in a tuple. Returns in format (accelerometer_values, gyroscope_values, quaternion, magnetometer_values)

Return type Tuple[Sequence, Sequence, Sequence, Sequence]

get_timestamp()

Return type datetime.timedelta

get_sequence_num()

Return type int

class depthai_sdk.**IntEnum**(*value*)

Bases: int, enum.Enum

Enum where members are also (and must be) ints

class depthai_sdk.**MouseClickTracker**

Bases: object

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

Attributes:

points	Stores selected point position per frame
values	Stores values assigned to specific point per frame

Methods:

selectPoint(name)	Returns callback function for cv2.setMouseCallback that will update the selected point on mouse click event from frame.
extractValue(name, frame)	Extracts value from frame for a specific point

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** (*numpy.ndarray*) –

class depthai_sdk.**OutputConfig**(*img_scale=1.0*, *show_fps=False*, *clickable=True*)

Bases: object

Configuration for output of the visualizer.

Attributes:

img_scale	
show_fps	
clickable	

Methods:

__init__([img_scale, show_fps, clickable])	Initialize self.

img_scale: float = 1.0

show_fps: bool = False

clickable: bool = True

__init__(*img_scale=1.0*, *show_fps=False*, *clickable=True*)

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

Parameters

- **img_scale** (*float*) –
- **show_fps** (*bool*) –
- **clickable** (*bool*) –

Return type None

class depthai_sdk.**Path**(**args*, ***kwargs*)

Bases: pathlib.PurePath

PurePath subclass that can make system calls.

Path represents a filesystem path but unlike PurePath, also offers methods to do system calls on path objects. Depending on your system, instantiating a Path will return either a PosixPath or a WindowsPath object. You can also instantiate a PosixPath or WindowsPath directly, but cannot instantiate a WindowsPath on a POSIX system or vice versa.

Methods:

cwd()	Return a new path pointing to the current working directory (as returned by os.getcwd()).
home()	Return a new path pointing to the user's home directory (as returned by os.path.expanduser('~')).
samefile(other_path)	Return whether other_path is the same or not as this file (as returned by os.path.samefile()).
iterdir()	Iterate over the files in this directory.

continues on next page

Table 22 – continued from previous page

glob(pattern)	Iterate over this subtree and yield all existing files (of any kind, including directories) matching the given relative pattern.
rglob(pattern)	Recursively yield all existing files (of any kind, including directories) matching the given relative pattern, anywhere in this subtree.
absolute()	Return an absolute version of this path.
resolve([strict])	Make the path absolute, resolving all symlinks on the way and also normalizing it (for example turning slashes into backslashes under Windows).
stat()	Return the result of the stat() system call on this path, like os.stat() does.
owner()	Return the login name of the file owner.
group()	Return the group name of the file gid.
open([mode, buffering, encoding, errors, . . .])	Open the file pointed by this path and return a file object, as the built-in open() function does.
read_bytes()	Open the file in bytes mode, read it, and close the file.
read_text([encoding, errors])	Open the file in text mode, read it, and close the file.
write_bytes(data)	Open the file in bytes mode, write to it, and close the file.
write_text(data[, encoding, errors])	Open the file in text mode, write to it, and close the file.
touch([mode, exist_ok])	Create this file with the given access mode, if it doesn't exist.
mkdir([mode, parents, exist_ok])	Create a new directory at this given path.
chmod(mode)	Change the permissions of the path, like os.chmod().
lchmod(mode)	Like chmod(), except if the path points to a symlink, the symlink's permissions are changed, rather than its target's.
unlink([missing_ok])	Remove this file or link.
rmdir()	Remove this directory.
lstat()	Like stat(), except if the path points to a symlink, the symlink's status information is returned, rather than its target's.
rename(target)	Rename this path to the target path.
replace(target)	Rename this path to the target path, overwriting if that path exists.
symlink_to(target[, target_is_directory])	Make this path a symlink pointing to the target path.
link_to(target)	Make the target path a hard link pointing to this path.
exists()	Whether this path exists.
is_dir()	Whether this path is a directory.
is_file()	Whether this path is a regular file (also True for symlinks pointing to regular files).
is_mount()	Check if this path is a POSIX mount point
is_symlink()	Whether this path is a symbolic link.
is_block_device()	Whether this path is a block device.
is_char_device()	Whether this path is a character device.
is_fifo()	Whether this path is a FIFO.
is_socket()	Whether this path is a socket.

continues on next page

Table 22 – continued from previous page

expanduser()	Return a new path with expanded ~ and ~user constructs (as returned by os.path.expanduser)

classmethod cwd()
Return a new path pointing to the current working directory (as returned by os.getcwd()).

classmethod home()
Return a new path pointing to the user's home directory (as returned by os.path.expanduser('~')).

samefile(*other_path*)
Return whether other_path is the same or not as this file (as returned by os.path.samefile()).

iterdir()
Iterate over the files in this directory. Does not yield any result for the special paths '.' and '..'.

glob(*pattern*)
Iterate over this subtree and yield all existing files (of any kind, including directories) matching the given relative pattern.

rglob(*pattern*)
Recursively yield all existing files (of any kind, including directories) matching the given relative pattern, anywhere in this subtree.

absolute()
Return an absolute version of this path. This function works even if the path doesn't point to anything.

No normalization is done, i.e. all '.' and '..' will be kept along. Use resolve() to get the canonical path to a file.

resolve(*strict=False*)
Make the path absolute, resolving all symlinks on the way and also normalizing it (for example turning slashes into backslashes under Windows).

stat()
Return the result of the stat() system call on this path, like os.stat() does.

owner()
Return the login name of the file owner.

group()
Return the group name of the file gid.

open(*mode='r', buffering=- 1, encoding=None, errors=None, newline=None*)
Open the file pointed by this path and return a file object, as the built-in open() function does.

read_bytes()
Open the file in bytes mode, read it, and close the file.

read_text(*encoding=None, errors=None*)
Open the file in text mode, read it, and close the file.

write_bytes(*data*)
Open the file in bytes mode, write to it, and close the file.

write_text(*data, encoding=None, errors=None*)
Open the file in text mode, write to it, and close the file.

touch(*mode=438, exist_ok=True*)
Create this file with the given access mode, if it doesn't exist.

mkdir(*mode=511, parents=False, exist_ok=False*)
Create a new directory at this given path.

chmod(*mode*)
Change the permissions of the path, like os.chmod().

lchmod(*mode*)
Like chmod(), except if the path points to a symlink, the symlink's permissions are changed, rather than its target's.

unlink(*missing_ok=False*)
Remove this file or link. If the path is a directory, use rmdir() instead.

rmdir()
Remove this directory. The directory must be empty.

lstat()
Like stat(), except if the path points to a symlink, the symlink's status information is returned, rather than its target's.

rename(*target*)
Rename this path to the target path.

The target path may be absolute or relative. Relative paths are interpreted relative to the current working directory, *not* the directory of the Path object.

Returns the new Path instance pointing to the target path.

replace(*target*)
Rename this path to the target path, overwriting if that path exists.

The target path may be absolute or relative. Relative paths are interpreted relative to the current working directory, *not* the directory of the Path object.

Returns the new Path instance pointing to the target path.

symlink_to(*target*, *target_is_directory=False*)
Make this path a symlink pointing to the target path. Note the order of arguments (link, target) is the reverse of os.symlink.

link_to(*target*)
Make the target path a hard link pointing to this path.

Note this function does not make this path a hard link to *target*, despite the implication of the function and argument names. The order of arguments (target, link) is the reverse of Path.symlink_to, but matches that of os.link.

exists()
Whether this path exists.

is_dir()
Whether this path is a directory.

is_file()
Whether this path is a regular file (also True for symlinks pointing to regular files).

is_mount()
Check if this path is a POSIX mount point

is_symlink()
Whether this path is a symbolic link.

is_block_device()
Whether this path is a block device.

is_char_device()
Whether this path is a character device.

is_fifo()
Whether this path is a FIFO.

is_socket()
Whether this path is a socket.

expanduser()
Return a new path with expanded ~ and ~user constructs (as returned by os.path.expanduser)

class depthai_sdk.**PreviewDecoder**
Bases: object

Methods:

jpegDecode(data, type)	
nnInput(packet[, manager])	Produces NN passthough frame from raw data packet
color(packet[, manager])	Produces color camera frame from raw data packet
left(packet[, manager])	Produces left camera frame from raw data packet
right(packet[, manager])	Produces right camera frame from raw data packet
rectifiedLeft(packet[, manager])	Produces rectified left frame (depthai.node.StereoDepth.rectifiedLeft) from raw data packet
rectifiedRight(packet[, manager])	Produces rectified right frame (depthai.node.StereoDepth.rectifiedRight) from raw data packet
depthRaw(packet[, manager])	Produces raw depth frame (depthai.node.StereoDepth.depth) from raw data packet
depth(depthRaw[, manager])	Produces depth frame from raw depth frame (converts to disparity and applies color map)
disparity(packet[, manager])	Produces disparity frame (depthai.node.StereoDepth.disparity) from raw data packet
disparityColor(disparity[, manager])	Applies color map to disparity frame

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**(*value*)

Bases: `enum.Enum`

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

Attributes:

nnInput(*args, **kwargs)	
color(*args, **kwargs)	
left(*args, **kwargs)	
right(*args, **kwargs)	

continues on next page

Table 24 – continued from previous page

rectifiedLeft(*args, **kwargs)	
rectifiedRight(*args, **kwargs)	
depthRaw(*args, **kwargs)	
depth(*args, **kwargs)	
disparity(*args, **kwargs)	
disparityColor(*args, **kwargs)	

nnInput(**args, **kwargs*) **= functools.partial(<function PreviewDecoder.nnInput>)**

color(**args, **kwargs*) **= functools.partial(<function PreviewDecoder.color>)**

left(**args, **kwargs*) **= functools.partial(<function PreviewDecoder.left>)**

right(**args, **kwargs*) **= functools.partial(<function PreviewDecoder.right>)**

rectifiedLeft(**args, **kwargs*) **= functools.partial(<function PreviewDecoder.rectifiedLeft>)**

rectifiedRight(**args, **kwargs*) **= functools.partial(<function PreviewDecoder.rectifiedRight>)**

depthRaw(**args, **kwargs*) **= functools.partial(<function PreviewDecoder.depthRaw>)**

depth(**args, **kwargs*) **= functools.partial(<function PreviewDecoder.depth>)**

disparity(**args, **kwargs*) **= functools.partial(<function PreviewDecoder.disparity>)**

disparityColor(**args, **kwargs*) **= functools.partial(<function PreviewDecoder.disparityColor>)**

class depthai_sdk.**Queue**(*maxsize=0*)

Bases: object

Create a queue object with a given maximum size.

If maxsize is <= 0, the queue size is infinite.

Methods:

__init__([maxsize])	Initialize self.
task_done()	Indicate that a formerly enqueued task is complete.
join()	Blocks until all items in the Queue have been gotten and processed.
qsize()	Return the approximate size of the queue (not reliable!).
empty()	Return True if the queue is empty, False otherwise (not reliable!).
full()	Return True if the queue is full, False otherwise (not reliable!).
put(item[, block, timeout])	Put an item into the queue.
get([block, timeout])	Remove and return an item from the queue.
put_nowait(item)	Put an item into the queue without blocking.
get_nowait()	Remove and return an item from the queue without blocking.

__init__(*maxsize=0*)
Initialize self. See help(type(self)) for accurate signature.

task_done()
Indicate that a formerly enqueued task is complete.

Used by Queue consumer threads. For each get() used to fetch a task, a subsequent call to task_done() tells the queue that the processing on the task is complete.

If a join() is currently blocking, it will resume when all items have been processed (meaning that a task_done() call was received for every item that had been put() into the queue).

Raises a ValueError if called more times than there were items placed in the queue.

join()
Blocks until all items in the Queue have been gotten and processed.

The count of unfinished tasks goes up whenever an item is added to the queue. The count goes down whenever a consumer thread calls task_done() to indicate the item was retrieved and all work on it is complete.

When the count of unfinished tasks drops to zero, join() unblocks.

qsize()
Return the approximate size of the queue (not reliable!).

empty()
Return True if the queue is empty, False otherwise (not reliable!).

This method is likely to be removed at some point. Use qsize() == 0 as a direct substitute, but be aware that either approach risks a race condition where a queue can grow before the result of empty() or qsize() can be used.

To create code that needs to wait for all queued tasks to be completed, the preferred technique is to use the join() method.

full()
Return True if the queue is full, False otherwise (not reliable!).

This method is likely to be removed at some point. Use qsize() >= n as a direct substitute, but be aware that either approach risks a race condition where a queue can shrink before the result of full() or qsize() can be used.

put(*item*, *block=True*, *timeout=None*)
Put an item into the queue.

If optional args 'block' is true and 'timeout' is None (the default), block if necessary until a free slot is available. If 'timeout' is a non-negative number, it blocks at most 'timeout' seconds and raises the Full exception if no free slot was available within that time. Otherwise ('block' is false), put an item on the queue if a free slot is immediately available, else raise the Full exception ('timeout' is ignored in that case).

get(*block=True*, *timeout=None*)
Remove and return an item from the queue.

If optional args 'block' is true and 'timeout' is None (the default), block if necessary until an item is available. If 'timeout' is a non-negative number, it blocks at most 'timeout' seconds and raises the Empty exception if no item was available within that time. Otherwise ('block' is false), return an item if one is immediately available, else raise the Empty exception ('timeout' is ignored in that case).

put_nowait(*item*)
Put an item into the queue without blocking.

Only enqueue the item if a free slot is immediately available. Otherwise raise the Full exception.

get_nowait()

Remove and return an item from the queue without blocking.

Only get an item if one is immediately available. Otherwise raise the Empty exception.

class depthai_sdk.**Record**(*path*, *record_type*)

Bases: object

This class records depthai streams from OAK cameras into different formats. It will also save calibration .json, so depth reconstruction will be possible.

Methods:

__init__(path, record_type)	**param path** Path to the recording folder
write(packets)	
start(device, xouts)	Start recording process.
config_mcap(pointcloud)	
close()	

__init__(*path*, *record_type*)

Parameters

- **path** (Path) – Path to the recording folder
- **record_type** (RecordType) – Recording type

write(*packets*)

start(*device*, *xouts*)

Start recording process. This will create and start the pipeline, start recording threads, and initialize all queues.

Parameters

- **device** (*depthai.Device*) –
- **xouts** (*List[depthai_sdk.oak_outputs.xout.xout_frames.XoutFrames]*) –

config_mcap(*pointcloud*)

Parameters **pointcloud** (*bool*) –

close()

class depthai_sdk.**RecordType**(*value*)

Bases: enum.IntEnum

An enumeration.

Attributes:

VIDEO
VIDEO_LOSSLESS
ROSBAG
MCAP
DB3

VIDEO = 1

VIDEO_LOSSLESS = 2

ROSBAG = 3

MCAP = 4

DB3 = 5

class depthai_sdk.**Recorder**

Bases: abc.ABC

Methods:

write(name, frame)	
close()	
update(path, device, xouts)	

abstract write(*name*, *frame*)

Parameters

- **name** (*str*) –
- **frame** (*depthai.ImgFrame*) –

abstract close()

abstract update(*path*, *device*, *xouts*)

Parameters

- **path** (*pathlib.Path*) –
- **device** (*depthai.Device*) –
- **xouts** (*List[XoutFrames]*) –

class depthai_sdk.**Replay**(*path*)

Bases: object

Methods:

__init__(path)	Helper file to replay recorded depthai stream.
toggle_pause()	Toggle pausing of sending frames to the OAK camera.
set_fps(fps)	Sets frequency at which Replay module will send frames to the camera.
set_loop(flag)	Sets whether to loop the replay.
get_fps()	
resize(stream_name, size[, mode])	Resize color frames prior to sending them to the device.
keepAspectRatio(keepAspectRatio)	
disableStream(stream_name[, disableReading])	Disable sending a recorded stream to the device.
specify_socket(stream_name, socket)	
initPipeline([pipeline])	Prepares the pipeline for replaying.
initStereoDepth(stereo[, left_name, . . .])	
start()	Start sending frames to the OAK device on a new thread

continues on next page

Table 29 – continued from previous page

`sendFrames`()	Reads and sends recorded frames from all enabled streams to the OAK camera.
`createQueues`(device)	Creates input queue for each enabled stream
`getStreams`()	
`getShape`(name)	Get shape of a stream
`close`()	Closes all video readers.

__init__(*path*)

Helper file to replay recorded depthai stream. It reads from recorded files (mjpeg/avi/mp4/h265/h264/bag) and sends frames back to OAK camera to replay the scene, including depth reconstruction from 2 synced mono streams.

Parameters **path** (`str`) – Path to the recording folder.

toggle_pause()

Toggle pausing of sending frames to the OAK camera.

set_fps(*fps*)

Sets frequency at which Replay module will send frames to the camera. Default 30FPS.

Parameters **fps** (`float`) –

set_loop(*flag*)

Sets whether to loop the replay.

Parameters **flag** (`bool`) – Whether to loop the replay.

get_fps()

Return type float

resize(*stream_name*, *size*, *mode=<ResizeMode.STRETCH: 1>*)

Resize color frames prior to sending them to the device.

Parameters

- **stream_name** (`str`) – Name of the stream we want to resize
- **size** (`Tuple(width, heigth)`) – Size of color frames that are sent to the camera
- **mode** (`ResizeMode`) – How to actually resize the stream

keepAspectRatio(*keepAspectRatio*)

Parameters **keepAspectRatio** (`bool`) –

disableStream(*stream_name*, *disableReading=False*)

Disable sending a recorded stream to the device.

Parameters

- **streamName** (`str`) – Name of the stream to disable (eg. 'left', 'color', 'depth', etc.)
- **disableReading** (`bool, Optional`) – Also disable reading frames from the file
- **stream_name** (`str`) –

specify_socket(*stream_name*, *socket*)

Parameters

- **stream_name** (`str`) –
- **socket** (`depthai.CameraBoardSocket`) –

initPipeline(*pipeline=None*)
Prepares the pipeline for replaying. It creates XLinkIn nodes and sets up StereoDepth node. Returns: dai.Pipeline

Parameters **pipeline** (*Optional[depthai.Pipeline]*) –

initStereoDepth(*stereo*, *left_name='left'*, *right_name='right'*, *align_to=''*)

Parameters

- **stereo** (*depthai.node.StereoDepth*) –
- **left_name** (*str*) –
- **right_name** (*str*) –
- **align_to** (*str*) –

start()
Start sending frames to the OAK device on a new thread

sendFrames()
Reads and sends recorded frames from all enabled streams to the OAK camera.

Returns True if successful, otherwise False.

Return type bool

createQueues(*device*)
Creates input queue for each enabled stream

Parameters **device** (*dai.Device*) – Device to which we will stream frames

getStreams()

Return type List[str]

getShape(*name*)
Get shape of a stream

Parameters **name** (*str*) –

Return type Tuple[int, int]

close()
Closes all video readers.

class depthai_sdk.**ReplayStream**
Bases: object

Attributes:

shape	

Methods:

__init__()	Initialize self.
get_socket()	

property shape

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

get_socket()

Return type depthai.CameraBoardSocket

class depthai_sdk.**SegmentationConfig**(*mask_alpha=0.5*)

Bases: object

Configuration for drawing segmentation masks.

Attributes:

mask_alpha

Methods:

__init__([mask_alpha])	Initialize self.

mask_alpha: float = 0.5

__init__(*mask_alpha=0.5*)

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

Parameters **mask_alpha** (*float*) –

Return type None

class depthai_sdk.**StereoColor**(*value*)

Bases: enum.IntEnum

An enumeration.

Attributes:

GRAY
RGB
RGBD

GRAY = 1

RGB = 2

RGBD = 3

class depthai_sdk.**StereoConfig**(*colorize: depthai_sdk.visualize.configs.StereoColor = <StereoColor.RGB: 2>, colormap: numpy.ndarray = <factory>, wls_filter: bool = False, wls_lambda: float = 8000, wls_sigma: float = 1.5*)

Bases: object

Attributes:

colorize
wls_filter
wls_lambda
wls_sigma

Methods:

__init__([colorize, colormap, wls_filter, . . .])	Initialize self.

colorize: *depthai_sdk.visualize.configs.StereoColor* **= 2**

colormap: numpy.ndarray

wls_filter: bool **= False**

wls_lambda: float **= 8000**

wls_sigma: float **= 1.5**

__init__(*colorize=<StereoColor.RGB: 2>, colormap=<factory>, wls_filter=False, wls_lambda=8000, wls_sigma=1.5*)

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

Parameters

- **colorize** (depthai_sdk.visualize.configs.StereoColor) –
- **colormap** (*numpy.ndarray*) –
- **wls_filter** (*bool*) –
- **wls_lambda** (*float*) –
- **wls_sigma** (*float*) –

Return type None

class depthai_sdk.**TextConfig**(*font_face=0, font_color=(255, 255, 255), font_transparency=0.5, font_scale=1.0, font_thickness=2, font_position=<TextPosition.TOP_LEFT: 0>, background_color=None, background_transparency=0.5, outline_color=(0, 0, 0), line_type=16, auto_scale=True*)

Bases: object

Configuration for drawing labels.

Attributes:

font_face
font_color
font_transparency
font_scale
font_thickness
font_position
background_color
background_transparency
outline_color
line_type
auto_scale

Methods:

__init__([font_face, font_color, . . .])	Initialize self.

font_face: int **= 0**

font_color: Tuple[int, int, int] **= (255, 255, 255)**

font_transparency: float = 0.5

font_scale: float = 1.0

font_thickness: int = 2

font_position: *depthai_sdk.visualize.configs.TextPosition* = 0

background_color: Optional[Tuple[int, int, int]] = None

background_transparency: float = 0.5

outline_color: Tuple[int, int, int] = (0, 0, 0)

line_type: int = 16

auto_scale: bool = True

__init__(*font_face=0, font_color=(255, 255, 255), font_transparency=0.5, font_scale=1.0, font_thickness=2, font_position=<TextPosition.TOP_LEFT: 0>, background_color=None, background_transparency=0.5, outline_color=(0, 0, 0), line_type=16, auto_scale=True*)

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

Parameters

- **font_face** (*int*) –
- **font_color** (*Tuple[int, int, int]*) –
- **font_transparency** (*float*) –
- **font_scale** (*float*) –
- **font_thickness** (*int*) –
- **font_position** (depthai_sdk.visualize.configs.TextPosition) –
- **background_color** (*Optional[Tuple[int, int, int]]*) –
- **background_transparency** (*float*) –
- **outline_color** (*Tuple[int, int, int]*) –
- **line_type** (*int*) –
- **auto_scale** (*bool*) –

Return type None

class depthai_sdk.**TextPosition**(*value*)

Bases: enum.IntEnum

Where on frame do we want to print text.

Attributes:

TOP_LEFT
MID_LEFT
BOTTOM_LEFT
TOP_MID
MID
BOTTOM_MID
TOP_RIGHT
MID_RIGHT
BOTTOM_RIGHT

TOP_LEFT = 0

MID_LEFT = 1

BOTTOM_LEFT = 2

TOP_MID = 10

MID = 11

BOTTOM_MID = 12

TOP_RIGHT = 20

MID_RIGHT = 21

BOTTOM_RIGHT = 22

class depthai_sdk.**Thread**(*group=None*, *target=None*, *name=None*, *args=()*, *kwargs=None*, *, *daemon=None*)

Bases: object

A class that represents a thread of control.

This class can be safely subclassed in a limited fashion. There are two ways to specify the activity: by passing a callable object to the constructor, or by overriding the run() method in a subclass.

Methods:

__init__([group, target, name, args, . . .])	This constructor should always be called with keyword arguments.
start(*a, **kw)	
run()	Method representing the thread's activity.
join([timeout])	Wait until the thread terminates.
is_alive()	Return whether the thread is alive.
isAlive()	Return whether the thread is alive.
isDaemon()	
setDaemon(daemonic)	
getName()	
setName(name)	

Attributes:

name	A string used for identification purposes only.
ident	Thread identifier of this thread or None if it has not been started.
native_id	Native integral thread ID of this thread, or None if it has not been started.
daemon	A boolean value indicating whether this thread is a daemon thread.

__init__(*group=None*, *target=None*, *name=None*, *args=()*, *kwargs=None*, *, *daemon=None*)

This constructor should always be called with keyword arguments. Arguments are:

group should be None; reserved for future extension when a ThreadGroup class is implemented.

target is the callable object to be invoked by the run() method. Defaults to None, meaning nothing is called.

name is the thread name. By default, a unique name is constructed of the form "Thread-N" where N is a

small decimal number.

args is the argument tuple for the target invocation. Defaults to ().

kwargs is a dictionary of keyword arguments for the target invocation. Defaults to {}.

If a subclass overrides the constructor, it must make sure to invoke the base class constructor (Thread.__init__()) before doing anything else to the thread.

start(**a*, ***kw*)

Parameters

- **a** (Thread) –
- **kw** (Any) –

Return type Any

run()

Method representing the thread's activity.

You may override this method in a subclass. The standard run() method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the args and kwargs arguments, respectively.

join(*timeout=None*)

Wait until the thread terminates.

This blocks the calling thread until the thread whose join() method is called terminates – either normally or through an unhandled exception or until the optional timeout occurs.

When the timeout argument is present and not None, it should be a floating point number specifying a timeout for the operation in seconds (or fractions thereof). As join() always returns None, you must call is_alive() after join() to decide whether a timeout happened – if the thread is still alive, the join() call timed out.

When the timeout argument is not present or None, the operation will block until the thread terminates.

A thread can be join()ed many times.

join() raises a RuntimeError if an attempt is made to join the current thread as that would cause a deadlock. It is also an error to join() a thread before it has been started and attempts to do so raises the same exception.

property name

A string used for identification purposes only.

It has no semantics. Multiple threads may be given the same name. The initial name is set by the constructor.

property ident

Thread identifier of this thread or None if it has not been started.

This is a nonzero integer. See the get_ident() function. Thread identifiers may be recycled when a thread exits and another thread is created. The identifier is available even after the thread has exited.

property native_id

Native integral thread ID of this thread, or None if it has not been started.

This is a non-negative integer. See the get_native_id() function. This represents the Thread ID as reported by the kernel.

is_alive()

Return whether the thread is alive.

This method returns True just before the run() method starts until just after the run() method terminates. The module function enumerate() returns a list of all alive threads.

isAlive()

Return whether the thread is alive.

This method is deprecated, use is_alive() instead.

property daemon

A boolean value indicating whether this thread is a daemon thread.

This must be set before start() is called, otherwise RuntimeError is raised. Its initial value is inherited from the creating thread; the main thread is not a daemon thread and therefore all threads created in the main thread default to daemon = False.

The entire Python program exits when only daemon threads are left.

isDaemon()

setDaemon(*daemonic*)

getName()

setName(*name*)

class depthai_sdk.**TrackingConfig**(*max_length=500, deletion_lost_threshold=5, line_thickness=1, fading_tails=False, line_color=(255, 255, 255), line_type=16, show_speed=False*)

Bases: object

Configuration for drawing tracking bounding boxes.

Attributes:

max_length
deletion_lost_threshold
line_thickness
fading_tails
line_color
line_type
show_speed

Methods:

__init__([max_length, ...])	Initialize self.

max_length: int = 500

deletion_lost_threshold: int = 5

line_thickness: int = 1

fading_tails: bool = False

line_color: Tuple[int, int, int] = (255, 255, 255)

line_type: int = 16

show_speed: bool = False

__init__(*max_length=500, deletion_lost_threshold=5, line_thickness=1, fading_tails=False, line_color=(255, 255, 255), line_type=16, show_speed=False*)

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

Parameters

- **max_length** (*int*) –
- **deletion_lost_threshold** (*int*) –
- **line_thickness** (*int*) –
- **fading_tails** (*bool*) –
- **line_color** (*Tuple[int, int, int]*) –
- **line_type** (*int*) –
- **show_speed** (*bool*) –

Return type None

class depthai_sdk.**TurboJPEG**(*lib_path=None*)

Bases: object

A Python wrapper of libjpeg-turbo for decoding and encoding JPEG image.

Methods:

__init__([lib_path])	Initialize self.
decode_header(jpeg_buf)	decodes JPEG header and returns image properties as a tuple.
decode(jpeg_buf[, pixel_format, ...])	decodes JPEG memory buffer to numpy array.
decode_to_yuv(jpeg_buf[, scaling_factor, ...])	decodes JPEG memory buffer to yuv array.
decode_to_yuv_planes(jpeg_buf[, ...])	decodes JPEG memory buffer to yuv planes.
encode(img_array[, quality, pixel_format, ...])	encodes numpy array to JPEG memory buffer.
scale_with_quality(jpeg_buf[, ...])	decompresstoYUV with scale factor, recompresstoYUV with quality factor
crop(jpeg_buf, x, y, w, h[, preserve, gray])	losslessly crop a jpeg image with optional grayscale
crop_multiple(jpeg_buf, crop_parameters[, ...])	Lossless crop and/or extension operations on jpeg image.

Attributes:

scaling_factors

__init__(*lib_path=None*)

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

decode_header(*jpeg_buf*)

decodes JPEG header and returns image properties as a tuple. e.g. (width, height, jpeg_subsample, jpeg_colorspace)

decode(*jpeg_buf, pixel_format=1, scaling_factor=None, flags=0*)

decodes JPEG memory buffer to numpy array.

decode_to_yuv(*jpeg_buf, scaling_factor=None, pad=4, flags=0*)

decodes JPEG memory buffer to yuv array.

decode_to_yuv_planes(*jpeg_buf, scaling_factor=None, strides=(0, 0, 0), flags=0*)

decodes JPEG memory buffer to yuv planes.

encode(*img_array, quality=85, pixel_format=1, jpeg_subsample=1, flags=0*)

encodes numpy array to JPEG memory buffer.

scale_with_quality (*jpeg_buf*, *scaling_factor=None*, *quality=85*, *flags=0*)
decompresstoYUV with scale factor, recompresstoYUV with quality factor

crop (*jpeg_buf*, *x*, *y*, *w*, *h*, *preserve=False*, *gray=False*)
losslessly crop a jpeg image with optional grayscale

crop_multiple (*jpeg_buf*, *crop_parameters*, *background_luminance=1.0*, *gray=False*)
Lossless crop and/or extension operations on jpeg image. Crop origin(s) needs be divisable by the MCU block size and inside the input image, or OSError: Invalid crop request is raised.

Parameters

- **jpeg_buf** (*bytes*) – Input jpeg image.
- **crop_parameters** (*List[Tuple[int, int, int, int]]*) – List of crop parameters defining start x and y origin and width and height of each crop operation.
- **background_luminance** (*float*) – Luminance level (0 -1) to fill background when extending image. Default to 1, resulting in white background.
- **gray** (*bool*) – Produce greyscale output

Returns Cropped and/or extended jpeg images.

Return type List[bytes]

property scaling_factors

class depthai_sdk.**VisConfig** (*output=<factory>*, *stereo=<factory>*, *detection=<factory>*, *text=<factory>*, *tracking=<factory>*, *circle=<factory>*)

Bases: object

Configuration for visualizer.

Attributes:

—

Methods:

__init__([output, stereo, detection, text, . . .])	Initialize self.

output: *depthai_sdk.visualize.configs.OutputConfig*

stereo: *depthai_sdk.visualize.configs.StereoConfig*

detection: *depthai_sdk.visualize.configs.DetectionConfig*

text: *depthai_sdk.visualize.configs.TextConfig*

tracking: *depthai_sdk.visualize.configs.TrackingConfig*

circle: *depthai_sdk.visualize.configs.CircleConfig*

__init__ (*output=<factory>*, *stereo=<factory>*, *detection=<factory>*, *text=<factory>*, *tracking=<factory>*, *circle=<factory>*)
Initialize self. See help(type(self)) for accurate signature.

Parameters

- **output** (depthai_sdk.visualize.configs.OutputConfig) –
- **stereo** (depthai_sdk.visualize.configs.StereoConfig) –

- **detection** (depthai_sdk.visualize.configs.DetectionConfig) –
- **text** (depthai_sdk.visualize.configs.TextConfig) –
- **tracking** (depthai_sdk.visualize.configs.TrackingConfig) –
- **circle** (depthai_sdk.visualize.configs.CircleConfig) –

Return type None

class depthai_sdk.**Visualizer**(*scale=None, fps=False*)

Bases: object

Methods:

__init__([scale, fps])	Initialize self.
add_object(obj)	Call *set_config*, *set_frame_shape* and *prepare* for the object and add it to the list of objects.
add_bbox(bbox[, color, thickness, ...])	Add a bounding box to the visualizer.
add_detections(detections[, normalizer, ...])	Add detections to the visualizer.
add_text(text[, coords, size, color, ...])	Add text to the visualizer.
add_trail(tracklets, label_map[, bbox])	Add a trail to the visualizer.
add_circle(coords, radius[, color, thickness])	Add a circle to the visualizer.
add_line(pt1, pt2[, color, thickness])	Add a line to the visualizer.
add_mask(mask, alpha)	Add a mask to the visualizer.
drawn(frame)	Draw all objects on the frame if the platform is PC.
show(packet)	Show the packet on the screen.
serialize([force_reset])	Serialize all contained objects to JSON.
reset()	Clear all objects.
output([img_scale, show_fps])	Configure the output of the visualizer.
stereo([colorize, colormap, wls_filter, ...])	
detections([thickness, fill_transparency, ...])	Configure how bounding boxes will look like.
text([font_face, font_color, ...])	Configure how text will look like.
tracking([max_length, ...])	Configure how tracking trails will look like.
segmentation([mask_alpha])	
close()	

Attributes:

frame_shape

__init__(*scale=None, fps=False*)

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

Parameters

- **scale** (*Optional[float]*) –
- **fps** (*bool*) –

add_object(*obj*)

Call *set_config*, *set_frame_shape* and *prepare* for the object and add it to the list of objects. :param obj: The object to add.

Returns self

Parameters **obj** (depthai_sdk.visualize.objects.GenericObject) –

Return type *depthai_sdk.visualize.visualizer.Visualizer*

add_bbox(*bbox*, *color=None*, *thickness=None*, *bbox_style=None*, *label=None*)

Add a bounding box to the visualizer.

Parameters

- **bbox** (*depthai_sdk.visualize.bbox.BoundingBox*) – Bounding box.
- **label** (*Optional[str]*) – Label for the detection.
- **thickness** (*Optional[int]*) – Bounding box thickness.
- **color** (*Optional[Tuple[int, int, int]]*) – Bounding box color (RGB).
- **bbox_style** (*Optional[depthai_sdk.visualize.configs.BboxStyle]*) – Bounding box style (one of depthai_sdk.visualize.configs.BboxStyle).

Returns self

Return type *depthai_sdk.visualize.visualizer.Visualizer*

add_detections(*detections*, *normalizer=None*, *label_map=None*, *spatial_points=None*, *label_color=None*, *label_background_color=None*, *label_background_transparency=None*, *is_spatial=False*, *bbox=None*)

Add detections to the visualizer.

Parameters

- **detections** (*List[Union[depthai.ImgDetection, depthai.Tracklet]]*) – List of detections.
- **normalizer** (*Optional[depthai_sdk.visualize.bbox.BoundingBox]*) – Normalizer object.
- **label_map** (*Optional[List[Tuple[str, Tuple]]]*) – List of tuples (label, color).
- **spatial_points** (*Optional[List[depthai.Point3f]]*) – List of spatial points. None if not spatial.
- **label_color** (*Optional[Tuple[int, int, int]]*) – Color for the label.
- **label_background_color** (*Optional[Tuple[int, int, int]]*) – Color for the label background.
- **label_background_transparency** (*Optional[float]*) – Transparency for the label background.
- **is_spatial** – Flag that indicates if the detections are spatial.
- **bbox** (*Optional[Union[numpy.ndarray, Tuple[int, int, int, int]]]*) – Bounding box, if there's a detection inside a bounding box.

Returns self

Return type *depthai_sdk.visualize.visualizer.Visualizer*

add_text(*text*, *coords=None*, *size=None*, *color=None*, *thickness=None*, *outline=True*, *background_color=None*, *background_transparency=0.5*, *bbox=None*, *position=<TextPosition.TOP_LEFT: 0>*, *padding=10*)

Add text to the visualizer.

Parameters

- **text** (*str*) – Text.
- **coords** (*Optional[Tuple[int, int]]*) – Coordinates.

- **size** (*Optional[int]*) – Size of the text.
- **color** (*Optional[Tuple[int, int, int]]*) – Color of the text.
- **thickness** (*Optional[int]*) – Thickness of the text.
- **outline** (*bool*) – Flag that indicates if the text should be outlined.
- **background_color** (*Optional[Tuple[int, int, int]]*) – Background color.
- **background_transparency** (*float*) – Background transparency.
- **bbox** (*Optional[Union[numpy.ndarray, Tuple, depthai_sdk.visualize.bbox.BoundingBox]]*) – Bounding box.
- **position** (depthai_sdk.visualize.configs.TextPosition) – Position.
- **padding** (*int*) – Padding.

Returns self

Return type *depthai_sdk.visualize.visualizer.Visualizer*

add_trail (*tracklets*, *label_map*, *bbox=None*)
Add a trail to the visualizer.

Parameters

- **tracklets** (*List[depthai.Tracklet]*) – List of tracklets.
- **label_map** (*List[Tuple[str, Tuple]]*) – List of tuples (label, color).
- **bbox** (*Optional[depthai_sdk.visualize.bbox.BoundingBox]*) – Bounding box.

Returns self

Return type *depthai_sdk.visualize.visualizer.Visualizer*

add_circle (*coords*, *radius*, *color=None*, *thickness=None*)
Add a circle to the visualizer.

Parameters

- **coords** (*Tuple[int, int]*) – Center of the circle.
- **radius** (*int*) – Radius of the circle.
- **color** (*Optional[Tuple[int, int, int]]*) – Color of the circle.
- **thickness** (*Optional[int]*) – Thickness of the circle.

Returns self

Return type *depthai_sdk.visualize.visualizer.Visualizer*

add_line (*pt1*, *pt2*, *color=None*, *thickness=None*)
Add a line to the visualizer.

Parameters

- **pt1** (*Tuple[int, int]*) – Start coordinates.
- **pt2** (*Tuple[int, int]*) – End coordinates.
- **color** (*Optional[Tuple[int, int, int]]*) – Color of the line.
- **thickness** (*Optional[int]*) – Thickness of the line.

Returns self

Return type *depthai_sdk.visualize.visualizer.Visualizer*

add_mask(*mask*, *alpha*)

Add a mask to the visualizer.

Parameters

- **mask** (*numpy.ndarray*) – Mask represented as uint8 numpy array.
- **alpha** (*float*) – Transparency of the mask.

Returns self

drawn(*frame*)

Draw all objects on the frame if the platform is PC. Otherwise, serialize the objects and communicate with the RobotHub application.

Parameters **frame** (*numpy.ndarray*) – The frame to draw on.

Returns np.ndarray if the platform is PC, None otherwise.

Return type Optional[numpy.ndarray]

show(*packet*)

Show the packet on the screen.

serialize(*force_reset=True*)

Serialize all contained objects to JSON.

Parameters **force_reset** (*bool*) – Flag that indicates if the objects should be cleared after serialization.

Returns Stringified JSON.

Return type str

reset()

Clear all objects.

output(*img_scale=None*, *show_fps=None*)

Configure the output of the visualizer.

Parameters

- **img_scale** (*Optional[float]*) – Scale of the output image.
- **show_fps** (*Optional[bool]*) – Flag that indicates if the FPS should be shown.

Returns self

Return type *depthai_sdk.visualize.visualizer.Visualizer*

stereo(*colorize=None*, *colormap=None*, *wls_filter=None*, *wls_lambda=None*, *wls_sigma=None*)

Parameters

- **colorize** (*Optional[depthai_sdk.visualize.configs.StereoColor]*) –
- **colormap** (*Optional[int]*) –
- **wls_filter** (*Optional[bool]*) –
- **wls_lambda** (*Optional[float]*) –
- **wls_sigma** (*Optional[float]*) –

detections(*thickness=None*, *fill_transparency=None*, *bbox_roundness=None*, *color=None*, *bbox_style=None*, *line_width=None*, *line_height=None*, *hide_label=None*, *label_position=None*, *label_padding=None*)

Configure how bounding boxes will look like. :param thickness: Thickness of the bounding box. :param fill_transparency: Transparency of the bounding box. :param bbox_roundness: Roundness of the bounding box. :param color: Color of the bounding box. :param bbox_style: Style of the bounding box. :param line_width: Width of the bbox horizontal lines CORNERS or ROUNDED_CORNERS style is used. :param line_height: Height of the bbox vertical lines when CORNERS or ROUNDED_CORNERS style is used. :param hide_label: Flag that indicates if the label should be hidden. :param label_position: Position of the label. :param label_padding: Padding of the label.

Returns self

Parameters

- **thickness** (*Optional[int]*) –
- **fill_transparency** (*Optional[float]*) –
- **bbox_roundness** (*Optional[float]*) –
- **color** (*Optional[Tuple[int, int, int]]*) –
- **bbox_style** (*Optional[depthai_sdk.visualize.configs.BboxStyle]*) –
- **line_width** (*Optional[float]*) –
- **line_height** (*Optional[float]*) –
- **hide_label** (*Optional[bool]*) –
- **label_position** (*Optional[depthai_sdk.visualize.configs.TextPosition]*) –
- **label_padding** (*Optional[int]*) –

Return type *depthai_sdk.visualize.visualizer.Visualizer*

text(*font_face=None*, *font_color=None*, *font_transparency=None*, *font_scale=None*, *font_thickness=None*, *font_position=None*, *background_transparency=None*, *background_color=None*, *outline_color=None*, *line_type=None*, *auto_scale=None*)

Configure how text will look like.

Parameters

- **font_face** (*Optional[int]*) – Font face (from cv2).
- **font_color** (*Optional[Tuple[int, int, int]]*) – Font color.
- **font_transparency** (*Optional[float]*) – Font transparency.
- **font_scale** (*Optional[float]*) – Font scale.
- **font_thickness** (*Optional[int]*) – Font thickness.
- **font_position** (*Optional[depthai_sdk.visualize.configs.TextPosition]*) – Font position.
- **background_transparency** (*Optional[float]*) – Text background transparency.
- **background_color** (*Optional[Tuple[int, int, int]]*) – Text background color.
- **outline_color** (*Optional[Tuple[int, int, int]]*) – Outline color.

- **line_type** (*Optional[int]*) – Line type (from cv2).
- **auto_scale** (*Optional[bool]*) – Flag that indicates if the font scale should be automatically adjusted.

Returns self

Return type *depthai_sdk.visualize.visualizer.Visualizer*

tracking(*max_length=None*, *deletion_lost_threshold=None*, *line_thickness=None*, *fading_tails=None*, *show_speed=None*, *line_color=None*, *line_type=None*, *bg_color=None*)

Configure how tracking trails will look like.

Parameters

- **max_length** (*Optional[int]*) – Maximum length of the trail (in pixels).
- **deletion_lost_threshold** (*Optional[int]*) – Number of consequent LOST statuses after which the trail is deleted.
- **line_thickness** (*Optional[int]*) – Thickness of the line.
- **fading_tails** (*Optional[bool]*) – Flag that indicates if the tails should fade.
- **show_speed** (*Optional[bool]*) – Flag that indicates if the speed should be shown.
- **line_color** (*Optional[Tuple[int, int, int]]*) – Color of the line.
- **line_type** (*Optional[int]*) – Type of the line (from cv2).
- **bg_color** (*Optional[Tuple[int, int, int]]*) – Text background color.

Returns self

Return type *depthai_sdk.visualize.visualizer.Visualizer*

segmentation(*mask_alpha=None*)

Parameters **mask_alpha** (*Optional[float]*) –

Return type *depthai_sdk.visualize.visualizer.Visualizer*

property frame_shape

close()

class depthai_sdk.**XoutFrames**(*frames*, *fourcc=None*)

Bases: depthai_sdk.oak_outputs.xout.xout_base.XoutBase

Stream of frames. Single message, no syncing required.

Methods:

__init__(frames[, fourcc])	**param frames** StreamXout object.
set_fourcc(fourcc)	
xstreams()	
new_msg(name, msg)	
get_codec()	

__init__(*frames*, *fourcc=None*)

Parameters

- **frames** (*depthai_sdk.oak_outputs.xout.xout_base.StreamXout*) –

StreamXout object.

- **fourcc** (*Optional[str]*) – Codec to use for encoding. If None, no encoding will be done.

set_fourcc(*fourcc*)

Parameters **fourcc** (*str*) –

Return type depthai_sdk.oak_outputs.xout.xout_frames.XoutFrames

xstreams()

Return type List[depthai_sdk.oak_outputs.xout.xout_base.StreamXout]

new_msg(*name*, *msg*)

Parameters **name** (*str*) –

get_codec()

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

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

depthai_sdk.**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

depthai_sdk.**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.**dataclass**(*cls=None*, */*, ***, *init=True*, *repr=True*, *eq=True*, *order=False*, *unsafe_hash=False*, *frozen=False*)

Returns the same class as was passed in, with dunder methods added based on the fields defined in the class.

Examines PEP 526 __annotations__ to determine fields.

If init is true, an __init__() method is added to the class. If repr is true, a __repr__() method is added. If order is true, rich comparison dunder methods are added. If unsafe_hash is true, a __hash__() method function is added. If frozen is true, fields may not be assigned to after instance creation.

depthai_sdk.**downloadContent**(*url*)

Parameters **url** (*str*) –

Return type pathlib.Path

depthai_sdk.**downloadRecording**(*name*, *keys*)

Parameters

- **name** (*str*) –

- **keys** (*List[str]*) –

Return type pathlib.Path

`depthai_sdk.downloadYTVideo(video, output_dir=None)`

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
- **output_dir** (*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.field(*, default=<dataclasses._MISSING_TYPE object>, default_factory=<dataclasses._MISSING_TYPE object>, init=True, repr=True, hash=None, compare=True, metadata=None)`

Return an object to identify dataclass fields.

default is the default value of the field. default_factory is a 0-argument function called to initialize a field's value. If init is True, the field will be a parameter to the class's __init__() function. If repr is True, the field will be included in the object's repr(). If hash is True, the field will be included in the object's hash(). If compare is True, the field will be used in comparison functions. metadata, if specified, must be a mapping which is stored but not otherwise examined by dataclass.

It is an error to specify both default and default_factory.

`depthai_sdk.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.getAvailableRecordings()`

Get available (online) depthai-recordings. Returns list of available recordings and it's size

Return type Dict[str, Tuple[List[str], int]]

`depthai_sdk.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.**getLocalRecording**(*recording*)

Parameters **recording** (*str*) –

Return type Optional[pathlib.Path]

depthai_sdk.**isUrl**(*source*)

Parameters **source** (*Union[str, pathlib.Path]*) –

Return type bool

depthai_sdk.**isYoutubeLink**(*source*)

Parameters **source** (*str*) –

Return type bool

depthai_sdk.**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.**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.**monotonic**() → float

Monotonic clock, cannot go backward.

class depthai_sdk.**partial**

Bases: object

partial(func, *args, **keywords) - new function with partial application of the given arguments and keywords.

Attributes:

args	tuple of arguments to future partial calls
func	function object to use in future partial calls
keywords	dictionary of keyword arguments to future partial calls

args
tuple of arguments to future partial calls

func
function object to use in future partial calls

keywords
dictionary of keyword arguments to future partial calls

`depthai_sdk.report_crash_dump(device)`
Report crash dump to Sentry if sentry is enabled and crash dump is available.

Parameters **device** (*depthai.Device*) – DepthAI device object that will be used to get crash dump.

Return type None

`depthai_sdk.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.set_sentry_status(status=True)`
Set sentry status in config file.

Parameters **status** (*bool*) – True if sentry should be enabled, False otherwise. Default is True.

Returns None.

Return type None

`depthai_sdk.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.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.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

PYTHON MODULE INDEX

d

INDEX

Symbols

A

B

C

D

E

F

G

H

I

O

P

Q

R

S

T

U

V

W

X