# DepthAI Docs

Luxonis

Mar 29, 2021

# SUBPROJECTS:

*Learn about DepthAI ecosystem, available devices, calibration instructions, and more*

DepthAI is the embedded spatial AI platform that helps you build products with true real-time 3D object localization (think 3D object detection) and tracking. DepthAI offloads AI, depth vision and more - processed direct from built-in cameras - freeing your host to process application-specific data. Best of all, it is modular and MIT-licensed open source, affording adding these Spatial AI/CV super powers to real commercial products.

CHAPTER

ONE

# AVAILABLE INTERFACES

CHAPTER

TWO

# LUXONIS GITHUB REPOSITORIES

Table 1: Core Repositories

| | |
|---|---|
| depthai-python | Here you'll find Python bindings creating the Python API of DepthAI |
| depthai-core | Our core API written in C++ |
| depthai-shared | This repository contains shared data between our main firmware and depthai-core host library. |
| depthai_ros2 | DepthAI ROS2 Wrapper. This is an attempt at basic DepthAI to ROS2 interface. It's largely leveraging the existing depthai python demo on https://github.com/luxonis/depthai. |
| depthai-spi-api | API of the SPI protocol |
| depthai-spi-library | DepthAI SPI Library |
| depthai-bootloader-shared | The depthai-bootloader-shared repository contains shared data between our bootloader firmware and depthai-core host library. |
| depthai-hardware | This repository contains Luxonis open sourced baseboards, and contains Altium design files, documentation, and pictures to help you understand more about the embedded hardware that powers DepthAI. |

Table 2: Tools / Docs Repositories

| | |
|---|---|
| depthai-docs-website | If you wan't to contribute and update our docs, you can simply create a pull request. |
| depthai-gui | DepthAI GUI is a WYSIWYG tool that allows to create a custom DepthAI pipelines, run them and see the results - all in one tool. |
| depthai-ml-training | Here you can find repositories to help you connect your NN and create BLOBs. |
| depthai-tutorials | This repo contains source code for tutorials published on docs.luxonis.com. |
| blobconverter | Web-based tool to convert model into MyriadX blob |
| Factory-calibration-DepthAI | Factory Calibration (WIP); This package contains two ROS workspace one is for depthai capture and calibration node and another is for Interbotix ViperX 300 Robot Arm 6DOF (KIT-VIPX300-6DOF) arm bot control using moveit. |
| depthai-docker | This repository contains a Dockerfile, that allows you to run OpenVINO on DepthAI inside a Docker container. |
| depthai-mock | This tool allows you to record the packets produced by DepthAI device into your disk and then play them back again as they would be produced normally - but without actually running the DepthAI |
| sbr-util | Utility to view and manipulate SBR binary images |

Table 3: Demo Repositories

| | |
|---|---|
| depthai | This repo contains a demo application, which can load different networks, create pipelines, record video, etc. This program includes an example of depth & CNN inference and ready to use models. |
| esp32-spi-message-demo | ESP32 reference app for interfacing with DepthAI over SPI |
| depthai-core-example | CMake example project which serves as a template on how to quickly get started with C++ and depthai library |

Table 4: Experiments Repositories:

| | |
|---|---|
| depthai-experiments | In this repository, you'll find various experiments using DepthAI. You can use those examples as a basis or a reference in your application. |
| remote-monitoring | Application that allows user to report an incident when a person or a car will be detected in specified zone. |

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

## 2.1 FAQs & How-To

### 2.1.1 Why Does DepthAI Exist?

In trying to solve an Embedded *Spatial AI* problem (details here), we discovered that although the perfect chip existed, there was no platform (hardware, firmware, or software) which allowed the chip to be used to solve such an Embedded Spatial AI problem.

So we built the platform, known as DepthAI and the OpenCV AI Kit (OAK).

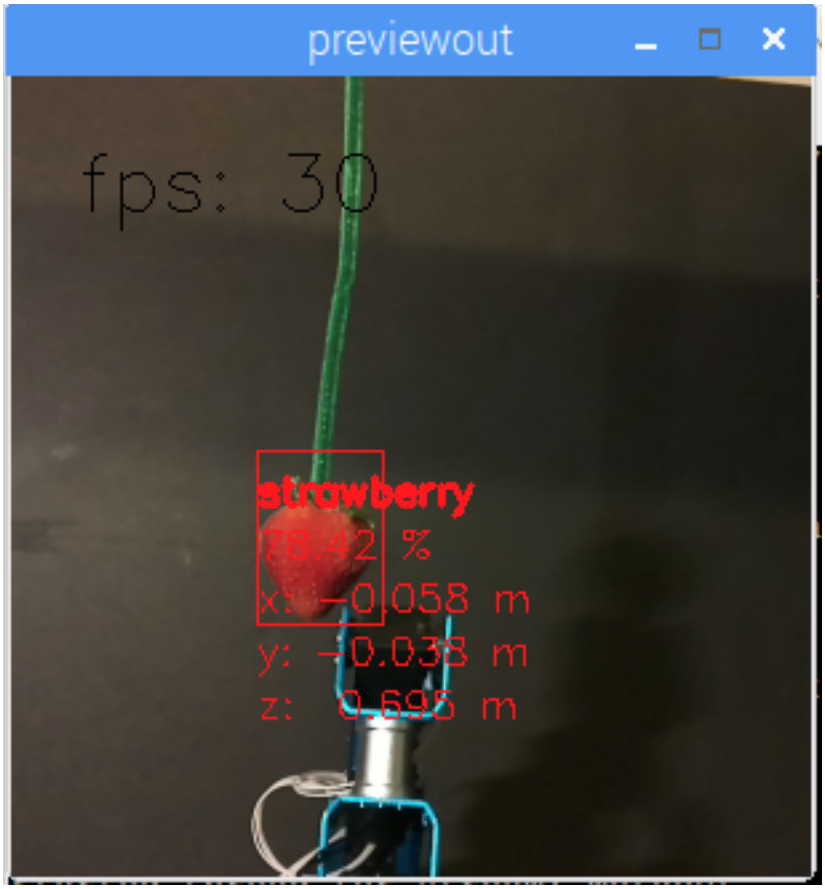
### 2.1.2 What is DepthAI?

DepthAI is *the* Embedded, Performant, Spatial AI+CV platform, composed of an open-source hardware, firmware, software ecosystem that provides turnkey embedded *Spatial AI+CV* and hardware-accelerated computer vision.

It gives embedded systems the super-power of human-like perception in real-time: what an object is and where it is in physical space.

It can be used with off-the-shelf AI models (how-to here) or with custom models using our completely-free training flow (how-to here).

An example of a custom-trained model is below, where DepthAI is used by a robot to autonomously pick and sort strawberries by ripeness.



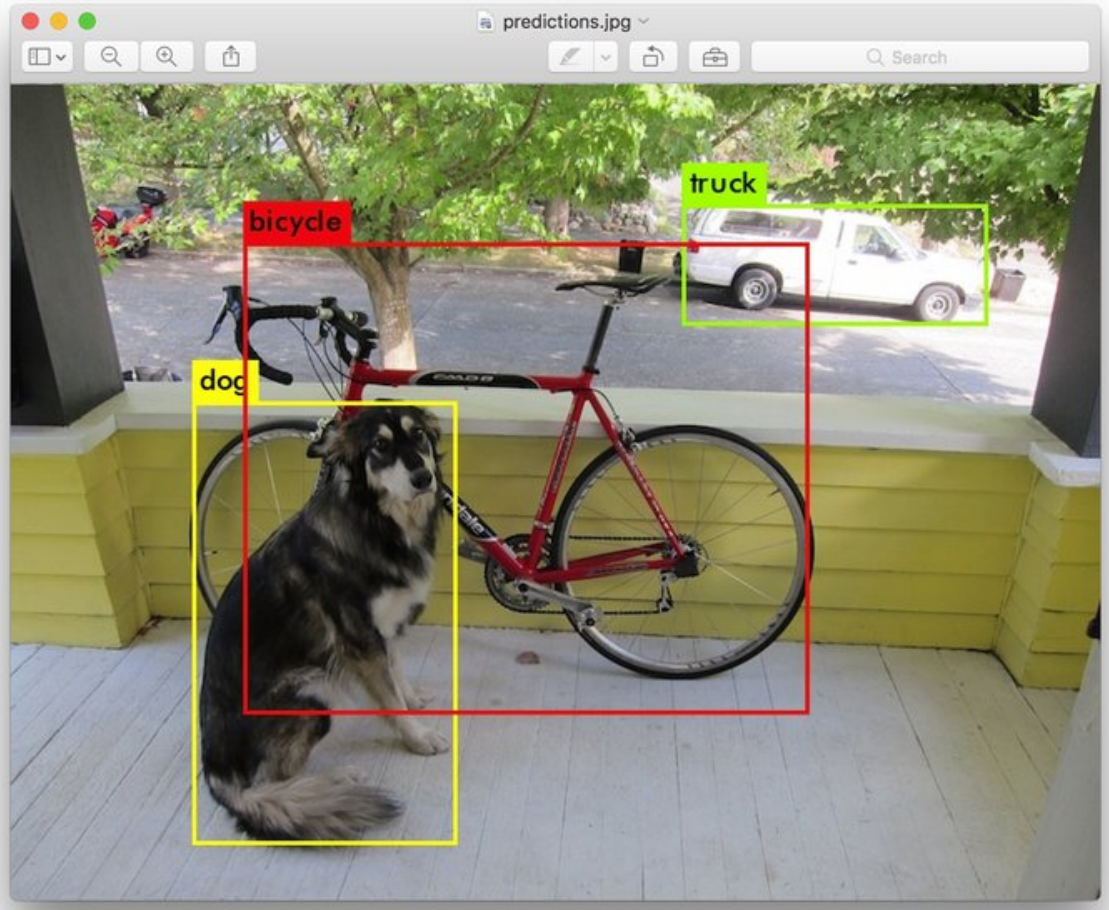

It was trained to do so over the course of a weekend, by a student (for a student project), using our free online training resources.

DepthAI is also open-source (including hardware). This is done so that companies (and even individuals) can prototype and productize solutions quickly, autonomously, and at low risk.

See the summary of our (MIT-Licensed) Github repositories *below*, which include open-source hardware, firmware, software, and machine-learning training.

### 2.1.3 What is SpatialAI? What is 3D Object Localization?

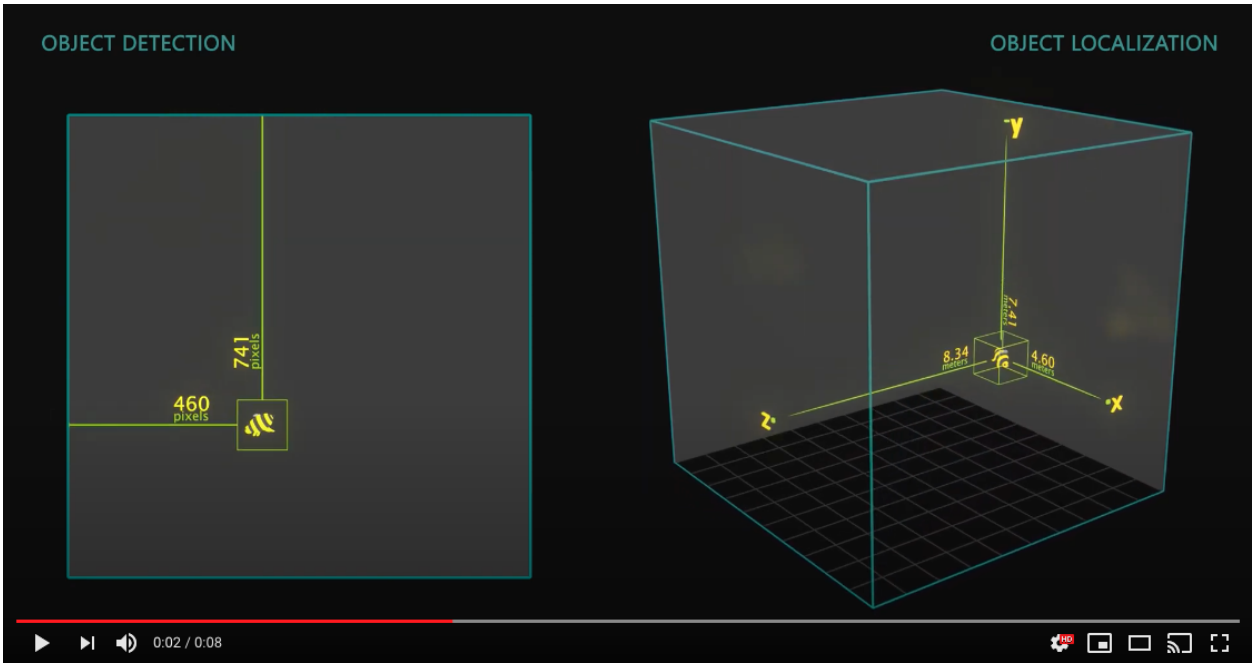
First, it is necessary to define what 'Object Detection' is:




It is the technical term for finding the bounding box of an object of interest, in pixel space (i.e. pixel coordinates), in an image.

3D Object Localization (or 3D Object Detection), is all about finding such objects in physical space, instead of pixel space. This is useful when trying to real-time measure or interact with the physical world.

Below is a visualization to showcase the difference between Object Detection and 3D Object Localization:




Spatial AI is then the super-set of such 2D-equivalent neural networks being extended with spatial information to give them 3D context. So in other words, it's not limited to object detectors being extended to 3D object localizers. Other network types can be extended as well, including any network which returns results in pixel space.

An example of such an extension is using a facial landmark detector on DepthAI. With a normal camera this network returns the 2D coordinates of all 45 facial landmarks (contours of eyes, ears, mouth, eyebrows, etc.) Using this same network with DepthAI, each of these 45 facial landmarks is now a 3D point in physical space instead of 2D points in pixel space.

### 2.1.4 How Does DepthAI Provide Spatial AI Results?

There are two ways to use DepthAI to get Spatial AI results:

1. **Monocular Neural Inference fused with Stereo Depth.** In this mode the neural network is run on a single camera and fused with disparity depth results. The left, right, or RGB camera can be used to run the neural inference.

2. **Stereo Neural Inference.** In this mode the neural network is run in parallel on both the left and right stereo cameras to produce 3D position data directly with the neural network.

In both of these cases, standard neural networks can be used. There is no need for the neural networks to be trained with 3D data.

DepthAI automatically provides the 3D results in both cases using standard 2D-trained networks, as detailed *here*. These modes have differing minimum depth-perception limits, detailed *here*.

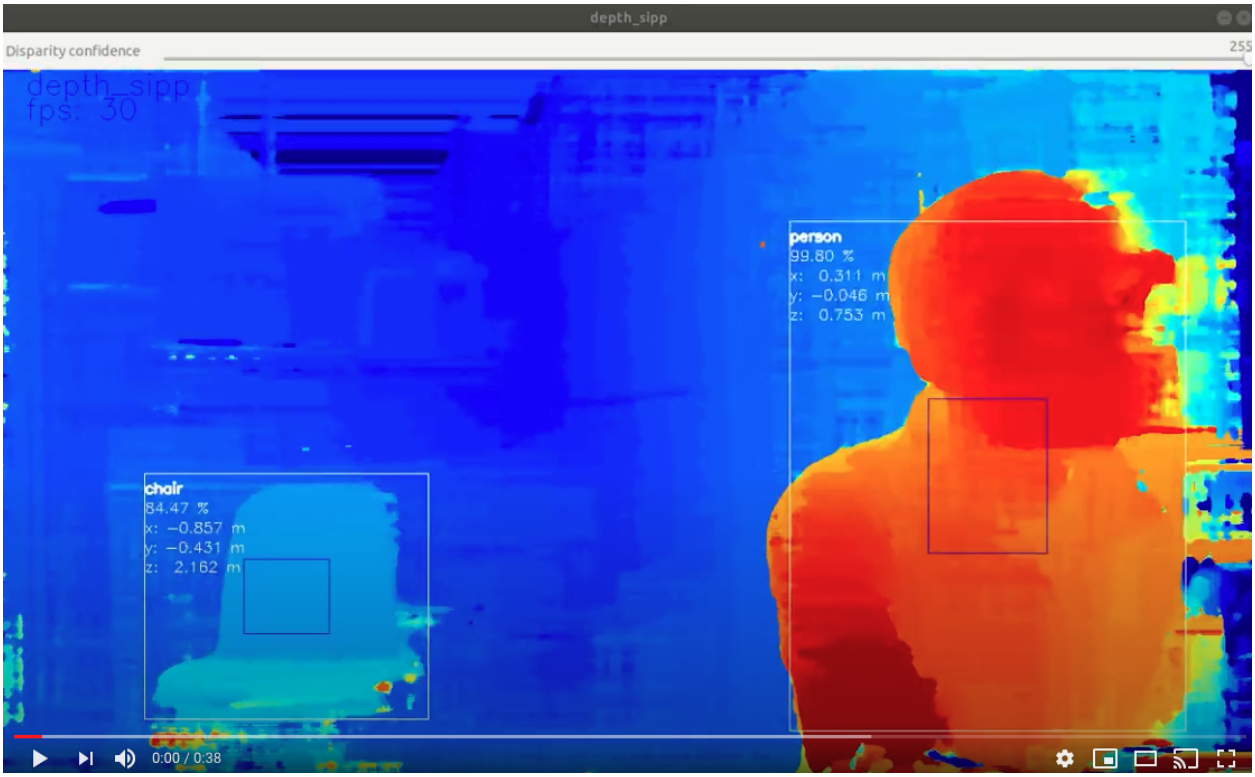
### Monocular Neural Inference fused with Stereo Depth

In this mode, DepthAI runs object detection on a single cameras (user's choice: left, right, or RGB) and the results are fused with the stereo disparity depth results. The stereo disparity results are produced in parallel and in real-time on DepthAI (based on semi global matching (SGBM)).

DepthAI automatically fuses the disparity depth results with the object detector results and uses this depth data for each object in conjunction with the known intrinsics of the calibrated cameras to reproject the 3D position of the detected object in physical space (X, Y, Z coordinates in meters).

And all of these calculations are done onboard to DepthAI without any processing load to any other systems. This technique is great for object detectors as it provides the physical location of the centroid of the object - and takes advantage of the fact that most objects are usually many pixels so the disparity depth results can be averaged to produce a more accurate location.

A visualization of this mode is below.




In this case the neural inference (20-class object detection per here) was run on the RGB camera and the results were overlaid onto the depth stream. The DepthAI reference Python script can be used to show this out (`python3 depthai_demo.py -s metaout depth -bb` is the command used to produce the video above).

And if you'd like to know more about the underlying math that DepthAI is using to perform the stereo depth, see this excellent blog post here here. And if you'd like to run the same example run in that blog, on DepthAI, see this depthai-experiment.

### What is the Max Stereo Disparity Depth Resolution?

The maximum resolution for the depthai depth map is 1280x800 (1MP), with either a 92-pixel (default) or 192-pixel disparity search (when *Extended Disparity* is enabled) and either a full-pixel (default) or sub-pixel matching with precision of 32 sub-pixel steps (when *Sub-Pixel Disparity* is enabled), resulting in a maximum theoretical depth precision of 192 (extended disparity search mode) * 32 (sub-pixel disparity search enabled) of 6,144. Note however that sub-pixel and extended disparity, as of this writing, are not supported simultaneously (but will be Q2 2021), so the maximum depth precision is 3,072 depth steps. More information on the disparity depth modes are below:

1. Default (96-pixel disparity search): 1280x800 or 640x400, 96 depth steps
2. Extended Disparity (192-pixel disparity search), *here*: 1280x800 or 640x400, 192 depth steps
3. Subpixel Disparity (32 sub-pixel steps), *here*, 1280x800 or 640x400, 96 depth steps * 32 subpixel depth steps = 3,072 depth steps.
4. LR-Check Disparity, *here*: 1280x800, with disparity run in both directions for allowing recentering of the depth.

(see *Extended Disparity* below

### Stereo Neural Inference

In this mode DepthAI runs the neural network in parallel on both the left and right stereo cameras. The disparity of the results are then triangulated with the calibrated camera intrinsics (programmed into the EEPROM of each DepthAI unit) to give 3D position of all the detected features.

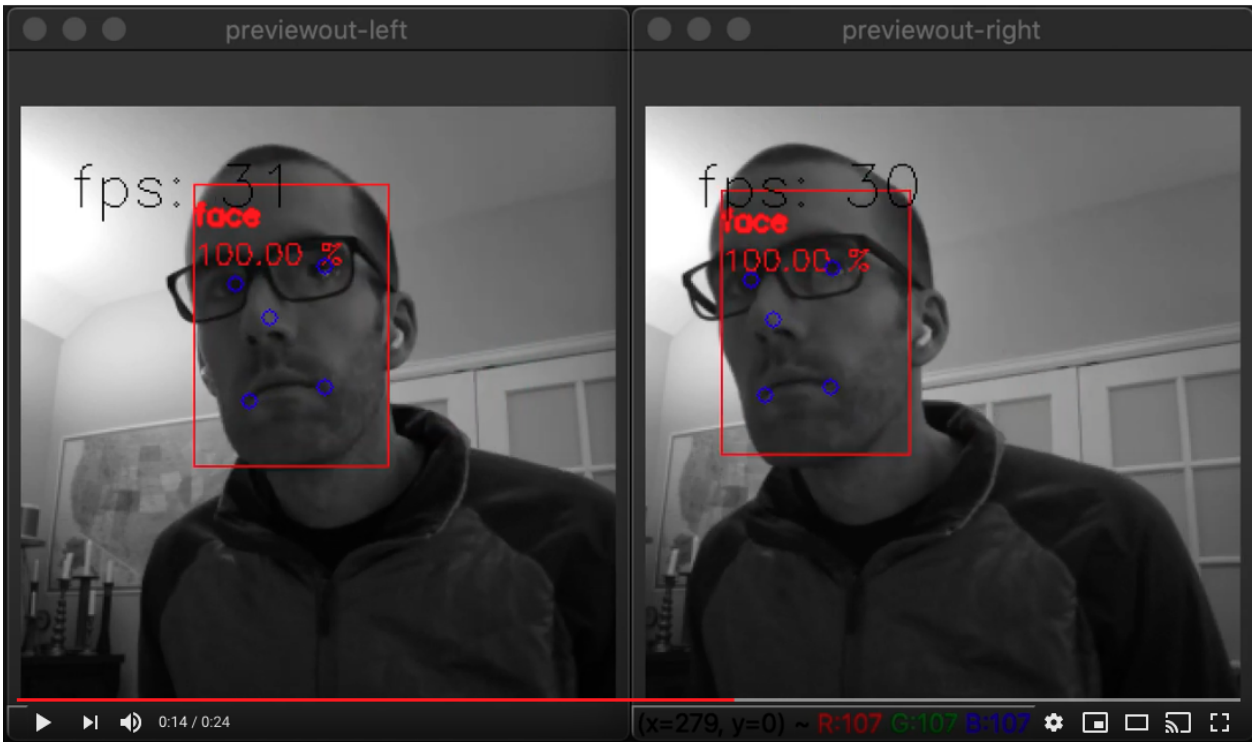
This **stereo neural inference** mode affords accurate 3D Spatial AI for networks which produce single-pixel locations of features such as facial landmark estimation, pose estimation, or other meta-data which provides feature locations like this.

Examples include finding the 3D locations of:

- Facial landmarks (eyes, ears, nose, edges of mouth, etc.)
- Features on a product (screw holes, blemishes, etc.)
- Joints on a person (e.g. elbow, knees, hips, etc.)
- Features on a vehicle (e.g. mirrors, headlights, etc.)
- Pests or disease on a plant (i.e. features that are too small for object detection + stereo depth)

Again, this mode does not require the neural networks to be trained with depth data. DepthAI takes standard, off-the-shelf 2D networks (which are significantly more common) and uses this stereo inference to produce accurate 3D results.

An example of stereo neural inference is below.




And this is actually an interesting case as it demonstrates two things on DepthAI:

1. Stereo inference (i.e. running the neural network(s) running on both the left and right cameras in parallel)
2. Multi-stage inference (i.e. face detection flowed directly into facial landmark directly on DepthAI)

The command used to run this on DepthAI is

```
python3 depthai_demo.py -cnn face-detection-retail-0004 -cnn2 landmarks-regression-
↪retail-0009 -cam left_right -dd -sh 12 -cmx 12 -nce 2 -monor 400 -monof 30
```

Where `cam` specifies to run the neural network on both cameras, `-cnn` specifies the first-stage network to run (face detection, in this case), `-cnn2` specifies the second-stage network (facial landmark detection, in this case), and `-dd` disables running disparity depth calculations (since they are unused in this mode).

### Notes

It is worth noting that monocular neural inference fused with stereo depth is possible for networks like facial-landmark detectors, pose estimators, etc. that return single-pixel locations (instead of for example bounding boxes of semantically-labeled pixels), but stereo neural inference is advised for these types of networks better results as unlike object detectors (where the object usually covers many pixels, typically hundreds, which can be averaged for an excellent depth/position estimation), landmark detectors typically return single-pixel locations. So if there doesn't happen to be a good stereo-disparity result for that single pixel, the position can be wrong.

And so running stereo neural inference excels in these cases, as it does not rely on stereo disparity depth at all, and instead relies purely on the results of the neural network, which are robust at providing these single pixel results. And triangulation of the parallel left/right outputs results in very-accurate real-time landmark results in 3D space.

### 2.1.5 What is the Gen2 Pipeline Builder?

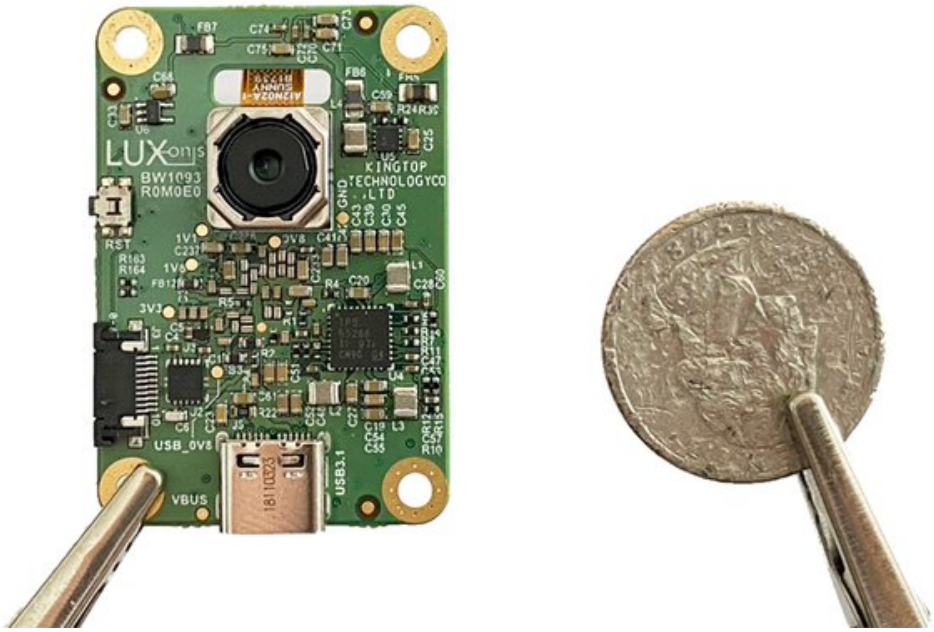
UPDATE: The Gen2 Pipeline Builder is now the standard release of DepthAI. This Gen2 API system was architected to be next-generation software suite for DepthAI and OAK. All DepthAI and OAK hardware work with Gen1 and Gen2 software, as Gen2 is purely a software re-write, no hardware changes. Gen2 is infinitely more flexible, and is the result of all that we learned from the customer deployments of Gen1. Amassing all the requests and need for flexibility from user of Gen1, we made Gen2. In short, Gen2 allows theoretically-infinite permutations of parallel and series CV + AI (neural inference) nodes, limited only by hardware capabilities, whereas Gen1 was limited for example to 2-series and 2-parallel neural inference. Full background on the Gen2 Pipeline Builder is here.

Several Gen2 Examples are here and also the docs for Gen2 are now available in the main docs page.

### 2.1.6 What is megaAI?

The monocular (single-camera) version of DepthAI is megaAI. Because not all solutions to embedded AI/CV problems require spatial information.

We named it `mega` because it's tiny:




megaAI uses all the same hardware, firmware, software, and training stacks as DepthAI (and uses the same DepthAI Github repositories), it is simply the tiny single-camera variant.

You can buy megaAI from our distributors and also our online store here.

### 2.1.7 Which Model Should I Order?

Embedded CV/AI requires all sorts of different shapes/sizes/permutations. And so we have a variety of options to meet these needs. Below is a quick/dirty summary for the ~10,000-foot view of the options:

- **USB3C with Onboard Cameras** (BW1098OBC) - Great for quickly using DepthAI with a computer. All cameras are onboard, and it has a USB3C connection which can be used with any USB3 or USB2 host. This is the basis for OAK-D.
- **USB3C with Modular Cameras** (BW1098FFC) - Great for prototyping flexibility. Since the cameras are modular, you can place them at various stereo baselines. This flexibility comes with a trade - you have to figure out how/where you will mount them, and then once mounted, do a stereo calibration. This is not a TON of work, but keep this in mind, that it's not 'plug and play' like other options - it's more for applications that require custom mounting, custom baseline, or custom orientation of the cameras.
- **MegaAI Single Camera** (BW1093) - This is just like the BW1098OBC, but for those who don't need depth information. Single, small, plug-and-play USB3C AI/CV camera.
- **Raspberry Pi Compute Module Edition** (BW1097) - this one has a built-in Raspberry Pi Compute Module 3B+. So you literally plug it into power and HDMI, and it boots up showing off the power of DepthAI.
- **Embedded DepthAI with WiFi/BT** (BW1092) - Currently this is in Alpha testing. So only buy it if you are comfortable with working with bleeding-edge tech and want to help us refine this product. It is the first Embedded (i.e. SPI-interface) version of DepthAI - so it has additional 128MB NOR flash, so it can boot on its own out of the NOR flash, and not host needs to be present to run. In contrast, the BW1097 can also run on its own, but it is still booting over USB from the Raspberry Pi. This BW1092, the Myriad X can run completely standalone and with no other devices. The built-in ESP32 then provides easy/convenient WiFi/BT support as well as popular integrations like plug-and-play AWS-IoT support, great iOS/Android BT examples, etc.

#### System on Modules

For designing products around DepthAI, we offer system on modules. You can then design your own variants, leveraging our open source hardware. There are three system on modules available:

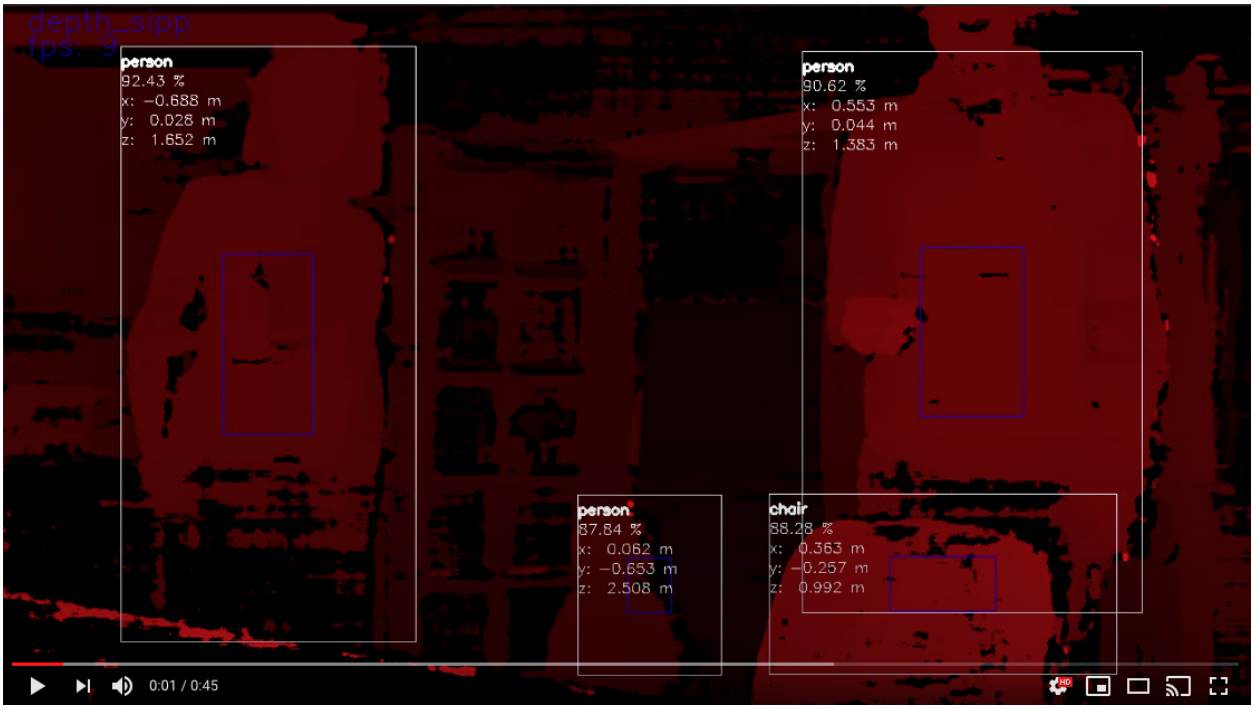
1. BW1099 - USB-boot system on module. For making devices which interface over USB to a host processor running Linux, MacOS, or Windows. In this case, the host processor stores everything, and the BW1099 boots up over USB from the host.
2. BW1099EMB - NOR-flash boot (also capable of USB-boot). For making devices that run standalone, or work with embedded MCUs like ESP32, AVR, STM32F4, etc. Can also USB-boot if/as desirable.
3. BW2099 - NOR flash, eMMC, SD-Card, and USB-boot (selectable via IO on the 2x 100-pin connectors). For making devices that run standalone and require onboard storage (16GB eMMC) and/or Ethernet Support (the onboard PCIE interface through one of the 2x 100-pin connectors, paired with an Ethernet-capable base-board provides Ethernet support).

### 2.1.8 How hard is it to get DepthAI running from scratch? What Platforms are Supported?

Not hard. Usually DepthAI is up/running on your platform within a couple minutes (most of which is download time). The requirements are Python and OpenCV (which are great to have on your system anyway!). see here for supported platforms and how to get up/running with them.

**Raspbian, Ubuntu, macOS, Windows,** and many others are supported and are easy to get up/running. For Install on various platforms are here.

It's a matter of minutes to be up and running with the power of Spatial AI, on the platform of your choice. Below is DepthAI running on my Mac.

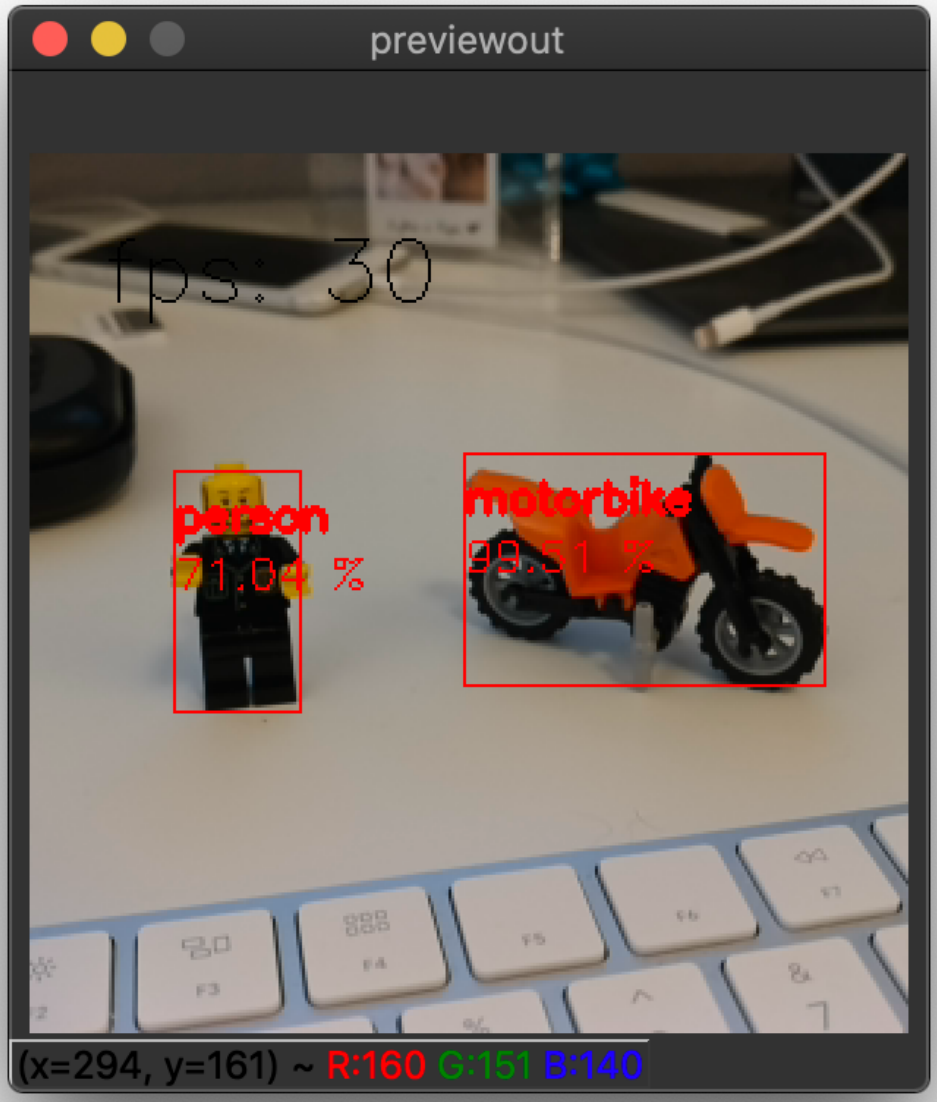



(Click on the image above to pull up the YouTube video.)

The command to get the above output is

```
python3 depthai_demo.py -s metaout previewout depth -ff -bb
```

Here is a single-camera version (megaAI) running with `python3 depthai_demo.py -dd` (to disable showing depth info):


previewout
fps: 30
person
71.04 %
motorbike
99.51 %
(x=294, y=161) ~ R:160 G:151 B:140

### 2.1.9 Is DepthAI and MegaAI easy to use with Raspberry Pi?

Very. It's designed for ease of setup and use, and to keep the Pi CPU not-busy.

See here to get up and running quickly!

### 2.1.10 Can all the models be used with the Raspberry Pi?

Yes, every model can be used, including:

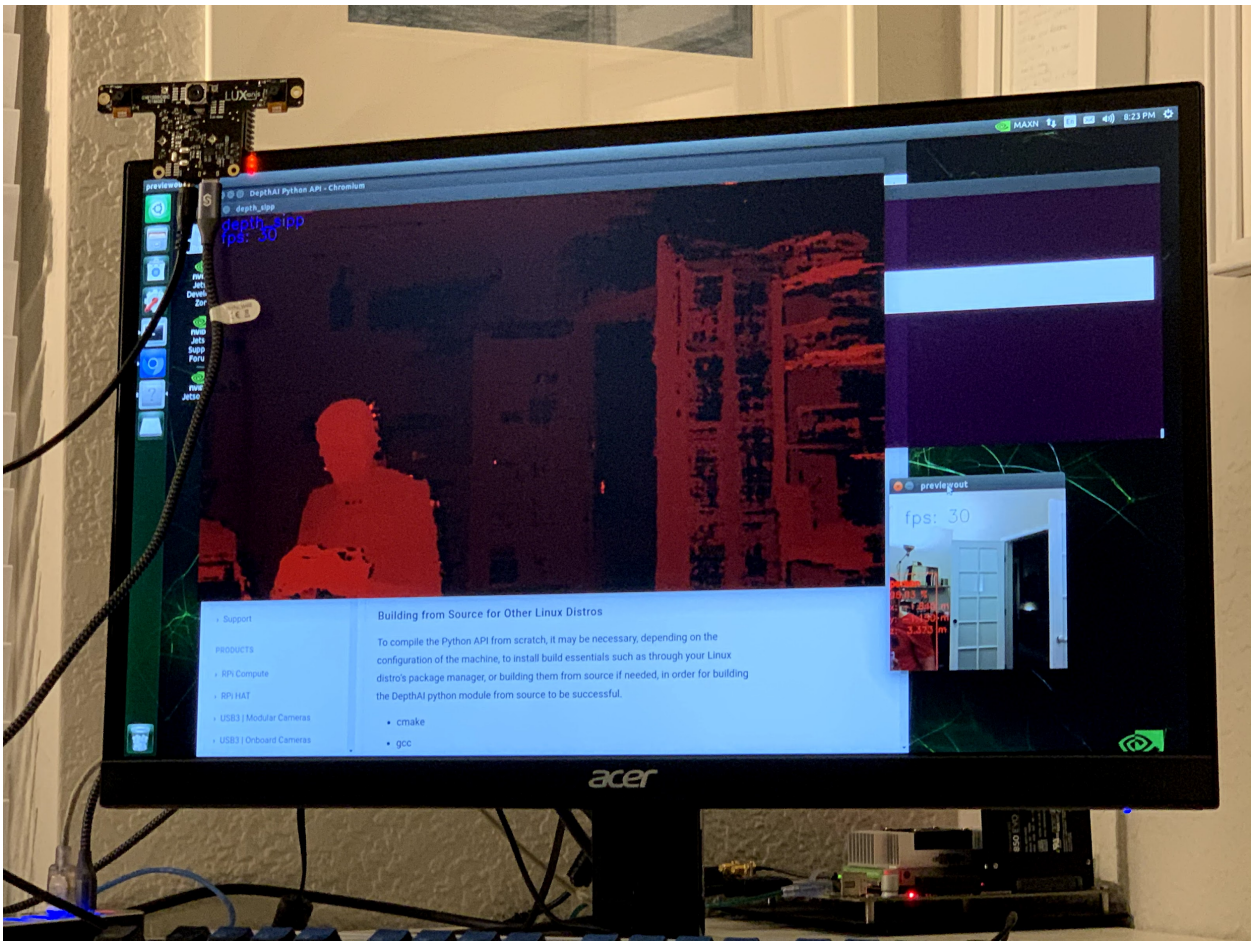
- Raspberry Pi Compute Module Edition (BW1097 - this one has a built-in Raspberry Pi Compute Module 3B+
- Raspberry Pi HAT (BW1094) - this can also be used with other hosts as its interface is USB3
- USB3C with Onboard Cameras BW1098OBC
- USB3C with Modular Cameras BW1098FFC
- MegaAI Single Camera BW1093

We even have some basic ROS support going as well which can be used on the Pi also.

### 2.1.11 Does DepthAI Work on the Nvidia Jetson Series?

Yes, DepthAI and megaAI work cleanly on all the Jetson/Xavier series, and installation is easy. Jetson Nano, Jetson Tx1, Jetson Tx2, Jetson Xavier NX, Jetson AGX Xavier, etc. are all supported.

See below for DepthAI running on a Jetson Tx2 I have on my desk:




Installing for NVIDIA Jetson and Xavier is now the same set of instructions as Ubuntu. See here and following the standard Ubuntu instructions.

Also don't forget about the udev rules after you have that set up. And make sure to unplug and replug your depthai after having run the following commands (this allows Linux to execute the modification of the USB rules).

```
echo 'SUBSYSTEM=="usb", ATTRS{idVendor}=="03e7", MODE="0666"' | sudo tee /etc/udev/
↪rules.d/80-movidius.rules
sudo udevadm control --reload-rules && sudo udevadm trigger
```

### 2.1.12 Can I use multiple DepthAI with one host?

Yes. DepthAI is architected to put as-little-as-possible burden on the host. So even with a Raspberry Pi you can run a handful of DepthAI with the Pi and not burden the Pi CPU.

See here for instructions on how to do so.

### 2.1.13 Is DepthAI OpenVINO Compatible?

Yes. DepthAI is fully compatible with OpenVINO 2020.1, 2020.2, 2020.3, 2020.4, 2021.1 and 2021.2.

### 2.1.14 Can I train my own Models for DepthAI?

Yes.

We have a tutorial around Google Colab notebooks you can even use for this. See here

### 2.1.15 Do I need Depth data to train my own custom Model for DepthAI?

No.

That's the beauty of DepthAI. It takes standard object detectors (2D, pixel space) and fuses these neural networks with stereo disparity depth to give you 3D results in physical space.

Now, could you train a model to take advantage of depth information? Yes, and it would likely be even more accurate than the 2D version. To do so, record all the streams (left, right, and color) and retrain on all of those (which would require modifying the front-end of say MobileNet-SSD to allow 5 layers instead of 3 (1 for each grayscale, 3 for the color R, G, B).

### 2.1.16 If I train my own network, which Neural Operations are supported by DepthAI?

See the `VPU` section here.

Anything that's supported there under `VPU` will work on DepthAI. It's worth noting that we haven't tested all of these permutations though.

### 2.1.17 What network backbones are supported on DepthAI?

All the networks listed here are supported by DepthAI.

We haven't tested all of them though. So if you have a problem, contact us and we'll figure it out.

### 2.1.18 My Model Requires Pre-Processing (normalization, for example). How do I do that in DepthAI?

The OpenVINO toolkit allows adding these pre-processing steps to your model, and then these steps are performed automatically by DepthAI. See here for how to take advantage of this.

For instance, to scale frame pixels to the range [0,1], consider adding the following parameters to the model optimizer: `--data_type=FP16 --scale_values [255,255,255]`

To scale to the range [-1, 1], mean values should be added, e.g. for mobilenet: `--scale_values [127.5, 127.5, 127.5] --mean_values [127.5, 127.5, 127.5]`

More model converting options here

### 2.1.19 Can I Run Multiple Neural Models in Parallel or in Series (or Both)?

Yes. The Gen2 Pipeline Builder is what allows you to do this. And we have several example implementations of parallel, series, and parallel+series in depthai-experiments repository. A notable example is the Gaze estimation example, here, which shows series and parallel all together in one example.

### 2.1.20 Can DepthAI do Arbitrary Crop, Resize, Thumbnail, etc.?

Yes, see here for ane example of how to do this, with WASD controls of a cropped section. And see here for extension of the cropping for non-rectangular crops, and warping those to be rectangular (which can be useful for OCR).

### 2.1.21 Can DepthAI Run Custom CV Code? Say CV Code From PyTorch?

Yes, although we have yet to personally do this. But folks in the community have. Rahul Ravikumar is one, and was quite nice to have written up the process on how to do this, see here. This code can then be run as a node in the Gen2 Pipeline Builder, to be paired with other CV nodes, neural inference, depth processing, etc. that are supported on the platform.

### 2.1.22 How do I Integrate DepthAI into Our Product?

How to integrate DepthAI/megaAI depends on whether the product you are building includes

1. a processor running an operating system (Linux, MacOS, or Windows) or
2. a microcontroller (MCU) with no operating system (or an RTOS like FreeRTOS) or
3. no other processor or microcontroller (i.e. DepthAI is the only processor in the system).

We offer hardware to support all 3 use-cases, but firmware/software maturity varies across the 3 modes:

1. Using our Python API and/or C++ API (equal capabilities)
2. Using our C++ SPI API (see here),
3. Using our standalone flashing utility to flash a depthai application for standalone boot (as part of Pipeline Builder Gen2, leveraging our SBR Util here).

In all cases, DepthAI (and megaAI) are compatible with OpenVINO for neural models. The only thing that changes between the modalities is the communication (USB, Ethernet, SPI, etc.) and what (if any) other processor is involved.

#### Use-Case 1: DepthAI/megaAI are a co-processor to a processor running Linux, MacOS, or Windows.

In this case, DepthAI can be used in two modalities:

- NCS2 Mode (USB, *here*) - in this mode, the device appears as an NCS2 and the onboard cameras are not used and it's as if they don't exist. This mode is often use for initial prototyping, and in some cases, where a product simply needs an 'integrated NCS2' - accomplished by integrating a BW1099.
- DepthAI Mode (USB, using our USB API, here) - this uses the onboard cameras directly into the Myriad X, and boots the firmware over USB from a host processor running Linux, Mac, or Windows. This is the main use-case of DepthAI/megaAI when used with a host processor capable of running an operating system (e.g Raspberry Pi, i.MX8, etc.).

### Use-Case 2: Using DepthAI with a MicroController like ESP32, ATTiny8, etc.

In this case, DepthAI boot off of internal flash on the BW1099EMB and communicates over SPI, allowing DepthAI to be used with microcontroller such as the STM32, MSP430, ESP32, ATMega/Arduino, etc. We even have an embedded reference design for ESP32 (BW1092) available on our store. We will also be open-sourcing this design after it is fully verified (contact us if you would like the design files before we open source it).

The code-base/API for this is in active development, and a pre-release/Alpha version is available here as of this writing.

### Use-Case 3: Using DepthAI as the Only Processor on a Device.

This will be supported through running microPython directly on the BW1099EMB as nodes in the Gen2 Pipeline Builder.

The microPython nodes are what will allow custom logic, driving I2C, SPI, GPIO, UART, etc. controls, allowing direct controls of actuators, direct reading of sensors, etc. from/to the pipeline of CV/AI functions. A target example is making an entire autonomous, visually-controlled robotic platform with DepthAI as the only processor in the system.

This is now initially implemented and usable in Alpha. Reach out to use if you'd like to try it.

### Hardware for Each Case:

- BW1099: USB boot. So it is intended for working with a host processor running Linux, Mac, or Windows and this host processor boots the BW1099 over USB
- BW1099EMB: USB boot or NOR-flash boot. This module can work with a host computer just like the BW1099, but also has a 128MB NOR flash built-in and boot switches onboard - so that it can be programmed to boot off of NOR flash instead of of USB. So this allows use of the DepthAI in pure-embedded applications where there is no operating system involved at all. So this module could be paired with an ATTiny8 for example, communicating over SPI, or an ESP32 like on the BW1092 (which comes with the BW1099EMB pre-installed).

### Getting Started with Development

Whether intending to use DepthAI with an *OS-capable host*, a *microcontroller over SPI* (in development), or *completely standalone* (in Alpha testing) - we recommend starting with either *NCS2 mode* or with the DepthAI USB API for prototype/test/etc. as it allows faster iteration/feedback on neural model performance/etc. And in particular, with NCS2 mode, all the images/video can be used directly from the host (so that you don't have to point the camera at the thing you want to test).

In DepthAI mode, theoretically anything that will run in NCS2 mode will run - but sometimes it needs host-side processing if it's a network we've never run before. And this work is usually not heavy lifting... for example we had never run semantic segmentation networks before via the DepthAI API (and therefore had no reference code for doing so), but despite this one of our users actually got it working in a day without our help (e.g here).

For common object detector formats (MobileNet-SSD, Tiny YOLO v1/2/3, etc.) there's effectively no work to go from NCS2 mode to DepthAI mode. You can just literally replace the classes in example MobileNet-SSD or Tiny YOLO examples we have. For example for Tiny YOLO v3, you can just change the labels from "mask", "no mask" and "no mask 2" to whatever your classes are from this example here and just change the blob file here to your blob file. And the same thing is true for MobileNet-SSD here.

### 2.1.23 What Hardware-Accelerated Capabilities Exist in DepthAI and/or megaAI?

#### Available in DepthAI API Today:

- Neural Inference (e.g. object detection, image classification, etc., including multi-stage inference, e.g. here and here)
- Stereo Depth (including median filtering) (e.g. here)
- Stereo Inference (with two-stage, e.g. here)
- 3D Object Localization (augmenting 2D object detectors with 3D position in meters, e.g. here and here)
- Object Tracking (e.g. here, including in 3D space)
- H.264 and H.265 Encoding (HEVC, 1080p & 4K Video, e.g. here)
- JPEG Encoding (e.g. here)
- MJPEG Encoding
- Warp/Dewarp (for RGB-depth alignment/etc.)
- Enhanced Disparity Depth Modes (Sub-Pixel, LR-Check, and Extended Disparity), here
- SPI Support, here
- Arbitrary crop/rescale/reformat and ROI return (e.g. here)
- Integrated Text Detection (e.g. here)
- Pipeline Builder Gen2 (arbitrary series/parallel combination of neural nets and CV functions, background here and API documentation is here).
- Lossless zoom (from 12MP full to 4K, 1080p, or 720p, here)

The above features are available in the Luxonis Pipeline Builder Gen2 which is now the main API for DepthAI. The Gen1 API is still supported, and can be accessed via the version switcher at the bottom left of this page. See below for in-progress additional functionality/flexibility which will be added as modular nodes to the Luxonis pipeline builder for DepthAI.

#### On our Roadmap (Most are in development/integration)

- Improved Stereo Neural Inference Support (here)
- microPython Support, here
- Feature Tracking (including IMU-assisted feature tracking, here)
- Integrated IMU Support (here)
- Motion Estimation (here)
- Background Subtraction (here)
- Edge Detection (here)
- Harris Filtering (here)
- AprilTags (PR here)
- OpenCL Support (supported through OpenVINO (here))

And see our Github project here to follow along with the progress of these implementations.

#### Pipeline Builder Gen2

The 2nd-generation DepthAI pipeline builder which incorporates all the feedback we learned from our first Generation API. It is now the mainline way to use DepthAI.

It allows multi-stage neural networks to be pieced together in conjunction with CV functions (such as motion estimation or Harris filtering) and logical rules, all of which run on DepthAI/megaAI/OAK without any load on the host.

### 2.1.24 Are CAD Files Available?

Yes.

The full designs (including source Altium files) for all the carrier boards are in our depthai-hardware Github

### 2.1.25 How to enable depthai to perceive closer distances

If the depth results for close-in objects look weird, this is likely because they are below the minimum depth-perception distance of DepthAI/OAK-D.

For DepthAI Onboard Cameras (BW1098OBC) and OAK-D, the standard-settings minimum depth is around 70cm.

This can be cut in 1/2 and 1/4 with the following options:

1. Change the resolution to 640x400, instead of the standard 1280x800.

Since the disparity-search of 96 is what limits the minimum depth, this means the minimum depth is now 1/2 of standard settings - 35cm instead of 70cm. To do this with the example script, run *python3 depthai_demo.py -monor 400 -s previewout metaout depth -bb*. In Gen1 software, this is the only option. But in Gen2, Extended Disparity can again cut this min depth in 1/2.

2. Enable Extended Disparity.

In Gen2, Extended Disparity is supported, which extends the disparity search to 192 pixels from the standard 96 pixels, thereby 1/2-ing the minimum depth, so making the minimum depth for BW1098OBC/OAK-D 35cm for 1280x800 resolution and around 19.6cm (limited by the focal distance of the grayscale cameras) for 640x400 resolution.

See these examples for how to enable Extended Disparity.

### 2.1.26 What are the Minimum Depths Visible by DepthAI?

There are two ways to use DepthAI for 3D object detection and/or using neural information to get real-time 3D position of features (e.g. facial landmarks):

1. Monocular Neural Inference fused with Stereo Depth
2. Stereo Neural Inference

### Monocular Neural Inference fused with Stereo Depth

OAK-D:

- ~ 70cm with standard disparity (1280x800 resolution)
- ~ 35cm with extended disparity (1280x800 resolution)
- ~ 35cm with 640x400 resolution
- ~ 19.6cm with extended disparity and 640x400 resolution

In this mode, the AI (object detection) is run on the left, right, or RGB camera, and the results are fused with stereo disparity depth, based on semi global matching (SGBM). The minimum depth is limited by the maximum disparity search, which is by default 96, but is extendable to 192 in extended disparity modes (see *Extended Disparity* below).

To calculate the minimum distance in this mode, use the following formula, where base_line_dist and min_distance are in meters [m]:

```
min_distance = focal_length * base_line_dist / 96
```

Where 96 is the standard maximum disparity search used by DepthAI and so for extended disparity (192 pixels), the minimum distance is:

```
min_distance = focal_length * base_line_dist / 192
```

For DepthAI, the HFOV of the the grayscale global shutter cameras is 73.5 degrees (this can be found on your board, see here, so the focal length is

```
focal_length = 1280/(2*tan(73.5/2/180*pi)) = 857.06
```

Calculation here (and for disparity depth data, the value is stored in `uint16`, where 0 is a special value, meaning that distance is unknown.)

### How Does DepthAI Calculate Disparity Depth?

DepthAI makes use of a combination of hardware-blocks (a semi-global-matching disparity (SGBM) hardware block) as well as accelerated vector processing code in the SHAVES of the Myriad X to produce the disparity depth. This block is accessible via the Gen2 Pipeline Builder system, with an example here.

The SGBM hardware-block can process up to 1280x800 pixels, this is its hardware limit. Using higher-resolution sensors is technically possible via downscaling. So for example, using the 12MP color camera with the 1280x800 grayscale camera is possible (and has been prototyped by some users with the Gen2 pipeline builder). Or 2x 12MP image sensors could be used for depth (theoretically). But in both cases, the image data needs to be either decimated down to 1280x800, or converted in some other way (e.g. selectively cropped/windowed).

### What Disparity Depth Modes are Supported?

1. Default (96-pixel disparity search)
2. Extended Disparity (192-pixel disparity search), *here*
3. Subpixel Disparity (32 sub-pixel steps), *here*
4. LR-Check Disparity, *here*

### Stereo Neural Inference

In this mode, the neural inference (object detection, landmark detection, etc.) is run on the left *and* right cameras to produce stereo inference results. Unlike monocular neural inference fused with stereo depth - there is no max disparity search limit - so the minimum distance is purely limited by the greater of (a) horizontal field of view (HFOV) of the stereo cameras themselves and (b) the hyperfocal distance of the cameras.

The hyperfocal distance of the global shutter synchronized stereo pair is 19.6cm. So objects closer than 19.6cm will appear out of focus. This is effectively the minimum distance for this mode of operation, as in most cases (except for very wide stereo baselines with the *BW1098FFC*), this **effective** minimum distance is higher than the **actual** minimum distance as a result of the stereo camera field of views. For example, the objects will be fully out of the field of view of both grayscale cameras when less than 5.25cm from the *BW1098OBC*), but that is closer than the hyperfocal distance of the grayscale cameras (which is 19.6cm), so the actual minimum depth is this hyperfocal distance.

Accordingly, to calculate the minimum distance for this mode of operation, use the following formula:

```
min_distance = max(tan((90-HFOV/2)*pi/2)*base_line_dist/2, 19.6)
```

This formula implements the maximum of the HFOV-imposed minimum distance, and 19.6cm, which is the hyperfocal-distance-imposed minimum distance.

### Onboard Camera Minimum Depths

Below are the minimum depth perception possible in the disparity depth and stereo neural inference modes.

### Monocular Neural Inference fused with Stereo Depth Mode

For DepthAI units with onboard cameras, this works out to the following minimum depths:

- DepthAI Raspberry Pi Compute Module Edition (*BW1097*) the minimum depth is **0.827** meters for full 1280x800 stereo resolution and **0.414** meters for 640x400 stereo resolution:

```
min_distance = 857.06.15 * 0.09 / 96 = 0.803 # m
```

calculation here

**OAK-D** and USB3C Onboard Camera Edition (*BW1098OBC*) is - **0.689** meters for standard disparity, - **0.345** meters for Extended Disparity (192 pixel) at 1280x800 resolution or standard disparity at 640x400 resolution, and - **0.196** meters for Extended Disparity at 640x400 resolution (this distance is limited by the focal distance of the cameras on OAK-D)

```
min_distance = 857.06*.075/96 = 0.669 # m
```

calculation here

### Stereo Neural Inference Mode

For DepthAI units with onboard cameras, all models (*BW1097* and *BW1098OBC*) are limited by the hyperfocal distance of the stereo cameras, so their minimum depth is **0.196** meters.

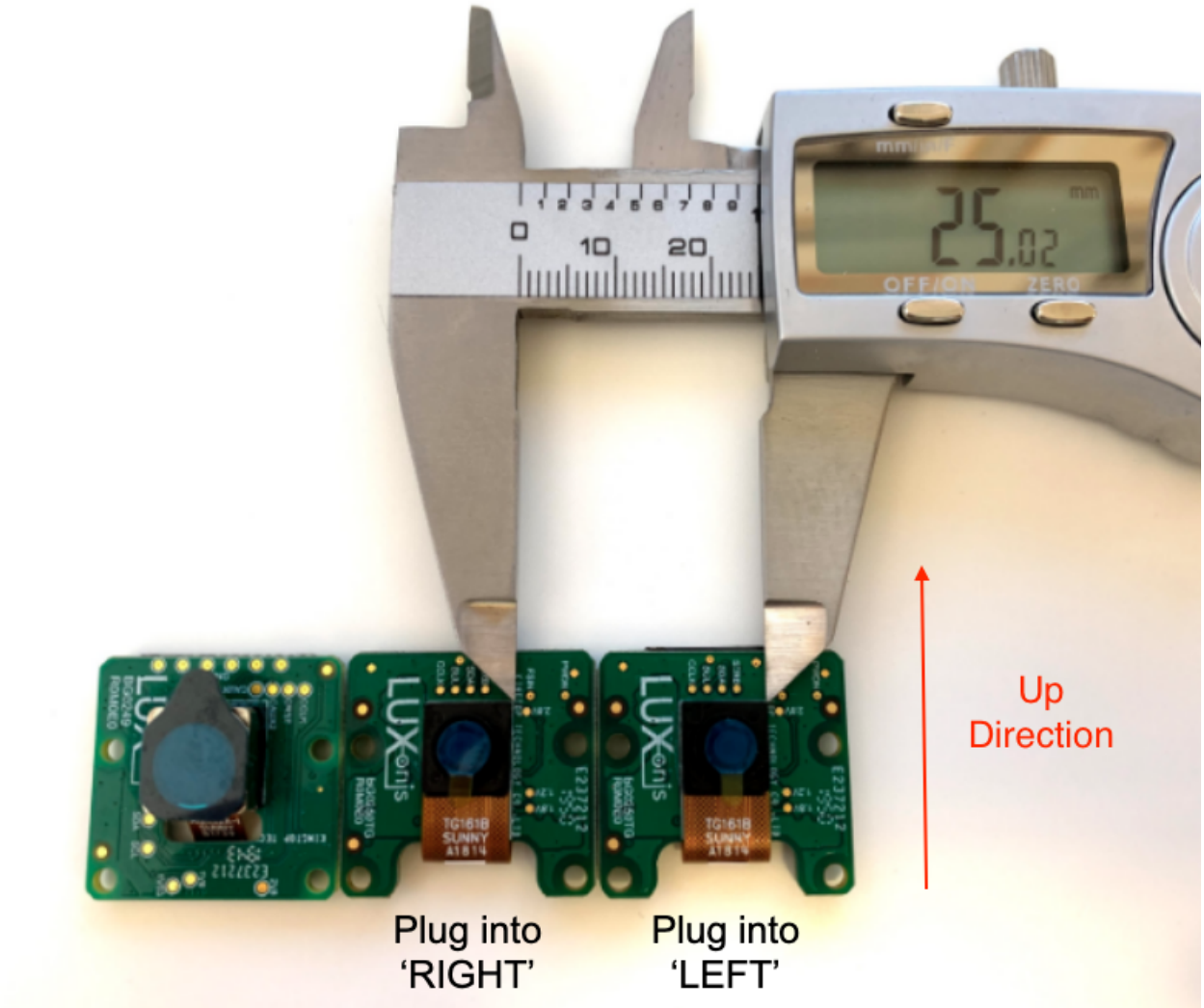
### Modular Camera Minimum Depths:

Below are the minimum depth perception possible in the disparity disparity depth and stereo neural inference modes.

### Monocular Neural Inference fused with Stereo Depth Mode

For DepthAI units which use modular cameras, the minimum baseline is 2.5cm (see image below) which means the minimum perceivable depth **0.229** meters for full 1280x800 resolution and **0.196** meters for 640x400 resolution (limited by the minimum focal distance of the grayscale cameras, as in stereo neural inference mode).

The minimum baseline is set simply by how close the two boards can be spaced before they physically interfere:




For any stereo baseline under 29cm, the minimum depth is dictated by the hyperfocal distance (the distance above which objects are in focus) of 19.6cm.

For stereo baselines wider than 29cm, the minimum depth is limited by the horizontal field of view (HFOV):

```
min_distance = tan((90-HFOV/2)*pi/2)*base_line_dist/2
```

### Extended Disparity Depth Mode

The `extended disparity` mode affords a closer minimum distance for the given baseline. This increases the maximum disparity search from 96 to 192. So this cuts the minimum perceivable distance in half (given that the minimum distance is now `focal_length * base_line_dist / 192` instead of `focal_length * base_line_dist / 96`).

- DepthAI Raspberry Pi Compute Module Edition (*BW1097*): **0.414** meters
- OAK-D and USB3C Onboard Camera Edition (*BW1098OBC*) is **0.345** meters
- Modular Cameras at Minimum Spacing (e.g. *BW1098FFC*) is **0.115** meters

See here for examples of how to use Extended Disparity Mode.

And for a bit more background as to how this mode is supported:

Extended disparity: allows detecting closer distance objects, without compromising on long distance values (integer disparity) by running the following flow. #. Computes disparity on the original size images (e.g. 1280x720) #. Computes disparity on 2x downscaled images (e.g. 640x360) #. Combines the two level disparities on Shave, effectively covering a total disparity range of 192 pixels (in relation to the original resolution).

### Left-Right Check Depth Mode

Left-Right Check, or LR-Check is used to remove incorrectly calculated disparity pixels due to occlusions at object borders (Left and Right camera views are slightly different). #. computes disparity by matching in R->L direction #. computes disparity by matching in L->R direction #. combines results from 1 and 2, running on Shave: each pixel d = disparity_LR(x,y) is compared with disparity_RL(x-d,y). If the difference is above a threshold, the pixel at (x,y) in final disparity map is invalidated.

To run LR-Check on DepthAI/OAK, use the example here.

## 2.1.27 What Are The Maximum Depths Visible by DepthAI?

The maximum depth perception for 3D object detection is is practically limited by how far the object detector (or other neural network) can detect what it's looking for. We've found that OpenVINO people detectors work to about 22 meters or so. But generally this distance will be limited by how far away the object detector can detect objects, and then after that, the minimum angle difference between the objects.

So if the object detector is not the limit, the maximum distance will be limited by the physics of the baseline and the number of pixels. So once an object is less than 0.056 degrees (which corresponds to 1 pixel difference) difference between one camera to the other, it is past the point where full-pixel disparity can be done. The formula used to calculate this distance is an approximation, but is as follows:

```
Dm = (baseline/2) * tan_d((90 - HFOV / HPixels)*pi/2)
```

For DepthAI HFOV = 73.5(+/-0.5) degrees, and HPixels = 1280. And for the BW1098OBC, the baseline is 7.5cm.

So using this formula for existing models the *theoretical* max distance is:

- BW1098OBC (OAK-D; 7.5cm baseline): 38.4 meters
- BW1097 (9cm baseline): 46 meters

- Custom baseline: Dm = (baseline/2) * tan_d(90 - 73.5 / 1280)

But these theoretical maximums are not achievable in the real-world, as the disparity matching is not perfect, nor are the optics, image sensor, etc., so the actual maximum depth will be application-specific depending on lighting, neural model, feature sizes, baselines, etc.

We also support subpixel depth mode, which extend this theoretical max, but again this will likely not be the -actual- limit of the max object detection distance, but rather the neural network itself will be. And this subpixel use will likely have application-specific benefits.

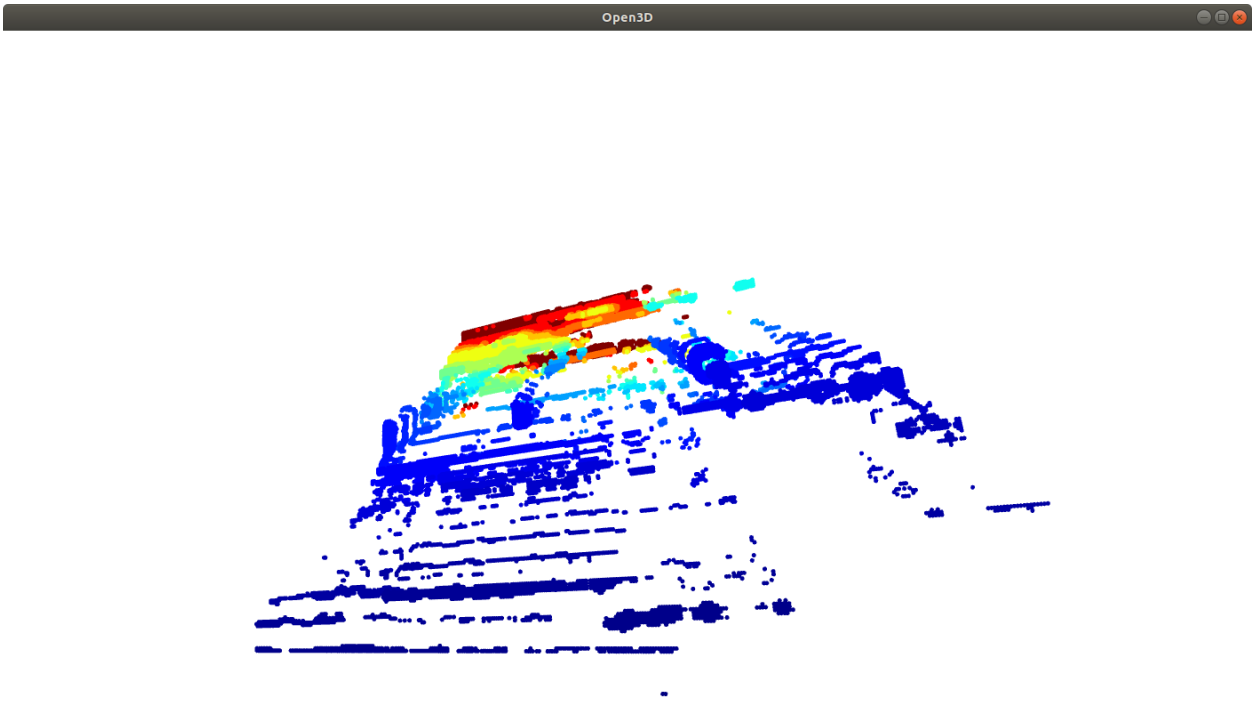
### Subpixel Disparity Depth Mode

Subpixel improves the precision and is especially useful for long range measurements. It also helps for better estimating surface normals (comparison of normal disparity vs. subpixel disparity is here).

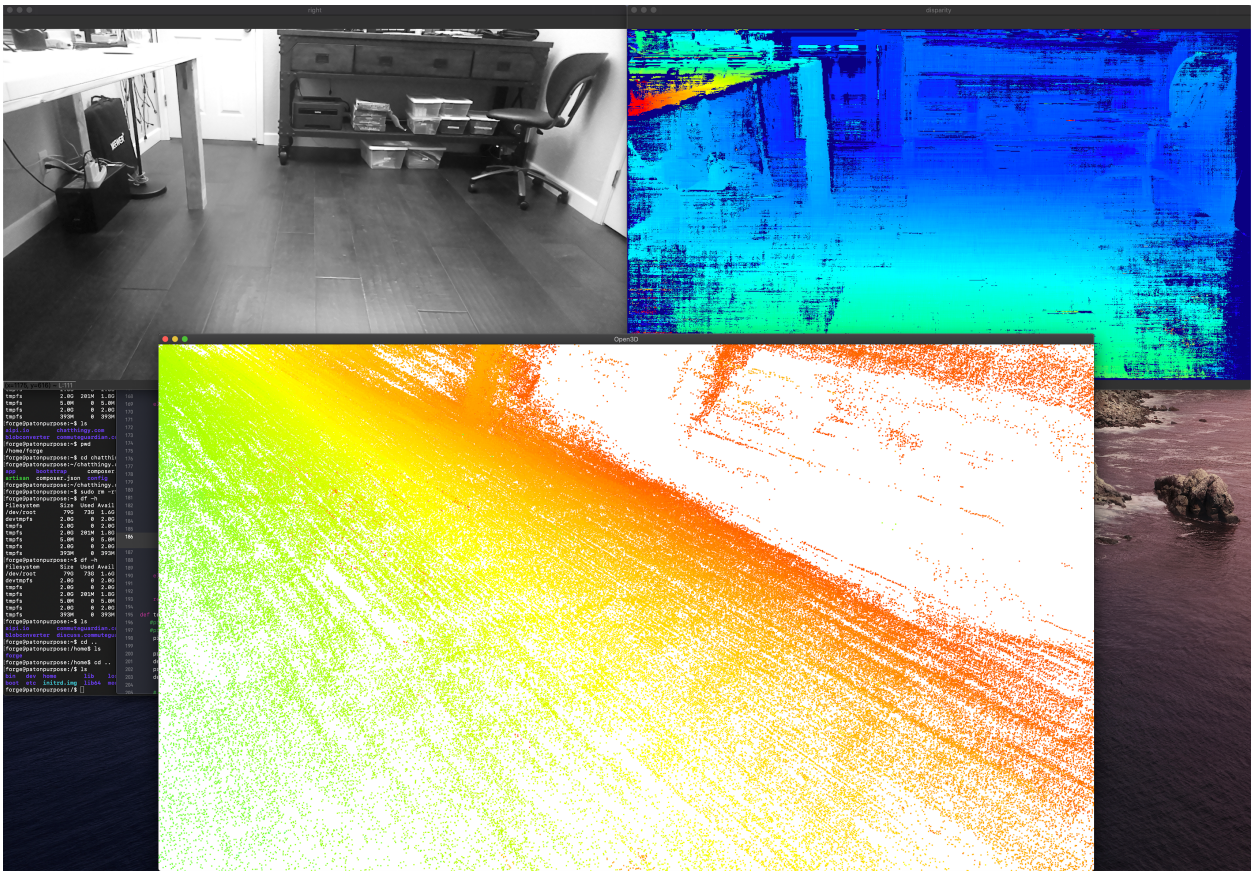
Beside the integer disparity output, the Stereo engine is programmed to dump to memory the cost volume, that is 96 bytes (disparities) per pixel, then software interpolation is done on Shave, resulting a final disparity with 5 fractional bits, resulting in significantly more granual depth steps (32 additional steps between the integer-pixel depth steps), and also theoretically, longer-distance depth viewing - as the maximum depth is no longer limited by a feature being a full integer pixel-step apart, but rather 1/32 of a pixel.

Examples of the difference in depth steps from standard disparity to subpixel disparity are shown below:

Standard Disparity (96 depth steps):




Subpixel Disparity (3,072 depth steps):



To run Subpixel on DepthAI/OAK, use the example here.

### 2.1.28 What Is the Format of the Depth Data in depth stream?

The output array is in uint16, so 0 to 65,535 with direct mapping to millimeters (mm).

So a value of 3,141 in the array is 3,141 mm, or 3.141 meters. So this whole array is the z-dimension of each pixel off of the camera plane, where the `center of the universe` is the camera marked `RIGHT`.

And the specific value of 65,535 is a special value, meaning an invalid disparity/depth result.

### 2.1.29 How Do I Calculate Depth from Disparity?

DepthAI does convert to depth onboard for both the `depth` stream and also for object detectors like MobileNet-SSD, YOLO, etc.

But we also allow the actual disparity results to be retrieved so that if you would like to use the disparity map directly, you can.

To calculate the depth map from the disparity map, it is (approximately) `baseline * focal / disparity`. Where the baseline is 7.5cm for BW1098OBC, 4.0cm for BW1092, and 9.0cm for BW1097, and the focal length is `883.15` (`focal_length = 1280/(2*tan(73.5/2/180*pi)) = 857.06`) for all current DepthAI models.

So for example, for a BW1092 (stereo baseline of 4.0cm), a disparity measurement of 60 is a depth of 58.8cm (`depth = 40 * 857.06 / 60 = 571 mm (0.571m)`).

### 2.1.30 How Do I Display Multiple Streams?

To specify which streams you would like displayed, use the `-s` option. For example for metadata (e.g. bounding box results from an object detector), the color stream (`previewout`), and for depth results (`depth`), use the following command:

```
python3 depthai_demo.py -s metaout previewout depth
```

**The available streams are:**

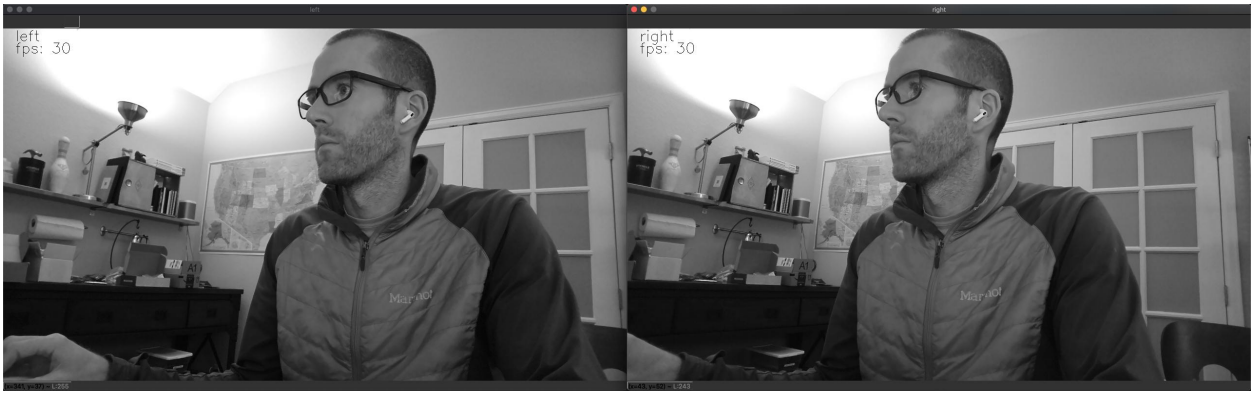
- `metaout` - Meta data results from the neural network
- `previewout` - Small preview stream from the color camera
- `color` - 4K color camera, biggest camera on the board with lens
- `left` - Left grayscale camera (marked *L* or *LEFT* on the board)
- `right` - Right grayscale camera (marked *R* or *RIGHT* on the board)
- `rectified_left` - Rectified left camera frames
- `rectified_right` - Rectified right camera frames
- `depth` - Depth in *uint16* (see here for the format.
- `disparity` - Raw disparity
- `disparity_color` - Disparity colorized on the host (`JET` colorized visualization of depth)
- `meta_d2h` - Device die temperature (max temp should be < 105C)
- `object_tracker` - Object tracker results

### Is It Possible to Have Access to the Raw Stereo Pair Stream on the Host?

Yes, to get the raw stereo pair stream on the host use the following command:

```
python3 depthai_demo.py -s left right
```

This will show the full RAW (uncompressed) 1280x720 stereo synchronized pair, as below:




## 2.1.31 How Do I Limit The FrameRate Per Stream?

So the simple way to select streams is to just use the `-s` option. But in some cases (say when you have a slow host or only USB2 connection -and- you want to display a lot of streams) it may be necessary to limit the frame rate of streams to not overwhelm the host/USB2 with too much data.

So to set streams to a specific frame rate to reduce the USB2 load and host load, simply specify the stream with `-s streamname` with a comma and FPS after the stream name like `-s streamname,FPS`.

So for limiting *depth* to 5 FPS, use the following command:

```
python3 depthai_demo.py -s depth,5
```

And this works equally for multiple streams:

```
python3 depthai_demo.py -s left,2 right,2 previewout depth,5
```

It's worth noting that the frame rate limiting works best for lower rates. So if you're say trying to hit 25FPS, it's best to just leave no frame-rate specified and let the system go to full 30FPS instead.

Specifying no limit will default to 30FPS.

One can also use the following override command structure, which allows you to set the frame rate per stream.

The following example sets the `depth` stream to 8 FPS and the `previewout` to 12 FPS:

```
python3 depthai_demo.py -co '{"streams": [{"name": "depth", "max_fps": 8.0},{"name":
↪"previewout", "max_fps": 12.0}]}'
```

You can pick/choose whatever streams you want, and their frame rate, but pasting in additional `{"name": "streamname", "max_fps": FPS}` into the expression above.

### 2.1.32 How do I Synchronize Streams and/or Meta Data (Neural Inference Results)

The `-sync` option is used to synchronize the neural inference results and the frames on which they were run. When this option is used, the device-side firmware makes a best effort to send metadata and frames in order of metadata first, immediately followed by the corresponding image.

When running heavier stereo neural inference, particularly with high host load, this system can break down, and there are two options which can keep synchronization:

1. Reduce the frame rate of the cameras running the inference to the speed of the neural inference itself, or just below it.
2. Or pull the timestamps or sequence numbers from the results (frames or metadata) and match them on the host.

#### Reducing the Camera Frame Rate

In the case of neural models which cannot be executed at the full 30FPS, this can cause lack of synchronization, particularly if stereo neural inference is being run using these models in parallel on the left and right grayscale image sensors.

A simple/easy way to regain synchronization is to reduce the frame rate to match, or be just below, the frame rate of the neural inference. This can be accomplished via the command line with the using `-rgbf` and `-monof` commands.

So for example to run a default model with both the RGB and both grayscale cameras set to 24FPS, use the following command:

```
./depthai_demo.py -rgbf 24 -monof 24 -sync
```

#### Synchronizing on the host

The two methods `FrameMetadata.getTimestamp()` and `FrameMetadata.getSequenceNum()` can be used to guarantee the synchronization on host side.

The NNPackets and DataPackets are being sent separately from device side, and get into individual queues per stream on host side. The function `CNNPipeline.get_available_nnet_and_data_packets()` returns what's available in the queues at the moment the function is called (it could be that just one NN packet is unread, or just one frame packet).

With the `-sync` CLI option from depthai.py, we are doing a best effort on the device side (i.e. on the Myriad X) to synchronize NN and previewout, and send them in order: first the NN packet is being sent (and in depthai.py it gets saved as the latest), then the previewout frame is being sent (and when received in depthai_demo.py, the latest saved NN data is overlaid on).

In most cases this works well, but there is a risk (especially under high system load on host side), that the packets may still get desynchronized, as the queues are handled by different threads (in the C++ library).

So in that case, `getMetadata().getTimestamp()` returns the device time (in seconds, as float) and is also used in the stereo calibration script to synchronize the Left and Right frames.

The timestamp corresponds to the moment the frames are captured from the camera, and is forwarded through the pipeline. And the method `getMetadata().getSequenceNum()` returns an incrementing number per camera frame. The same number is associated to the NN packet, so it could be an easier option to use, rather than comparing timestamps. The NN packet and Data packet sequence numbers should match.

Also, the left and right cameras will both have the same sequence number (timestamps will not be precisely the same, but few microseconds apart – that's because the timestamp is assigned separately to each from different interrupt handlers. But the cameras are started at the same time using an I2C broadcast write, and also use the same MCLK source, so shouldn't drift).

In this case we also need to check the camera source of the NN and Data packets. Currently, depthai.py uses getMetadata().getCameraName() for this purpose, that returns a string: `rgb`, `left` or `right` .

It is also possible to use `getMetadata().getInstanceNum()`, that returns a number: 0, 1 or 2 , respectively.

### 2.1.33 How do I Record (or Encode) Video with DepthAI?

DepthAI suppots h.264 and h.265 (HEVC) and JPEG encoding directly itself - without any host support. The *depthai_demo.py* script shows and example of how to access this functionality.

In Gen2 (current main line), see our encoding examples:

- RGB and Mono Encoding in parallel with MobileNetSSDv2, here.
- RGB and Mono Encoding in parallel with MobileNetSSDv2 and stereo depth, here.
- RGB, and both left/right camera encoding at maximum resolution and frame-rate, here.

Alternatively, to leverage this functionality from the depthai_demo.py (Gen1 API) command line, use the *-v* (or *–video*) command line argument as below:

```
python3 depthai_demo.py -v [path/to/video.h264]
```

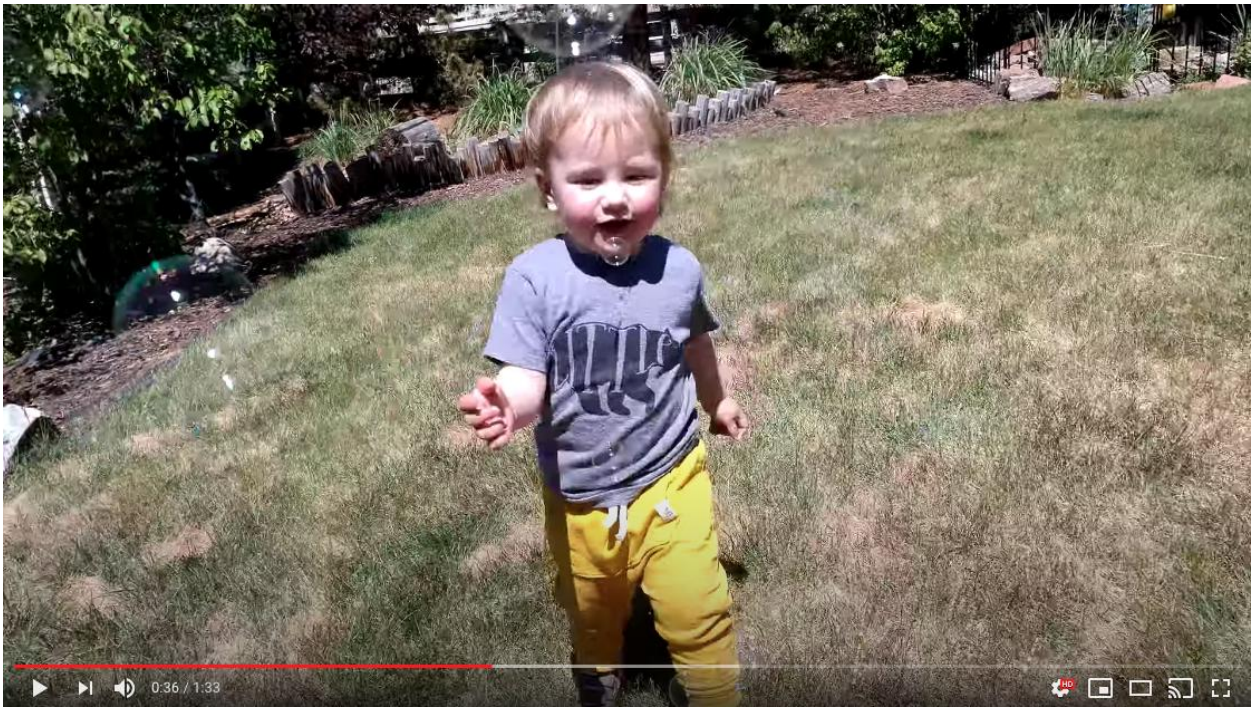
To then play the video in mp4/mkv format use the following muxing command:

```
ffmpeg -frame rate 30 -i [path/to/video.h264] -c copy [outputfile.mp4/mkv]
```

By default there are keyframes every 1 second which resolve the previous issues with traversing the video as well as provide the capability to start recording anytime (worst case 1 second of video is lost if just missed the keyframe)

When running `depthai_demo.py`, one can record a JPEG of the current frame by hitting `c` on the keyboard.

An example video encoded on DepthAI BW1097 (Raspberry Pi Compute Module Edition) is below. All DepthAI and megaAI units have the same 4K color camera, so will have equivalent performance to the video below.



### Gen1 Video Encoding Options

Additional options can be configured in the video encoding system by adding a `video_config` section to the JSON configuration file for the DepthAI pipeline builder, here, an example of which is here.

```
config = {
  ...
  'video_config':
  {
      'rateCtrlMode': 'cbr', # Options: 'cbr' / 'vbr' (constant bit rate or variable bit rate)
      'profile': 'h265_main', # Options: 'h264_baseline' / 'h264_main' / 'h264_high' / 'h265_main'
      'bitrate': 8000000, # When using CBR
      'maxBitrate': 8000000, # When using CBR
      'keyframeFrequency': 30, # In number of frames
      'numBFrames': 0,
      'quality': 80 # (0 - 100%) When using VBR
  }
  ...
}
```

The options above are all current options exposed for video encoding and not all must be set.

If the `video_config` member is **NOT** present in `config` dictionary then default is used: H264_HIGH, constant bit rate 8500 Kbps, key frame every 30 frames (once per second), `num B frames: 0`.

## 2.1.34 What are the Capabilities of the Video Encoder on DepthAI?

The max total encoding for h.264 and h.265 has 3 limits: - 4096 pixel max width for a frame. - Maximum pixels per second of 248 MegaPixel/second. - Maximum of 3 parallel encoding streams

The JPEG encoder is capable of 16384x8192 resolution at 500Mpixel/second.

Note the processing resources of the encoder are shared between H.26x and JPEG and both the width and height should be a multiple of 8 (which is usually the case with standard resolutions).

## 2.1.35 What Is The Stream Latency?

When implementing robotic or mechatronic systems it is often quite useful to know how long it takes from light hitting an image sensor to when the results are available to a user, the `photon-to-results` latency.

So the following results are an approximation of this `photon-to-results` latency, and are likely an over-estimate as we tested by actually seeing when results were updated on a monitor, and the monitor itself has some latency, so the results below are likely overestimated by whatever the latency of the monitor is that we used during the test. And we have also since done several optimizations since these measurements, so the latency could be quite a bit lower than these.

Table 5: Worst-case estimates of stream latency

| measured | requested | avg latency, ms |
|---|---|---|
| left | left | 100 |
| left | left, right | 100 |
| left | left, right, depth | 100 |
| left | left, right, depth, `metaout`, `previewout` | 100 |
| `previewout` | `previewout` | 65 |
| `previewout` | `metaout`, `previewout` | 100 |
| `previewout` | left, right, depth, `metaout`, `previewout` | 100 |
| `metaout` | `metaout` | 300 |
| `metaout` | `metaout`, `previewout` | 300 |
| `metaout` | left, right, depth, `metaout`, `previewout` | 300 |

### 2.1.36 Is it Possible to Use the RGB camera and/or the Stereo Pair as a Regular UVC Camera?

Yes, but currently not currently implemented in our API. It's on our roadmap, here

The `why` of our DepthAI API provides more flexibility in formats (unencoded, encoded, metadata, processing, frame-rate, etc.) and already works on any operating system (see here). So what we plan to do is to support UVC as part of our Gen2 Pipeline builder, so you can build a complex spatial AI/CV pipeline and then have the UVC endpoints output the results, so that DepthAI could then work on any system without drivers. For our embedded variants, this could then be flashed to the device so that the whole pipeline will automatically run on boot-up and show up to a computer a UVC device (a webcam).

Theoretically we can implement support for 3 UVC endpoints (so showing up as 3 UVC cameras), on for each of the 3 cameras.

We've prototyped 2x w/ internal proof of concept (but grayscale) but have not yet tried 3 but it would probably work. We could support a UVC stream per camera if it is of interest.

So if you would like this functionality please feel subscribe to the Github feature request here.

And in the meantime, if you would like to use depthai as a standard UVC camera, it is possible to use V4L2 loopback device (and some users have informed us that they have done so), but linking the output of the depthai API config into this loopback device on the host. We do not yet have instructions on this, but will circle back if/when we do.

### 2.1.37 How Do I Force USB2 Mode?

USB2 Communication may be desirable if you'd like to use extra-long USB cables and don't need USB3 speeds.

To force USB2 mode, simply use the *-fusb2* (or *–force_usb2*) command line option as below:

```
python3 depthai_demo.py -fusb2
```

Note that if you would like to use DepthAI at distances that are even greater than what USB2 can handle, we do have DepthAI PoE variants coming, see here, which allow DepthAI to use up to a 328.1 foot (100 meter) cable for both data and power - at 1 gigabit per second (1gbps).

### 2.1.38 What is "NCS2 Mode"?

All variants of DepthAI/megaAI come supporting what we call 'NCS2 mode'. This allows megaAI and DepthAI to pretend to be an NCS2.

So in fact, if you power your unit, plug it into a computer, and follow the instructions/examples/etc. of an NCS2 with OpenVINO, DepthAI/megaAI will behave identically.

This allows you to try out examples from OpenVINO directly as if our hardware is an NCS2. This can be useful when experimenting with models which are designed to operate on objects/items that you may not have available locally/physically. It also allows running inference in programmatic ways for quality assurance, refining model performance, etc., as the images are pushed from the host, instead of pulled from the onboard camera in this mode.

DepthAI/megaAI will also support an additional host-communication mode in the Gen2 Pipeline Builder, which will be available in December 2020.

### 2.1.39 What Information is Stored on the DepthAI Boards

Initial Crowd Supply backers received boards which had literally nothing stored on them. All information was loaded from the host to the board. This includes the BW1097 (*BW1097*), which had the calibration stored on the included microSD card.

So each hardware model which has stereo cameras (e.g. *BW1097*, *BW1098FFC*, *BW1098OBC*, and *BW1094*) has the capability to store the calibration data and field-of-view, stereo baseline (`L-R distance`) and relative location of the color camera to the stereo cameras (`L-RGB distance`) as well as camera orientation (`L/R swapped`). To retrieve this information, simply run `python3 depthai_demo.py` and look for `EEPROM data:`.

Example of information pulled from a *BW1098OBC* is below:

```
EEPROM data: valid (v2)
  Board name     : BW1098OBC
  Board rev      : R0M0E0
  HFOV L/R       : 73.5 deg
  HFOV RGB       : 68.7938 deg
  L-R   distance : 7.5 cm
  L-RGB distance : 3.75 cm
  L/R swapped    : yes
  L/R crop region: top
  Calibration homography:
    1.002324,   -0.004016,   -0.552212,
    0.001249,    0.993829,   -1.710247,
    0.000008,   -0.000010,    1.000000,
```

Current (those April 2020 and after) DepthAI boards with on-board stereo cameras (*BW1097*, *BW1098OBC*, and BW1092) ship calibration and board parameters pre-programmed into DepthAI's onboard EEPROM.

### 2.1.40 Dual-Homography vs. Single-Homography Calibration

As a result of some great feedback/insight from the OpenCV Spatial AI Competition we discovered and implemented many useful features (summary here).

Among those was the discovery that a dual-homography approach, although mathematically equivalent to a single-homography (as you can collapse the two homographies into one) actually outperforms single-homography in real-world practice.

As a result, we switched our calibration system in September 2020 to use dual-homography instead of single homography. So any units produced after September 2020 include dual homography. Any units with single homography can be recalibrated (see *here*) to use this updated dual-homography calibration.

### 2.1.41 What is the Field of View of DepthAI and megaAI?

DepthAI and megaAI use the same 12MP RGB Camera module based on the IMX378.

- 12MP RGB Horizontal Field of View (HFOV): 68.7938 deg
- 1MP Global Shutter Grayscale Camera Horizontal Field of View (HFOV): 73.5 deg

### 2.1.42 How Do I Get Different Field of View or Lenses for DepthAI and megaAI?

ArduCam is in the process of making a variety of camera modules specifically for DepthAI and megaAI, including a variety of M12-mount options (so that the optics/view-angles/etc. are change-able by you the user).

- M12-Mount IMX477 here
- M12-Mount OV9282 here
- Fish-Eye OV9282 (for better SLAM) here
- Mechanical, Optical, and Electrical equivalent OV9282 module with visible and IR capability here
- Global-Shutter Color Camera (OV9782) with same intrinsics as OV9282 grayscale here
- Original request for this here
- C/CS-Mount IMX283 (1" diagonal sensor, which is huge) here

With these, there will be a variety of options for view angle, focal length, filtering (IR, no IR, NDVI, etc.) and image sensor formats.

### 2.1.43 What are the Highest Resolutions and Recording FPS Possible with DepthAI and megaAI?

MegaAI can be used to stream raw/uncompressed video with USB3. Gen1 USB3 is capable of 5gbps and Gen2 USB3 is capable of 10gbps. DepthAI and MegaAI are capable of both Gen1 and Gen2 USB3 - but not all USB3 hosts will support Gen2, so check your hosts specifications to see if Gen2 rates are possible.

| Resolution | USB3 Gen1 (5gbps) | USB3 Gen2 (10gbps) |
|---|---|---|
| 12MP (4056x3040) | 21.09fps (390MB/s) | 41.2fps (762MB/s) |
| 4K (3840x2160) | 30.01fps (373MB/s) | 60.0fps (746MB/s) |

DepthAI and megaAI can do h.264 and h.265 (HEVC) encoding on-device. The max resolution/rate is 4K at 30FPS. With the default encoding settings in DepthAI/megaAI, this brings the throughput down from 373MB/s (raw/unencoded 4K/30) to 3.125MB/s (h.265/HEVC at 25mbps bit rate). An example video encoded on DepthAI BW1097 (Raspberry Pi Compute Module Edition) is below:


It's worth noting that all DepthAI and megaAI products share the same color camera specs and encoding capabilities. So footage filmed on a DepthAI unit with the color camera will be identical to that taken with a megaAI unit.

**Encoded:**

- 12MP (4056x3040) : JPEG Pictures/Stills
- 4K (3840x2160) : 30.00fps (3.125MB/s)

### 2.1.44 How Much Compute Is Available? How Much Neural Compute is Available?

DepthAI and megaAI are built around the Intel Movidius [Myriad X](https://www.intel.com/content/www/us/en/products/details/processors/movidius-vpu.html). More details/background on this part are here and also here.

**A brief overview of the capabilities of DepthAI/megaAI hardware/compute capabilities:**

- Overall Compute: 4 Trillion Ops/sec (4 TOPS)
- Neural Compute Engines (2x total): 1.4 TOPS (neural compute only)
- 16x SHAVES: 1 TOPS available for additional neural compute or other CV functions (e.g. through OpenCL)
- 20+ dedicated hardware-accelerated computer vision blocks including disparity-depth, feature matching/tracking, optical flow, median filtering, Harris filtering, WARP/de-warp, h.264/h.265/JPEG/MJPEG encoding, motion estimation, etc.
- 500+ million pixels/second total processing (see max resolution and frame rates over USB *here*)
- 450 GB/sec memory bandwidth
- 512 MB LPDDR4 (contact us for 1GB LPDDR version if of interest)

### 2.1.45 What Auto-Focus Modes Are Supported? Is it Possible to Control Auto-Focus From the Host?

DepthAI and megaAI support continuous video autofocus ('2' below, where the system is constantly autonomously searching for the best focus) and also and `auto` mode ('1' below) which waits to focus until directed by the host. (PR which adds this functionality is here.)

Example usage is shown in `depthai_demo.py`. When running `python3 depthai_demo.py` the functionality can be used by keyboard command while the program is running:

- '1' to change autofocus mode to auto
  - 'f' to trigger autofocus
- '2' to change autofocus mode to continuous video

And you can see the reference DepthAI API call here

### 2.1.46 What is the Hyperfocal Distance of the Auto-Focus Color Camera?

The hyperfocal distance is important, as it's the distance beyond which everything is in good focus. Some refer to this as 'infinity focus' colloquially.

The 'hyperfocal distance' (H) of DepthAI/megaAI's color camera module is quite close because of it's f.no and focal length.

From WIKIPEDIA, here, the hyperfocal distance is as follows:

$$H = \frac{f^2}{Nc} + f$$

Where:

- f = 4.52mm (the 'effective focal length' of the camera module)
- N = 2.0 (+/- 5%, FWIW)
- c = C=0.00578mm (see here, someone spelling it out for the 1/2.3" format, which is the sensor format of the IMX378)

So H = (4.52mm)^2/(2.0 * 0.00578mm) + 4.52mm ~= 1,772mm, or **1.772 meters** (**5.8 feet**).

We are using the effective focal length, and since we're not optics experts, we're not 100% sure if this is appropriate here, but the total height of the color module is 6.05mm, so using that as a worst-case focal length, this still puts the hyperfocal distance at **10.4 feet**.

So what does this mean for your application?

Anything further than 10 feet away from DepthAI/megaAI will be in focus when the focus is set to 10 feet or beyond. In other words, as long as you don't have something closer than 10 feet which the camera is trying to focus on, everything 10 feet or beyond will be in focus.

### 2.1.47 Is it Possible to Control the Exposure and White Balance and Auto-Focus (3A) Settings of the RGB Camera From the Host?

#### Auto-Focus (AF)

See *here* for details on controlling auto-focus/focus.

#### Exposure (AE)

It is possible to set frame duration (us), exposure time (us), sensitivity (iso) via the API. And we have a small example for the color camera to show how to do this for the color camera, which is here: https://github.com/luxonis/depthai/pull/279

We are planning on making these controls more self-documenting (see here), but in the meantime, all of the available controls are here: https://github.com/luxonis/depthai-shared/blob/82435d4/include/depthai-shared/metadata/camera_control.hpp#L107

And for example to set an exposure time of 23.4ms, with the maximum sensitivity of 1600, use:

```
self.device.send_camera_control(
   depthai.CameraControl.CamId.RGB,
   depthai.CameraControl.Command.AE_MANUAL,
   "23400 1600 33333")
```

#### White Balance (AWB)

This will be implemented at the same time as exposure and will be included. AWB lock, AWB modes. We will post more information as we dig into this task.

### 2.1.48 What Are the Specifications of the Global Shutter Grayscale Cameras?

The stereo pair is composed of synchronized global shutter OV9282-based camera modules.

**Specifications:**

- Effective Focal Length (EFL): 2.55
- F-number (F.NO): 2.2 +/- 5%
- Field of View (FOV): - Diagonal (DFOV): 82.6(+/-0.5) deg. - Horizontal (HFOV): 73.5(+/-0.5) deg. - Vertical (VFOV): 50.0(+/-0.5) deg.
- Distortion: < 1%
- Lens Size: 1/4 inch
- Focusing: Fixed Focus, 0.196 meter (hyperfocal distance) to infinity
- Resolution: 1280 x 800 pixel
- Pixel Size: 3x3 micrometer (um)

### 2.1.49 Am I able to attached alternate lenses to the camera? What sort of mounting system? S mount? C mount?

The color camera on megaAI and DepthAI is a fully-integrated camera module, so the lens, auto-focus, auto-focus motor etc. are all self-contained and none of it is replaceable or serviceable. You'll see it's all very small. It's the same sort of camera you would find in a high-end smart phone.

So the recommended approach, if you'd like custom optics, say IR-capable, UV-capable, different field of view (FOV), etc. is to use the ArduCam M12 or CS mount series of OV9281 and/or IMX477 modules.

- IMX477 M12-Mount
- IMX477 CS-Mount
- OV9281 M12-Mount

Note that these are require an adapter (here), and *below* and this adapter connects to the RGB port of the BW1098FFC. It is possible to make other adapters such that more than one of these cameras could be used at a time, or to modify the open-source BW1098FFC to accept the ArduCam FFC directly, but these have not yet been made.

That said, we have seen users attach the same sort of optics that they would to smartphones to widen field of view, zoom, etc. The auto-focus seems to work appropriately through these adapters. For example a team member has tested the Occipital *Wide Vision Lens* here to work with both megaAI and DepthAI color cameras. (We have not yet tried on the grayscale cameras.)

Also, see *below* for using DepthAI FFC with the Raspberry Pi HQ Camera to enable use of C- and CS-mount lenses.

### 2.1.50 Can I Power DepthAI Completely from USB?

So USB3 (capable of 900mA) is capable of providing enough power for the DepthAI models. However, USB2 (capable of 500mA) is not. So on DepthAI models power is provided by the 5V barrel jack power to prevent situations where DepthAI is plugged into USB2 and intermittent behavior occurs because of insufficient power (i.e. brownout) of the USB2 supply.

To power your DepthAI completely from USB (assuming you are confident your port can provide enough power), you can use this USB-A to barrel-jack adapter cable here. And we often use DepthAI with this USB power bank here.

### 2.1.51 What is the Screw Mount Specification on OAK-1 and OAK-D?

It is the standard 1/4-20 "Tripod" mount used on most cameras. More information on this type of mount on Wikipedia here.

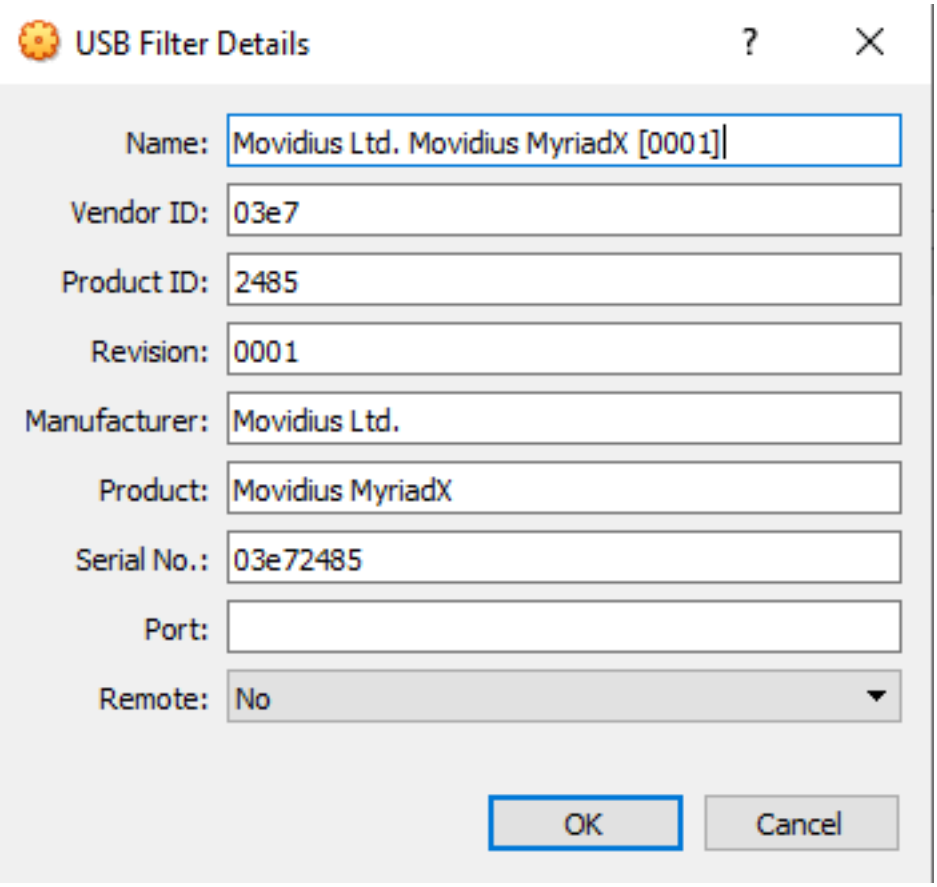
### 2.1.52 How to use DepthAI under VirtualBox

If you want to use VirtualBox to run the DepthAI source code, please make sure that you allow the VM to access the USB devices. Also, be aware that by default, it supports only USB 1.1 devices, and DepthAI operates in two stages:

1. For showing up when plugged in. We use this endpoint to load the firmware onto the device, which is a usb-boot technique. This device is USB2.
2. For running the actual code. This shows up after USB booting and is USB3.

In order to support the DepthAI modes, you need to download and install Oracle VM VirtualBox Extension Pack. Once this is installed, enable USB3 (xHCI) Controller in the USB settings.

Once this is done, you'll need to route the Myriad as USB device from Host to the VBox. This is the filter for depthai before it has booted, which is at that point a USB2 device:

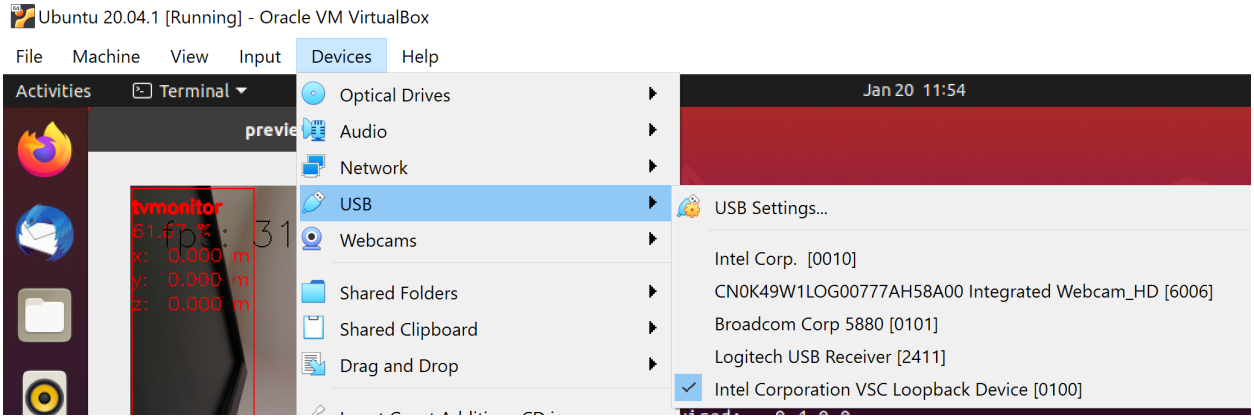



The last step is to add the USB Intel Loopback device. The depthai device boots its firmware over USB, and after it has booted, it shows up as a new device.

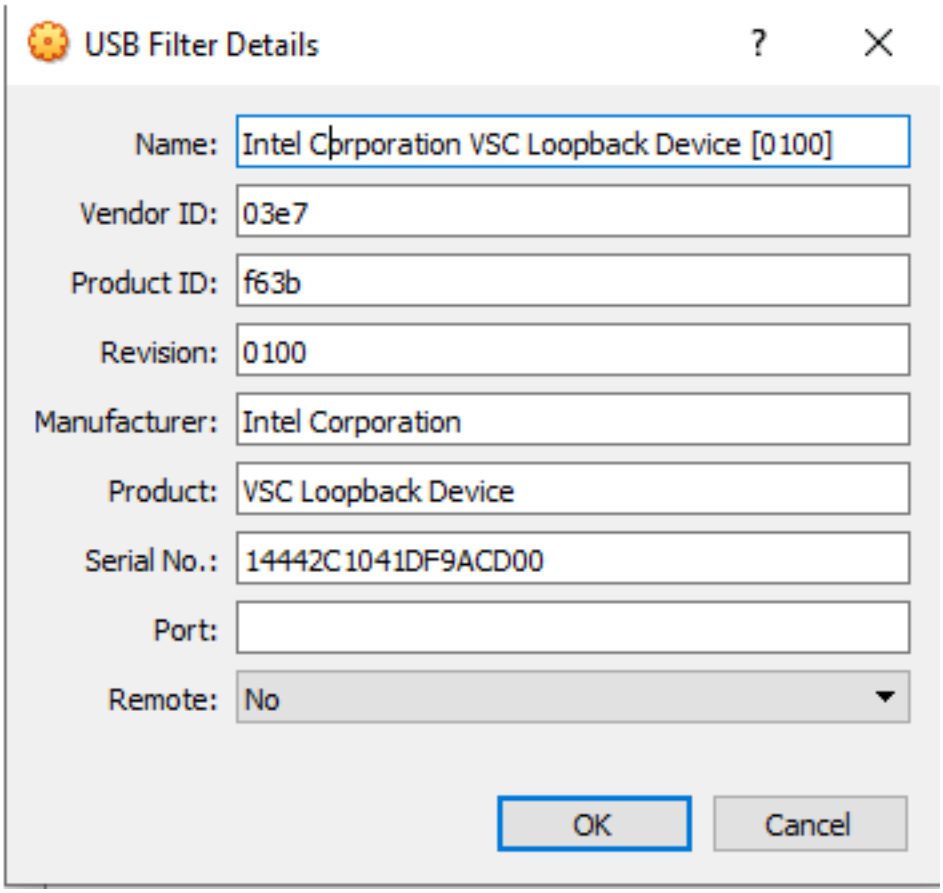
This device shows just up when the depthai/OAK is trying to reconnect (during runntime, so right after running a pipeline on depthai, such as *:bash: python3 depthai_demo.py*).

It might take a few tries to get this loopback device shown up and added, as you need to do this while depthai is trying to connect after a pipeline has been built (and so it has at that point now booted its internal firmware over USB2).

For enabling it only once, you can see the loopback device here (after the pipeline has been started):




And then for permanently enabling this pass-through to virtual box, enable this in setting below:




And then for each additional depthai/OAK device you would like to pass through, repeat just this last loopback settings step for each unit (as each unit will have its own unique ID).

### 2.1.53 What are the SHAVES?

The SHAVES are vector processors in DepthAI/OAK. The 2x NCE (neural compute engines) were architected for a slew of operations, but there are some that are not implemented. So the SHAVES take over these operations.

These SHAVES are also used for other things in the device, like handling reformatting of images, doing some ISP, etc.

So the higher the resolution, the more SHAVES are consumed for this.

- For 1080p, 13 SHAVES (of 16) are free for neural network stuff.
- For 4K sensor resolution, 10 SHAVES are available for neural operations.

There is an internal resource manager inside DepthAI firmware that coordinates the use of SHAVES, and warns if too many resources are requested by a given pipeline configuration.

### 2.1.54 How to increase NCE, SHAVES and CMX parameters?

If you want to specify how many Neural Compute Engines (NCE) to use, or how many SHAVE cores, or how many Connection MatriX blocks, you can do this with the DepthAI.

We have implemented the `-nce`, `-sh` and `-cmx` command line params in our example script. Just clone the DepthAI repository and do

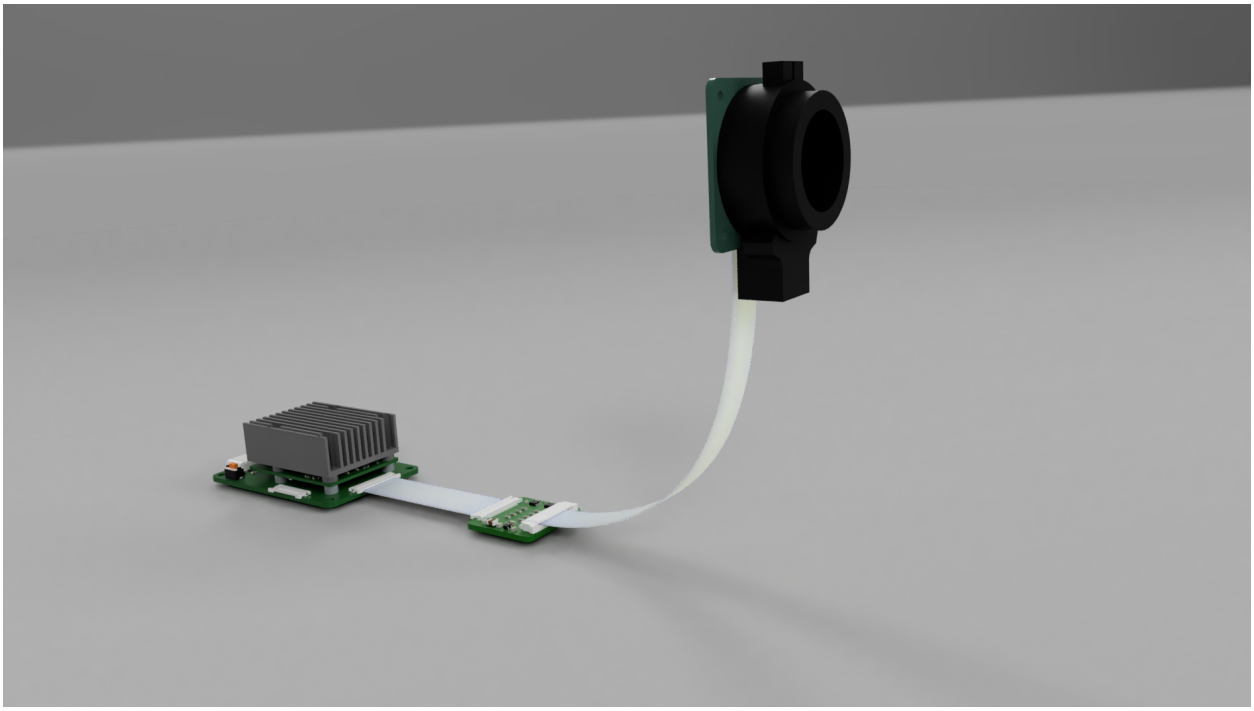
```
./depthai_demo.py -nce 2 -sh 14 -cmx 14
```

And it will run the default MobilenetSSD, compiled to use 2 NCEs, 14 SHAVEs and 14 CMXes. Note that these values **cannot be greater than the ones you can see above**, so you cannot use 15 SHAVEs or 3 NCEs. 14 is the limit for both SHAVE and CMX parameters, and 2 is the limit for NCE.

You can try it out yourself either by following local OpenVINO model conversion tutorial or by using our online Myriad X blob converter

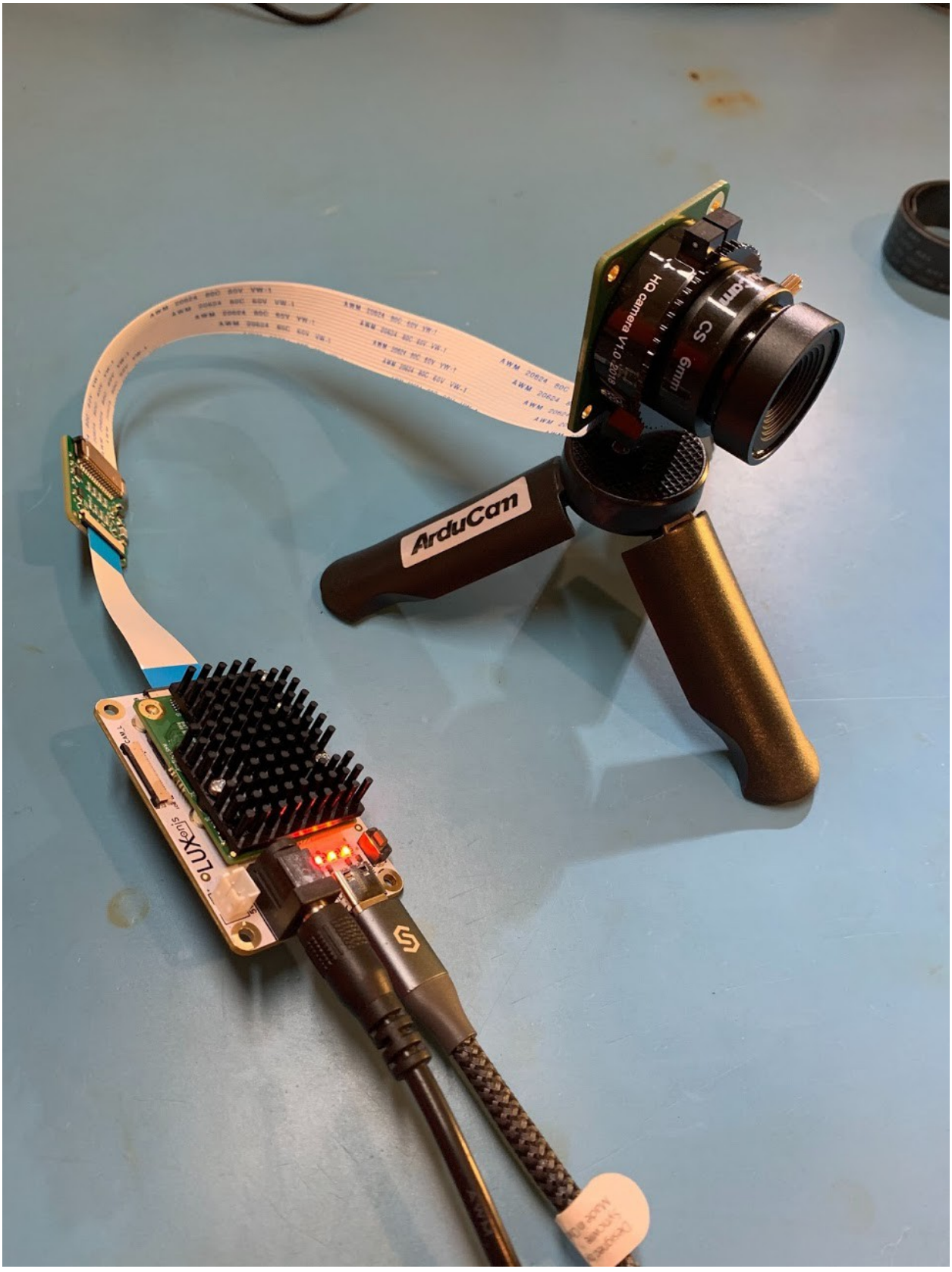
### 2.1.55 Can I Use DepthAI with the New Raspberry Pi HQ Camera?

DepthAI FFC Edition (BW1098FFC model here) also works via an adapter board with the Raspberry Pi HQ camera (IMX477 based), which then does work with a ton of C- and CS-mount lenses (see here). And see here for the adapter board for DepthAI FFC Edition.



This is a particularly interesting application of DepthAI, as it allows the Raspberry Pi HQ camera to be encoded to h.265 4K video (and 12MP stills) even with a Raspberry Pi 1 or *Raspberry Pi Zero* - because DepthAI does all the encoding onboard - so the Pi only receives a 3.125 MB/s encoded 4K h.265 stream instead of the otherwise 373 MB/s 4K RAW stream coming off the IMX477 directly (which is too much data for the Pi to handle, and is why the Pi when used with the Pi HQ camera directly, can only do 1080p video and not 4K video recording).

Here are some quick images and videos of it in use:

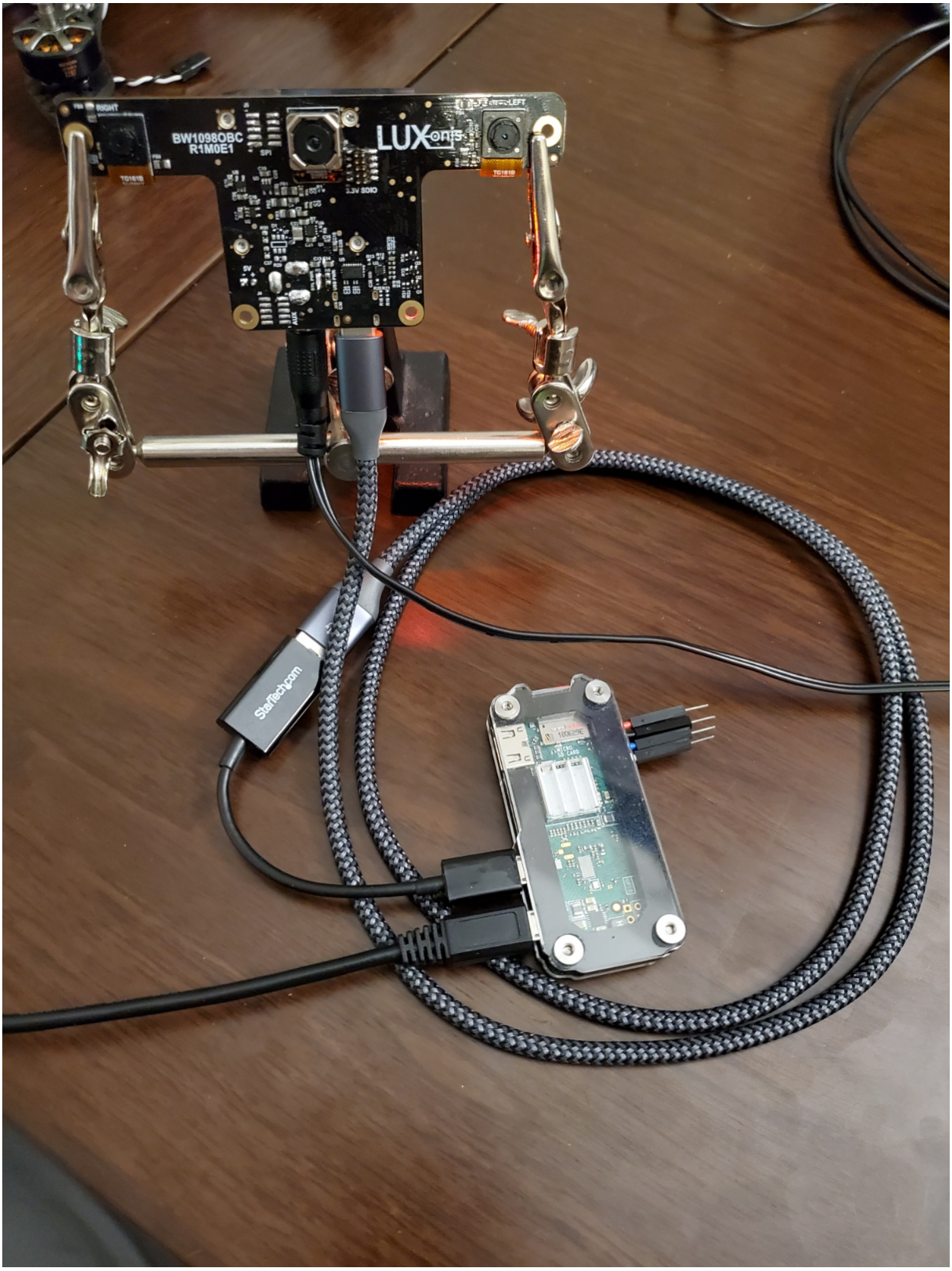

ArduCam
HQ camera V1.0 2018
6mm
CS

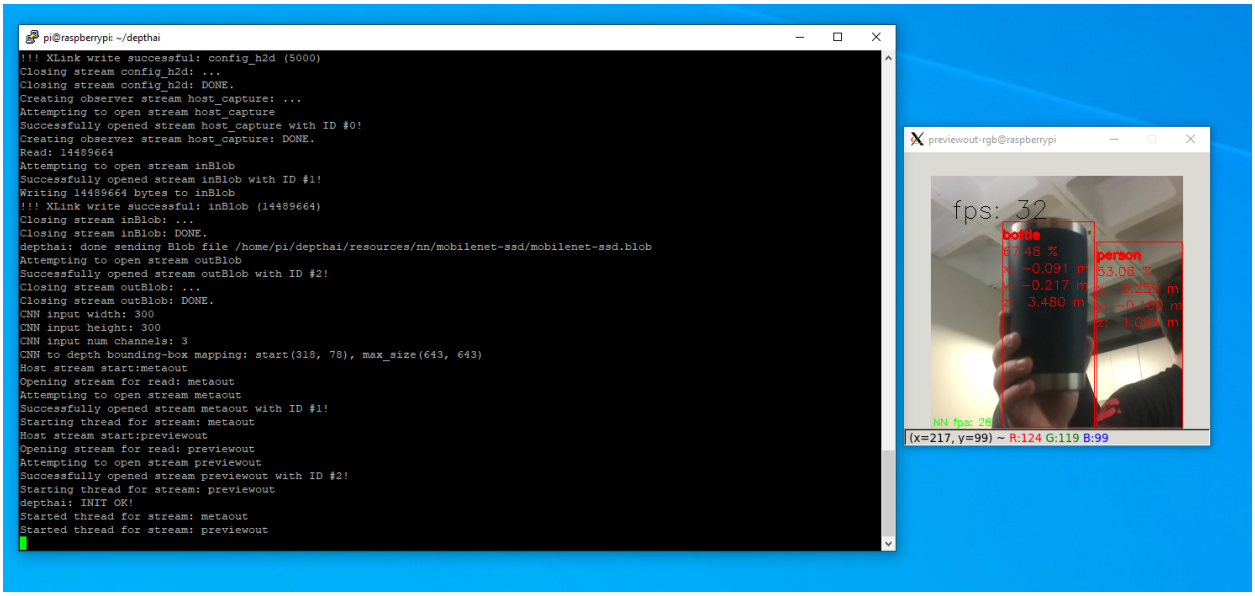

AmScope
ArduCam


You can buy this adapter kit for the DepthAI FFC Edition (BW1098FFC) here

### 2.1.56 Can I use DepthAI with Raspberry Pi Zero?

Yes, DepthAI is fully functional on it, you can see the example below:


BW1098OBC
R1M0E1
LUXonis
StarTech.com




Thanks to Connor Christie for his help building this setup!

And note that we now have a specific ARMv6 (Pi Zero) specific build of DepthAI.

### 2.1.57 How Much Power Does the DepthAI Raspberry Pi CME Consume?

The DepthAI Raspberry Pi Compute Module Edition (Raspberry Pi CME or BW1097 for short) consumes around 2.5W idle and 5.5W to 6W when DepthAI is running full-out.

- Idle: 2.5W (0.5A @ 5V)
- DepthAI Full-Out: 6W (1.2A @ 5V)

Below is a quick video showing this:


### 2.1.58 How Do I Get Shorter or Longer Flexible Flat Cables (FFC)?

- For the gray scale cameras, we use 0.5mm, 20-pin, same-side contact flex cables.
- For the RGB camera, we use a 0.5mm 26-pin, same-side contact flex cable.

One can purchase Molex's 15166 series FFCs directly to support shorter or longer lengths. Make sure you get **same-side** contacts, Molex calls this "**Type A**"

### 2.1.59 What are CSS MSS UPA and DSS Returned By meta_d2h?

- CSS: CPU SubSystem (main cores)
- MSS: Media SubSystem
- UPA: Microprocessor(UP) Array – Shaves
- DSS: DDR SubSystem

### 2.1.60 Where are the Github repositories? Is DepthAI Open Source?

DepthAI is an open-source platform across a variety of stacks, including hardware (electrical and mechanical), software, and machine-learning training using Google Colab.

See below for the pertinent Github repositories:

#### Overall

- https://github.com/luxonis/depthai-hardware - DepthAI hardware designs themselves.
- https://github.com/luxonis/depthai - Python demo and Examples
- https://github.com/luxonis/depthai-python - Python API
- https://github.com/luxonis/depthai-api - C++ Core and C++ API
- https://github.com/luxonis/depthai-ml-training - Online AI/ML training leveraging Google Colab (so it's free)
- https://github.com/luxonis/depthai-experiments - Experiments showing how to use DepthAI.

#### Embedded Use Case

- https://github.com/luxonis/depthai-experiments/tree/master/gen2-spi - user examples of SPI api and standalone mode.

The above examples include a few submodules of interest. You can read a bit more about them in their respective README files:

- https://github.com/luxonis/depthai-bootloader-shared - Bootloader source code which allows programming NOR flash of DepthAI to boot autonomously
- https://github.com/luxonis/depthai-spi-api - SPI interface library for Embedded (microcontroller) DepthAI application
- https://github.com/luxonis/esp32-spi-message-demo/tree/gen2_common_objdet - ESP32 Example applications for Embedded/ESP32 DepthAI use (e.g. with BW1092)

### 2.1.61 How Do I Build the C++ API?

Prebuilt binaries are available for Python bindings (or so called wheels).

We do not have prebuilt binaries for C++ core library.

One of the reasons is the vast number of different platforms and the second is that the library itself is quite lean so compiling along the other C++ source should not be a problem.

To compile the needed headers and a .dll follow this: https://github.com/luxonis/depthai-core/tree/main#building Under - And for the dynamic version of the library

You can optionally also install it into a desired directory by appending this cmake flag: cmake -DBUILD_SHARED_LIBS=ON -DCMAKE_INSTALL_PREFIX=[desired/installation/path] And then calling the install target cmake –build . –target install

This should result in the headers and the library being copied to that path.

Another option is integrating into your CMake project directly, for that see: https://github.com/luxonis/depthai-core-example

And a note on building for Windows: Windows does not use *libusb*, but rather uses Windows internal *winusb*.

### 2.1.62 Can I Use and IMU With DepthAI?

Yes, our BW1099 (here) has support to talk to IMUs. And we are in the process of making a future version of the BW1098OBC (as well as BW1092) which have built-in BNO085. We do not yet have support for this IMU in the DepthAI API, but we have done proof-of-concepts and will be making this a standard feature through the API.

### 2.1.63 Where are Product Brochures and/or Datasheets?

#### Brochures:

- Editions Summary here
- System on Module (BW1099) here
- USB3 Modular Cameras Edition (BW1098FFC) here
- USB3 Onboard Cameras Edition (BW1098OBC) here
- Raspberry Pi Compute Edition Module (BW1097) here
- Raspberry Pi HAT (BW1094) here
- megaAI (BW1093) here

#### Datasheets:

- DepthAI System on Module (BW1099) here
- PoE Modular Cameras Edition (BW2098FFC) here

### 2.1.64 How Much does LUX-D or OAK-D Weigh?

- 114.5 grams

### 2.1.65 How can I cite Luxonis products in publications?

If DepthAI and OAK-D products has been significantly used in your research and if you would like to acknowledge the DepthAI and OAK-D in your academic publication, we suggest citing them using the following bibtex format.

```
@misc{DepthAI,
title={ {DepthAI}: Embedded Machine learning and Computer vision api},
url={https://luxonis.com/},
note={Software available from luxonis.com},
author={luxonis},
year={2020},
}

@misc{OAK-D,
title={ {OAK-D}: Stereo camera with Edge AI},
url={https://luxonis.com/},
note={Stereo Camera with Edge AI capabilites from Luxonis and OpenCV},
author={luxonis},
year={2020},
}
```

### 2.1.66 How Do I Talk to an Engineer?

At Luxonis we firmly believe in the value of customers being able to communicate directly with our engineers. It helps our engineering efficiency. And it does so by making us make the things that matter, in the ways that matter (i.e. usability in the right ways) to solve real problems.

**As such, we have many mechanisms to allow direct communication:**

- Luxonis Community Discord. Use this for real-time communication with our engineers. We can even make dedicated channels for your project/effort public or private in here for discussions as needed.
- Luxonis Github. Feel free to make Github issues in any/all of the pertinent repositories with questions, feature requests, or issue reports. We usually respond within a couple ours (and often w/in a couple minutes). For a summary of our Github repositories, see *here*.
- discuss.luxonis.com. Use this for starting any public discussions, ideas, product requests, support requests etc. or generally to engage with the Luxonis Community. While you're there, check out this awesome visual-assistance device being made with DepthAI for the visually-impaired, here.

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

## 2.2 Support

Running into issues or have questions? We're here to help.

You can get help in a number of ways:

- Email support@luxonis.com
- Join our Discord Community for live assistance from us and developers like you
- Post a message to our forum
- Open an issue our DepthAI Github repository

### 2.2.1 Refunds and returns policy

At Luxonis, we are customer-focused. Our success is only possible if our customers believe in the value of our products. If for any reason you are not satisfied with your purchase, please let us know and we will make it right.

If you desire a refund, please contact support@luxonis.com with your order number and reason for the return. Refund requests within 60 days of the purchase date will be honored in full.

Shipping costs for returns within 60 days of purchase will be covered by Emporia Energy using USPS or Fedex. Shipping costs for returns after 60 days from the purchase date will be born by the customer.

If a return is initiated because of damaged, defective, or incorrect goods, Luxonis will provide a replacement order at no cost to the customer.

Refunds will be processed within 14 days after the product has been returned.

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

## 2.3 Troubleshooting

### 2.3.1 How can the start-up demo on the Raspberry Pi Compute Edition be disabled?

Delete the `autostart` file:

```
rm /home/pi/.config/autostart/runai.desktop
```

### 2.3.2 ImportError: No module named 'depthai'

This indicates that the `depthai` was not found by your python interpreter. There are a handful of reasons this can fail:

1. Is the Python API installed? Verify that it appears when you type:

   ```
   python3 -m pip list | grep depthai
   ```

2. Are you using a supported platform for your operating system? If not, you can always install from source:

   ```
   cat /etc/os-release
   ```

### 2.3.3 Why is the Camera Calibration running slow?

Poor photo conditions can dramatically impact the image processing time) during the camera calibration. Under normal conditions, it should take 1 second or less to find the chessboard corners per-image on an Raspberry Pi but this exceed 20 seconds per-image in poor conditions. Tips on setting up proper photo conditions:

- Ensure the checkerboard is not warped and is truly a flat surface. A high-quality option: print the checkerboard on a foam board.
- Reduce glare on the checkerboard (for example, ensure there are no light sources close to the board like a desk lamp).
- Reduce the amount of motion blur by trying to hold the checkerboard as still as possible.

### 2.3.4 Permission denied error

If `python3 -m pip install` fails with a `Permission denied` error, your user likely doesn't have permission to install packages in the system-wide path.

```
[Errno 13] Permission denied: '/usr/local/lib/python3.7/dist-packages/...'
```

Try installing in your user's home directory instead by adding the `--user` option. For example:

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

More information on Stackoverflow.

### 2.3.5 DepthAI does not show up under `/dev/video*` like web cameras do. Why?

The USB device enumeration could be checked with lsusb | grep 03e7 . It should print:

- `03e7:2485` after reset (boot loader running)
- `03e7:f63b` after the application was loaded

No `/dev/video*` nodes are created.

DepthAI implements the VSC (Vendor Specific Class) protocol and `libusb` is used for communication.

### 2.3.6 Intermittent Connectivity with Long (2 meter) USB3 Cables

- We've found that some hosts have trouble with USB3 + long cables (2 meter). It seems to have something to do with the USB controller on the host side.
- Other hosts have no problem at all and run for days (tested well over 3 days on some), even with long cables (tested w/ a total length of a bit over 8 feet). For example, all Apple computers we've tested with have never exhibited the problem.
- Ubuntu 16.04 has an independent USB3 issue, seemingly only on new machines though. We think this has to do w/ Ubuntu 16.04 being EOLed prior or around when these new machines having hit the market. For example, on this computer (here) has rampant USB3 disconnect issues under Ubuntu 16.04 (with a 1 meter cable), but has none under Ubuntu 18.04 (with a 1 meter cable).

So unfortunately we discovered this after we shipped with long USB3 cables (2 meter cables) with DepthAI units.

So if you have see this problem with your host, potentially 3 options:

1. Switch to a shorter USB3 cable (say 1 meter) will very likely make the problem disappear. These 1 meter (3.3 ft.) cables are a nice length and are now what we ship with DepthAI USB3 variants.
2. Force USB2 mode with `--force_usb2` option (examples below). This will allow use of the long cable still, and many DepthAI use cases do not necessitate USB3 communication bandwidth - USB2 is plenty.
3. Upgrade from Ubuntu 16.04 to Ubuntu 18.04.

#### Forcing USB2 Communication

If you are having trouble with communication with DepthAI/OAK, forcing USB2 can sometimes resolve the issue.

```
python3 depthai_demo.py --force_usb2
```

Or, the shorter form:

```
python3 depthai_demo.py -fusb2
```

For gen2, set the **usb2mode** to **True** when creating the device:

```
dai.Device(pipeline, usb2mode=True)
```

We've also seen an unconfirmed issue of running Ubuntu-compiled libraries on Linux Mint. If running on not Ubuntu 18.04/16.04 or Raspbian, please compile DepthAI from source.

### 2.3.7 Output from DepthAI keeps freezing

If the output from the device keeps freezing every few seconds, there may be a problem with the USB3 connection and forcing the device into USB2 mode could resolve this issue - instructions are in the chapter above.

When connection speed is USB2 (due to some hosts - Windows in particular - or USB controller/port/cable being USB2) - initialization of USB3-enabled firmware or streaming after a few frames may fail. The workaround here is to force the device to use the USB2-only firmware (mentioned in the chapter above).

### 2.3.8 Failed to boot the device: 1.3-ma2480, err code 3

This error often can occur if the udev rules are not set on Linux. This will coincide with `depthai: Error initializing xlink`.

To fix this, set the udev rules using the commands below, unplugging DepthAI and then plugging it back into USB afterwards.

```
echo 'SUBSYSTEM=="usb", ATTRS{idVendor}=="03e7", MODE="0666"' | sudo tee /etc/udev/
↪rules.d/80-movidius.rules
sudo udevadm control --reload-rules && sudo udevadm trigger
```

And in some cases, these were already set, but DepthAI was plugged in the entire time, so Linux could not reset the rules.

So make sure to unplug and then plug the DepthAI back in after having run these.

### 2.3.9 CTRL-C Is Not Stopping It!

If you are trying to kill a program with CTLR-C, and it's not working, try CTRL-instead. Usually this will work.

### 2.3.10 Is Your Raspberry Pi Locking Up?

The Raspberry Pi has a max limit of 1.2A across all its USB ports, and DepthAI/megaAI/OAK can take up to 1A (at max power, usually closer to 500mA).

So if you are seeing lockups, it could be that you are over this 1.2A limit as a result of the total power of the USB devices drawing from the Pi. Using a powered hub can prevent this, or powering fewer other things off of the Pi over USB.

### 2.3.11 "DLL load failed while importing cv2" on Windows

If you are seeing the following error after installing DepthAI for Windows:

```
(venv) C:\Users\Context\depthai>python depthai_demo.py
 Traceback (most recent call last):
   File "C:\Users\Context\depthai\depthai_demo.py", line 7, in <module>
     import cv2
   File "C:\Users\Context\depthai\venv\lib\site-packages\cv2\__init__.py", line 5, in
↪<module>
     from .cv2 import *
 ImportError: DLL load failed while importing cv2: The specified module could not be␣
↪found.
```

Then installing the Windows Media Feature Pack (here) is often the resolution, as Media Feature Pack must be installed for Windows 10 N editions.

(And more background from OpenCV directly is here)

### 2.3.12 `python3 depthai_demo.py` returns Illegal instruction

This so far has always meant there is a problem with the OpenCV install on the host (and not actually with the depthai library). To check this, run:

```
python3 -c "import cv2; import numpy as np; blank_image = np.zeros((500,500,3), np.
↪uint8); cv2.imshow('s', blank_image); cv2.waitKey(0)"
```

If a window is not displayed, or if you get the *:bash: Illegal instruction* result, this means there is a problem with the OpenCV install. The installation scripts here often will fix the OpenCV issues. But if they do not, running *:bash: python3 -m pip install opencv-python –force-reinstall* will often fix the OpenCV problem.

### 2.3.13 Neural network blob compiled with uncompatible openvino version

```
[NeuralNetwork(2)] [error] Neural network blob compiled with uncompatible openvino␣
↪version. Selected openvino version 2020.3. If you want to select an explicit␣
↪openvino version use: setOpenVINOVersion while creating pipeline
```

The reason for this error is that depthai can't resolve the OpenVINO version from the blob. The solution is simple, the user has to specify the OpenVINO version with which the blob was compiled (as mentioned in the error message):

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

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

## 2.4 Products

### 2.4.1 BW1092 - Three Camera ESP32 Board

The BW1092 is a modified version of the BW1098OBC with an added ESP32 System on Module (ESP32-WROOM-32). This board allows users to quickly prototype standalone embedded solutions.

Besides the standard DepthAI capabilites, the ESP32 gives users access to a lightweight processor with useful features such as Bluetooth, Bluetooth Low Energy (BTLE), and WiFi, and comes in a convenient FCC/CE-certified module.

### Requirements

- PC to program ESP32.

### What's in the box?

- BW1092
- BW1099EMB
- USB3C cable (6 in.)

### Setup

Install the Python API.

### Verify installation

We'll execute a DepthAI example Python script to ensure your setup is configured correctly. Follow these steps to test DepthAI:

1. Start a terminal session.
2. Clone the depthai example repository.

```
git clone https://github.com/luxonis/depthai.git
```

1. Access your local copy of depthai.

```
cd [depthai repo]
```

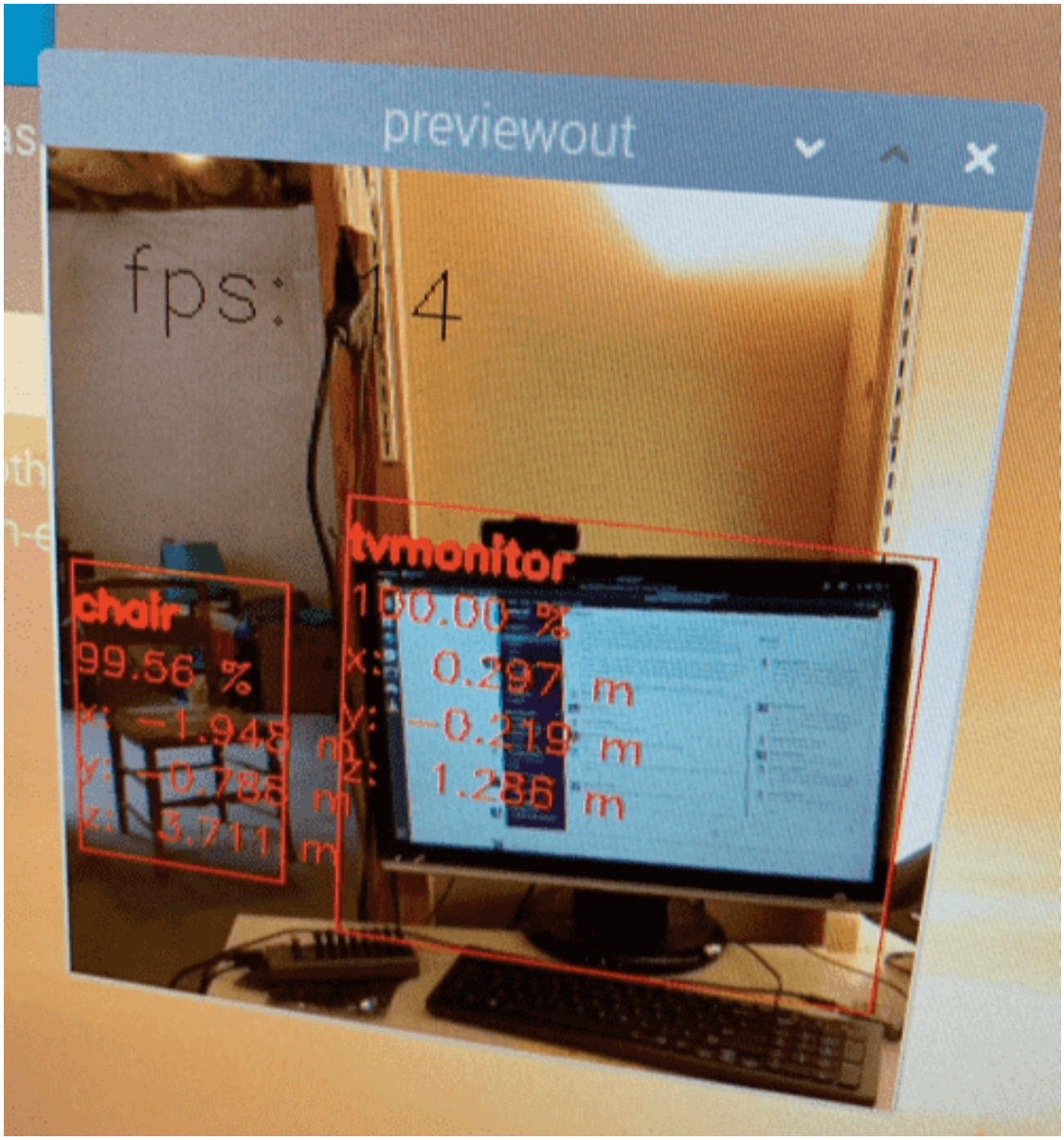
1. Install the example repository requirements.

```
python -m pip install -r requirements.txt
```

1. Run demo script.

```
python3 depthai_demo.py
```

The script launches a window, starts the cameras, and displays a video stream annotated with object localization metadata:




In the screenshot above, DepthAI identified a TV monitor (1.286 m from the camera) and a chair (3.711 m from the camera).

See the list of object labels in our pre-trained OpenVINO model tutorial.

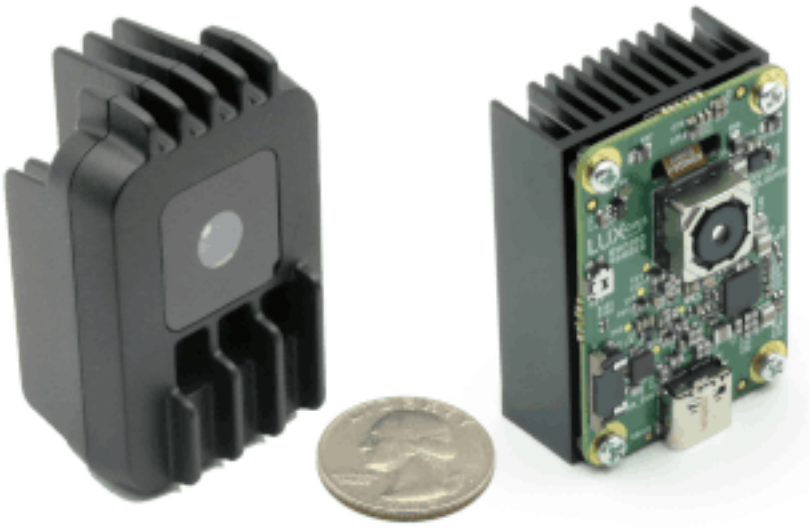
### Using the ESP32/SPI Interface

A number of examples that show how to communicate with the ESP32 over SPI can be found here:

https://github.com/luxonis/depthai-experiments/tree/master/gen2-spi

All of these examples consist of two parts: An application that runs on the ESP32 and an application that runs on the MyriadX using the gen2 Pipeline Builder. More information on how to set up both of these can be found in that github's README.

### 2.4.2 BW1093 - OAK-1 | MegaAI - 4K USB3 AI Camera



Use OAK-1/megaAI on your existing host. Since the AI/vision processing is done on the Myriad X, a typical desktop could handle tens of OAK-1/megaAI plugged in (the effective limit is how many USB ports the host can handle).

And since it can encode 1080p and 4K video (see *here*) you can now even save 4K video on a Pi Zero!

#### Requirements

- USB3C cable
- USB2 or USB3 port on the host

#### What's in the box?

- megaAI / OAK-1
- USB3C cable (3 ft.)
- Getting Started Card

#### Setup

Install the Python API.

#### Verify installation

We'll execute a DepthAI example Python script to ensure your setup is configured correctly. Follow these steps to test DepthAI:

1. Start a terminal session.
2. Clone the depthai example repository.

```
git clone https://github.com/luxonis/depthai.git
```

1. Access your local copy of depthai.

```
cd [depthai repo]
```

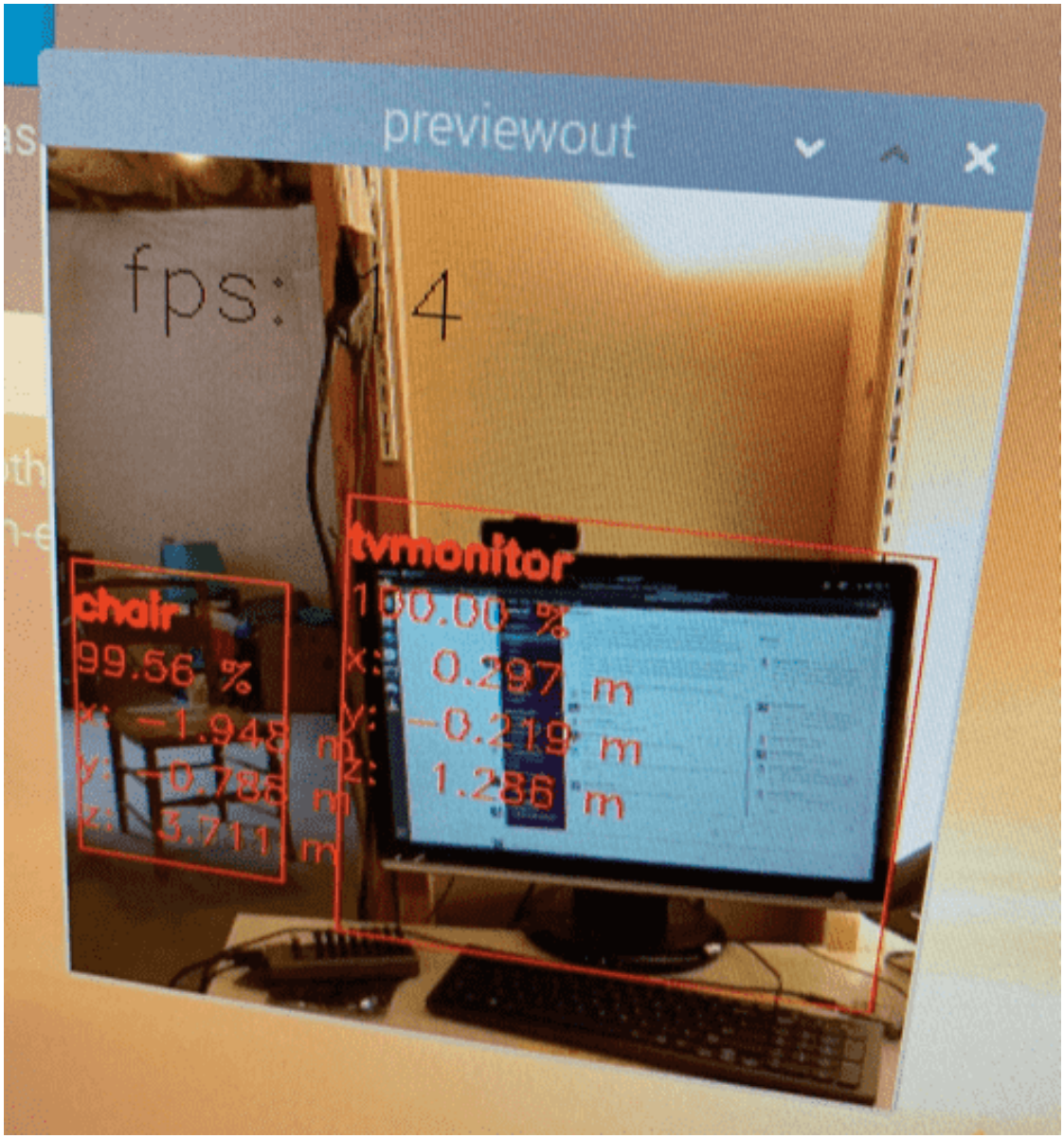
1. Install the example repository requirements.

```
python -m pip install -r requirements.txt
```

1. Run demo script.

```
python3 depthai_demo.py
```

The script launches a window, starts the cameras, and displays a video stream annotated with object localization metadata:




In the screenshot above, DepthAI identified a TV monitor (1.286 m from the camera) and a chair (3.711 m from the camera).

See the list of object labels in our pre-trained OpenVINO model tutorial.

### Camera Specs

Table 6: **Color Camera**

| | |
|---|---|
| **Shutter Type** | Rolling Shutter |
| **Image Sensor** | IMX378 |
| **Max Framerate** | 60fps |
| **H.265 Framerate** | 30fps |
| **Resolution** | 12MP (4056x3040 px/1.55um) |
| **Field of View** | 81° DFoV - 68.8° HFoV |
| **Lens Size** | 1/2.3 Inch |
| **Focus** | 8cm - ∞ (AutoFocus) |
| **F-number** | 2.0 |

### Compliance

Table 7: Compliance download & reference table

| | |
|---|---|
| megaAI / DepthAI BW1093 / OAK | `Declaration of conformity` |

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

## 2.4.3 BW1094 - RaspberryPi Hat

The Raspberry Pi HAT Edition allows using the Raspberry Pi you already have and passes through the Pi GPIO so that these are still accessible and usable in your system(s). Its modular cameras allow mounting to your platform where you need them, up to six inches away from the HAT.

- Mounts to Raspberry Pi as a HAT for easy integration
- All Raspberry Pi GPIO still accessible through pass-through header
- Flexible Camera Mounting with 6" flexible flat cables
- Includes three FFC Camera ports

### Requirements

- A RaspberryPi with an extended 40-pin GPIO Header.

### Board Layout

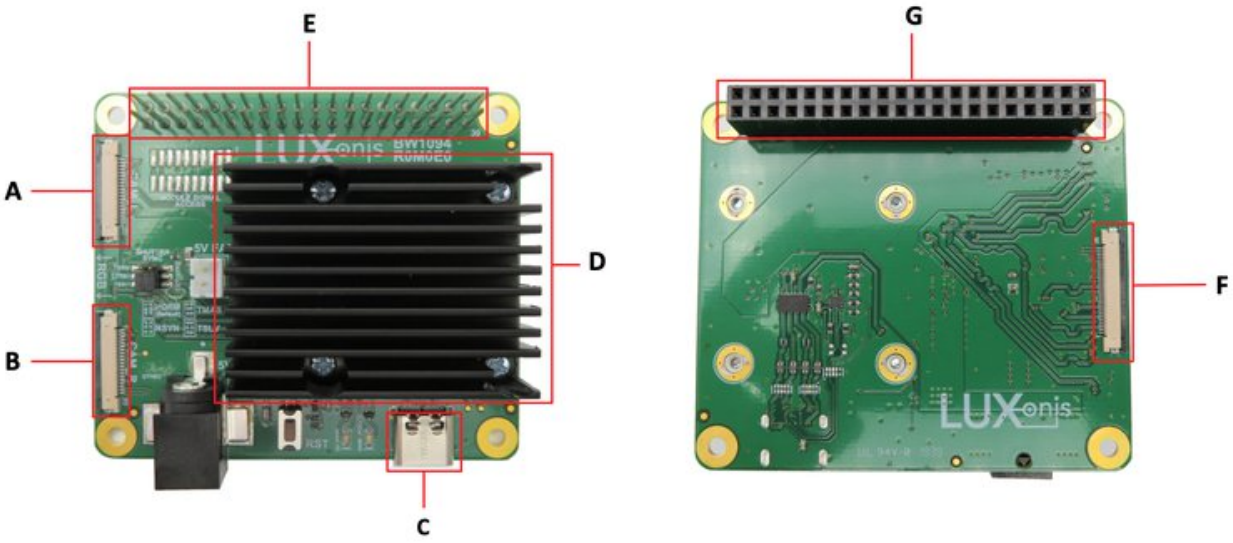



Table 8: Reference table

| | |
|---|---|
| **A.** Left Camera Port | **E.** Pass-through 40-Pin Raspberry Pi Header |
| **B.** Right Camera Port | **F.** Color Camera Port |
| **C.** USB 3.0 Type-C | **G.** 40-pin Raspberry Pi Header |
| **D.** DepthAI Module | |

### What's in the box?

- BW1094 Carrier Board
- Pre-flashed µSD card loaded with Raspbian 10 and DepthAI
- USB3C cable (6 in.)

### Setup

Follow the steps below to setup your DepthAI device.

1. **Power off your Raspberry Pi.**

   Safely power off your Raspberry Pi and unplug it from power.

2. **Insert the pre-flashed µSD card into your RPi.**

   The µSD card is pre-configured with Raspbian 10 and DepthAI.

3. **Mount the DepthAI RPi HAT.**

   Use the included hardware to mount the DepthAI RPi HAT to your Raspberry Pi.

4. **Reconnect your RPi power supply**

5. **Calibrate the cameras**.

   See *Calibration*

### Verify installation

We'll execute a DepthAI example Python script to ensure your setup is configured correctly. Follow these steps to test DepthAI:

1. Start a terminal session.
2. Clone the depthai example repository.

```
git clone https://github.com/luxonis/depthai.git
```

1. Access your local copy of depthai.

```
cd [depthai repo]
```

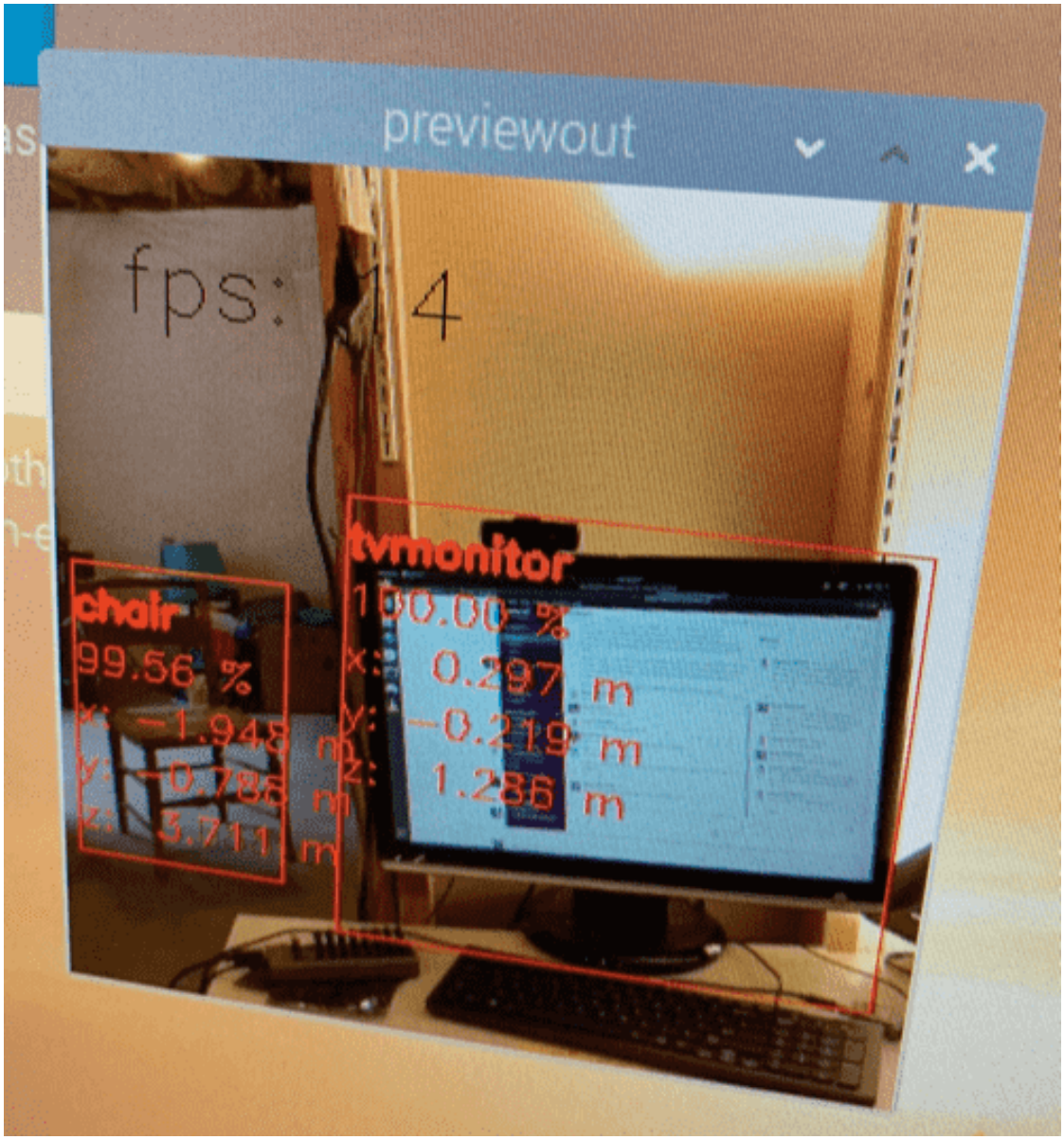
1. Install the example repository requirements.

```
python -m pip install -r requirements.txt
```

1. Run demo script.

```
python3 depthai_demo.py
```

   The script launches a window, starts the cameras, and displays a video stream annotated with object localization metadata:

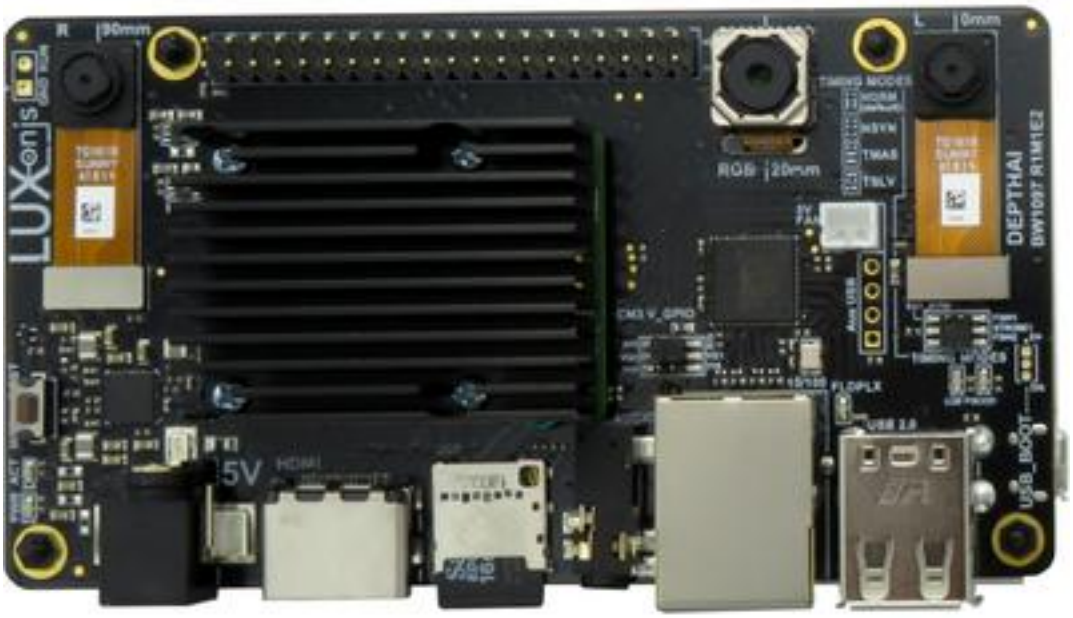



In the screenshot above, DepthAI identified a TV monitor (1.286 m from the camera) and a chair (3.711 m from the camera).

See the list of object labels in our pre-trained OpenVINO model tutorial.

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

### 2.4.4 BW1097 - RaspberryPi Compute Module




The Raspberry Pi Compute Module Edition comes with everything needed: pre-calibrated stereo cameras on-board with a 4K, 60 Hz color camera and a µSD card with Raspbian and DepthAI Python code automatically running on bootup. This allows using the power of DepthAI with literally no typing or even clicking: it just boots up doing its thing. Then you can modify the Python code with one-line changes, replacing the neural model for the objects you would like to localize.

- Built-in RaspberryPi Compute Module
- Three integrated cameras
- Complete system; everything you need is included

## Board Layout

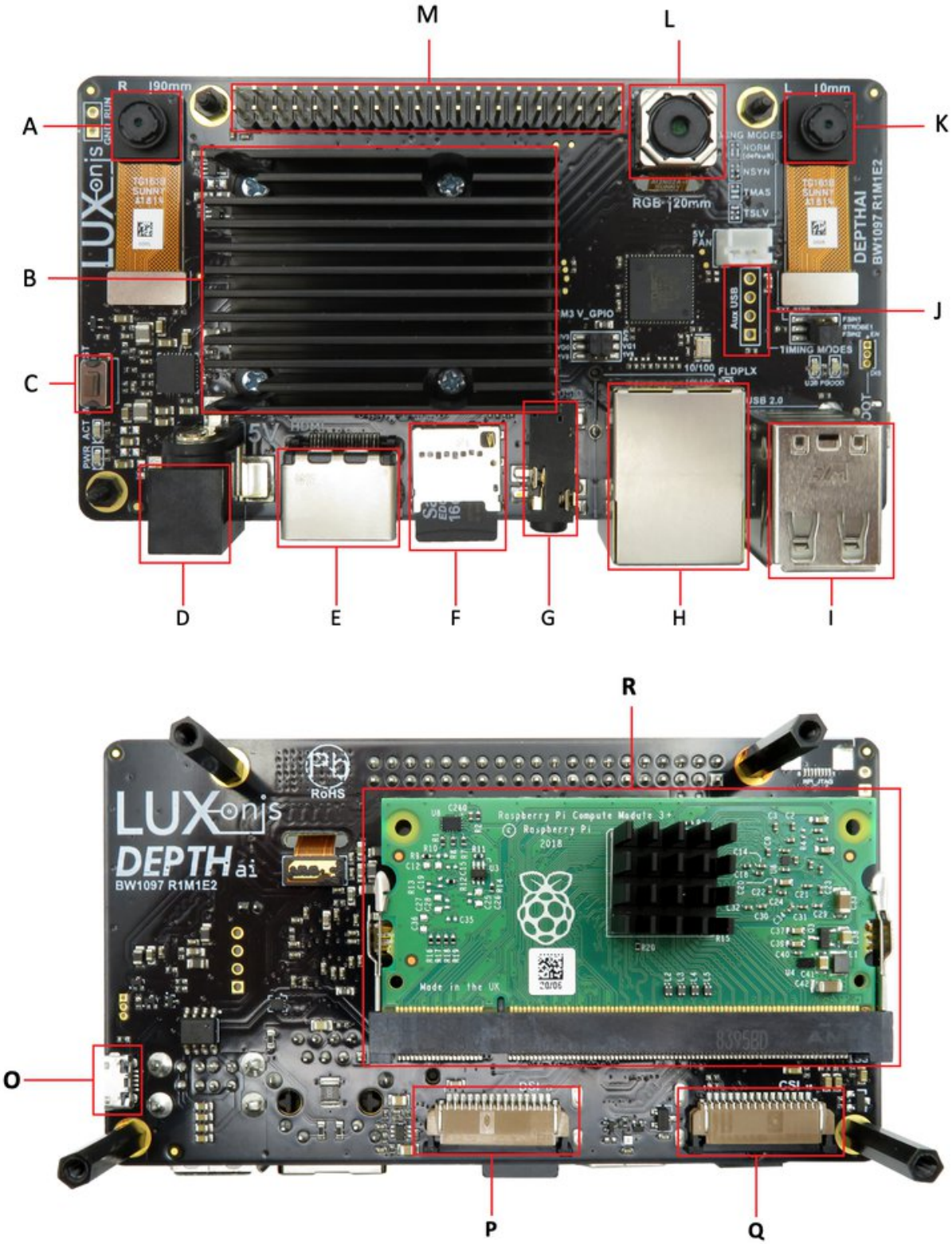


Table 9: Reference table

| | |
|---|---|
| **A.** 720p 120 Hz Global Shutter (Right) | **J.** 1x Solderable USB2.0 |
| **B.** DepthAI Module | **K.** 720p 120 Hz Global Shutter (Left) |
| **C.** DepthAI Reset Button | **L.** 4K 60 Hz Color |
| **D.** 5 V IN | **M.** RPi 40-Pin GPIO Header |
| **E.** HDMI | **O.** RPi USB-Boot |
| **F.** 16 GB µSD Card, Pre-configured | **P.** RPi Display Port |
| **G.** 3.5 mm Audio | **Q.** RPi Camera Port |
| **H.** Ethernet | **R.** Raspberry Pi Compute Module 3B+ |
| **I.** 2x USB2.0 | |

### What's in the box?

- BW1097 Carrier Board
- Pre-flashed µSD card loaded with Raspbian 10 and DepthAI
  - Default Password: `luxonis`
- WiFI USB dongle
- Power Supply

### Setup

To get started:

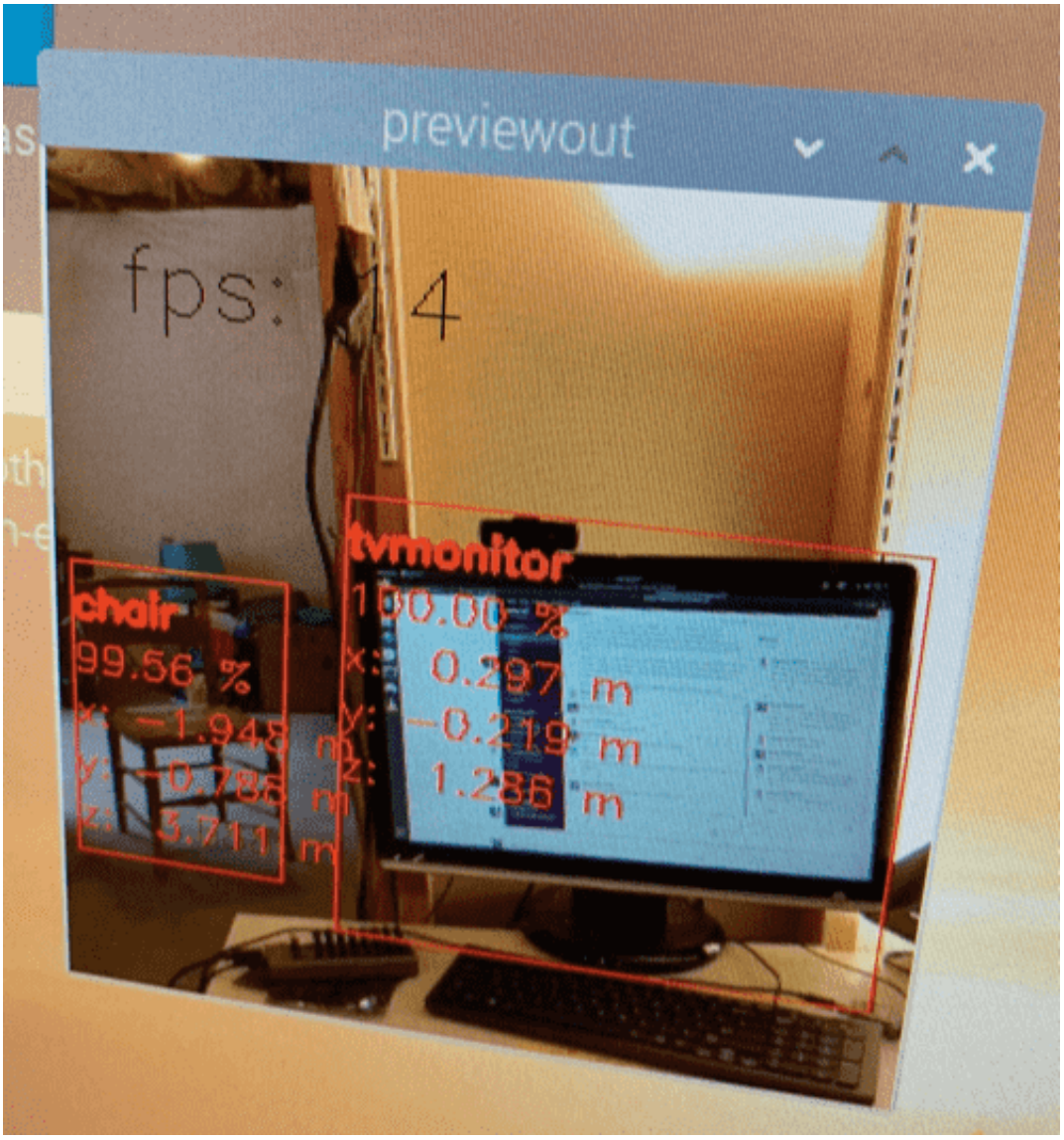
1. **Connect a display to the HDMI port.**

   Note that an HDMI cable is not included.

2. **Connect a keyboard and mouse via the USB port**

3. **Connect the power supply (included).**

   On boot, the Pi will run a Python demo script that displays a video stream annotated with object localization metadata:




In the screenshot above, DepthAI identified a tv monitor (1.286 m from the camera) and a chair (3.711 m from the camera). See the list of object labels on GitHub.

4. **Connect to the Internet.**

   Connect the Pi to the Internet to begin trying the DepthAI tutorials and examples.

   - **Connecting to a WiFi network**

     To connect to a WiFi network, use the included Linux-compatible USB WiFi dongle. The Pi should recognize the dongle and display available WiFi networks in the upper right corner of the Raspbian Desktop UI.

   - **Connecting to a network via Ethernet**

     The board includes an Ethernet port. Connecting an Ethernet cable to the port will enable Internet access.

#### [Optional] Using your own SD-Card

If you'd like to set up DetphAI on your own (say bigger) SD-Card, there are two options:

1. Download our pre-configured Raspbian image for the BW1097 (the Raspberry Pi Compute Module Edition), here: BW1097 Raspian Image. Then, after downloading, update the DepthAI firmware/software (by doing a git pull on the DepthAI code base checked out on the Desktop).
2. Set up your own Raspbian to your liking from say a fresh Raspbian download, and then use replace dt-blob.bin and config.txt in /boot with the following two files:
   - dt-blob.bin - For enabling the Pi MIPI display
   - config.txt - For enabling the 3.5mm headphone jack

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

### 2.4.5 BW1098FFC - USB3 with Modular Cameras


Use DepthAI on your existing host. Since the AI/vision processing is done on the Myriad X, a typical desktop could handle tens of DepthAIs plugged in (the effective limit is how many USB ports the host can handle).

### Requirements

- Ubuntu 18.04 or Raspbian 10
- Cameras
  - *Modular color camera*
  - *Stereo camera pair* (if depth is required)
- USB3C cable
- USB3C port on the host

### Board Layout

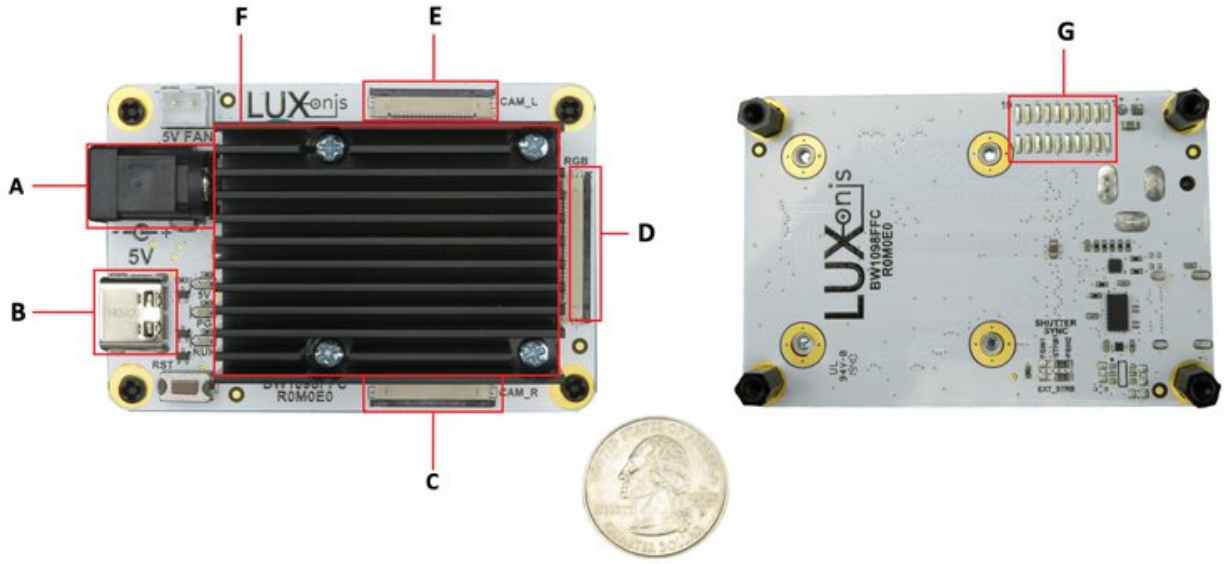



Table 10: Reference table

| | |
|---|---|
| **A.** 5V IN | **E.** Left Camera Port |
| **B.** USB3C | **F.** DepthAI Module |
| **C.** Right Camera Port | **G.** Myriad X GPIO Access |
| **D.** Color Camera Port | |

### What's in the box?

- BW1098FFC Carrier Board
- USB3C cable (3 ft.)
- Power Supply

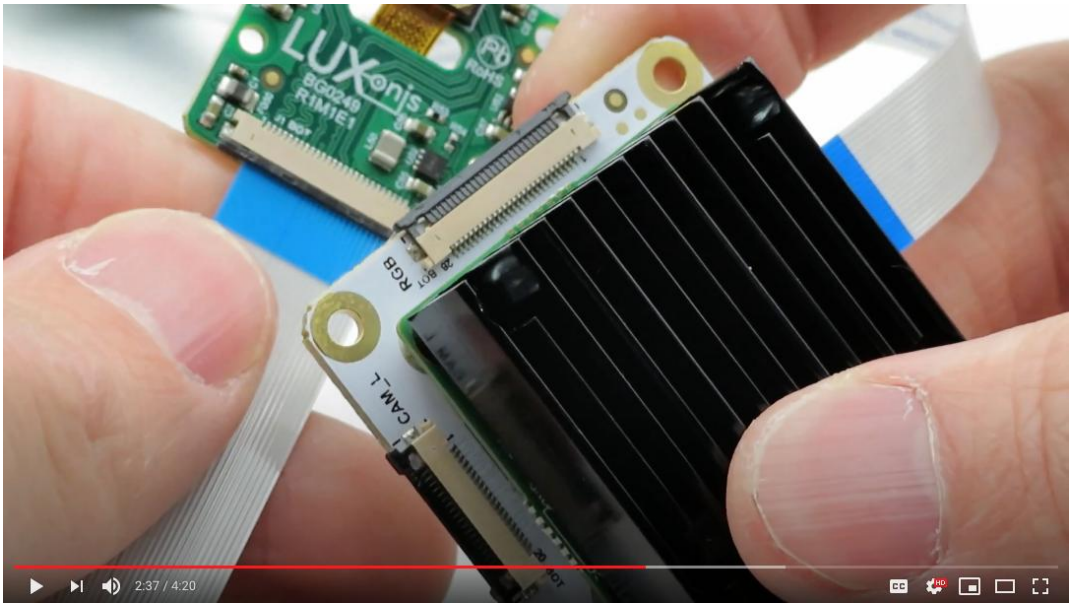
### Setup

Follow the steps below to setup your DepthAI device.

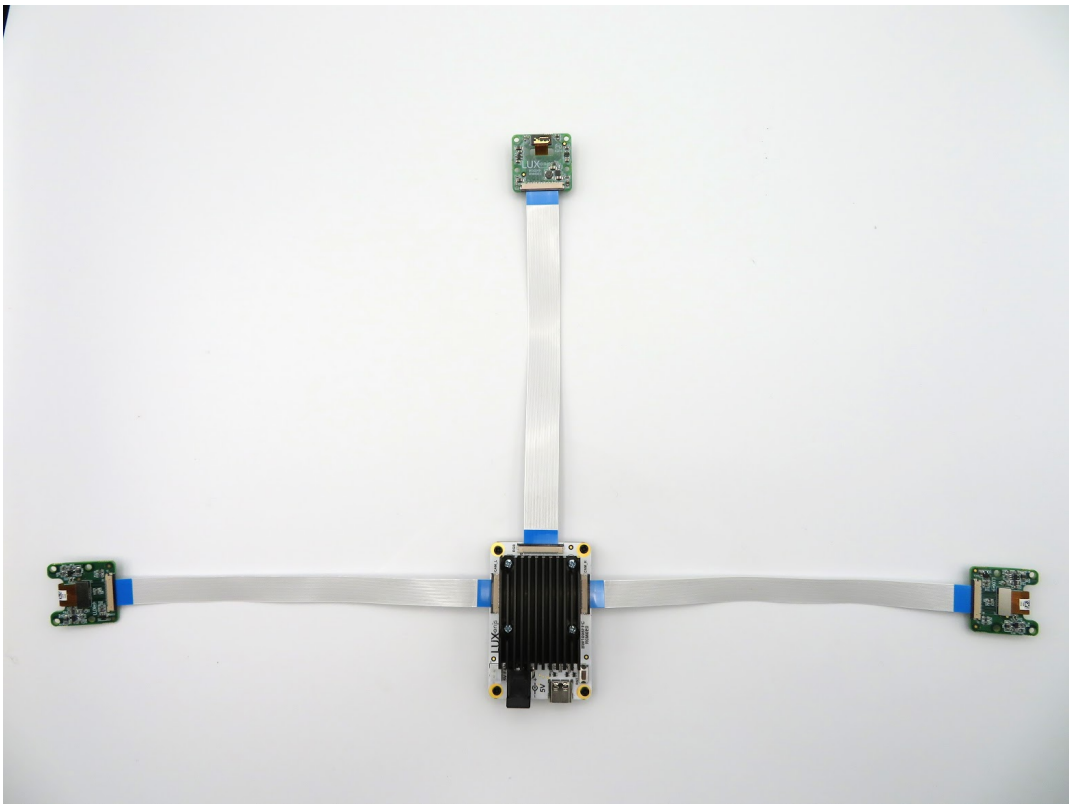
1. **Connect your modular cameras.**

   The FFC (flexible flat cable) Connectors on the BW1098FFC require care when handling. Once inserted and latched, the connectors are robust, but they are easily susceptible to damage during the de-latching process when handling the connectors, particularly if to much force is applied during this process.

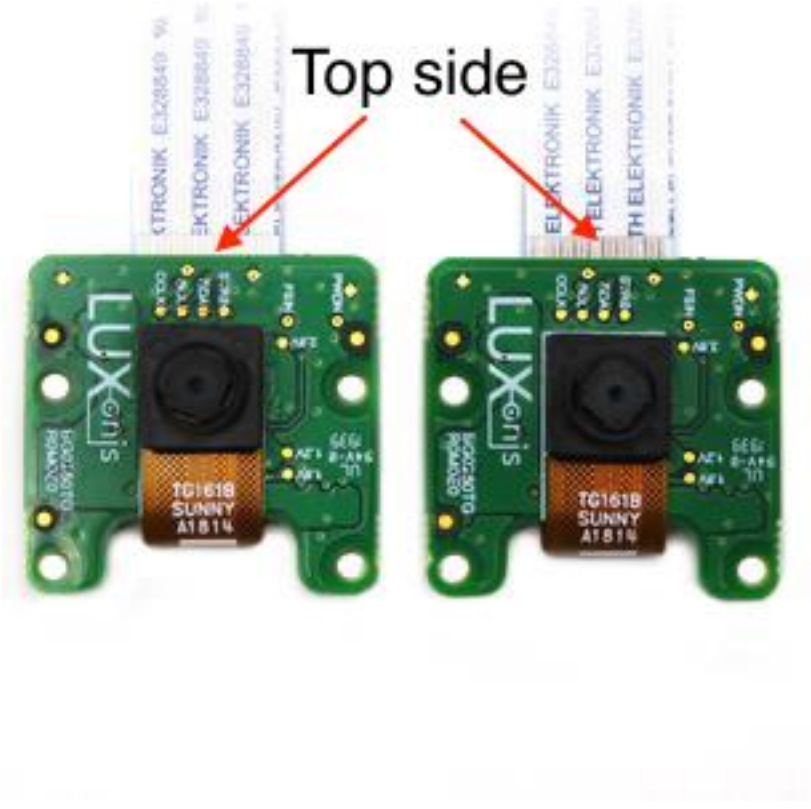
   The video below shows a technique without any tool use to safely latch and delatch these connectors.




   Once the flexible flat cables are securely latched, you should see something like this:



**Note:** Note when looking at the connectors, the blue stripe should be facing up.



> **Warning:** Make sure that the FFC cables connect to the camera is on the top side of the final setup to avoid inverted images and wrong `swap_left_and_right_cameras` setup.

2. **Connect your host to the DepthAI USB carrier board.**
3. **Connect the DepthAI USB power supply (included).**
4. **Calibrate the cameras**.

   See *Calibration*

### Verify installation

We'll execute a DepthAI example Python script to ensure your setup is configured correctly. Follow these steps to test DepthAI:

1. Start a terminal session.
2. Clone the depthai example repository.

```
git clone https://github.com/luxonis/depthai.git
```

1. Access your local copy of depthai.

```
cd [depthai repo]
```

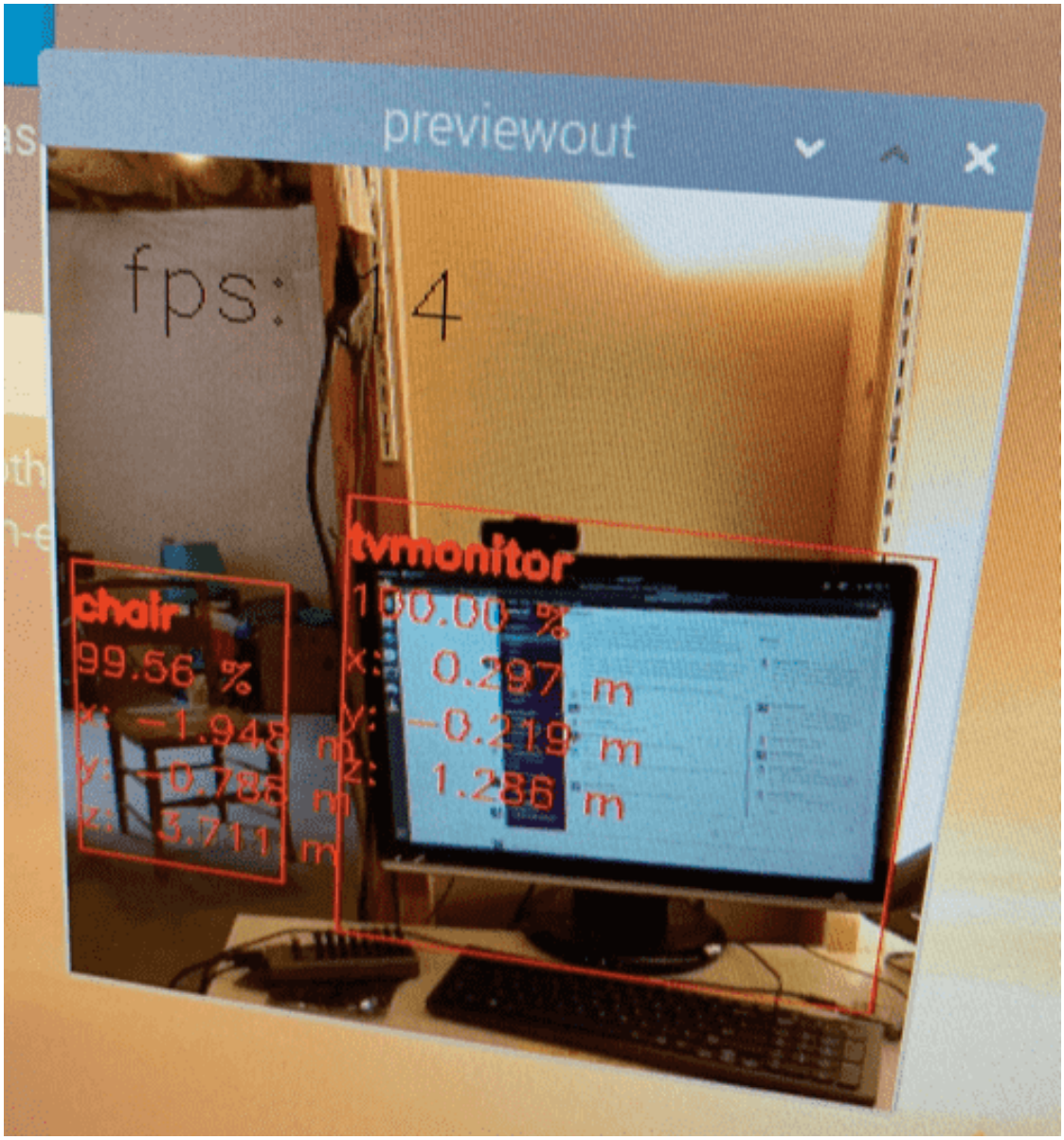
1. Install the example repository requirements.

```
python -m pip install -r requirements.txt
```

1. Run demo script.

```
python3 depthai_demo.py
```

The script launches a window, starts the cameras, and displays a video stream annotated with object localization metadata:




In the screenshot above, DepthAI identified a TV monitor (1.286 m from the camera) and a chair (3.711 m from the camera).

See the list of object labels in our pre-trained OpenVINO model tutorial.

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

### 2.4.6 BW1098OBC - OAK-D | DepthAI Onboard Cameras

The Spatial AI + CV Power House.

#### Requirements

- USB3C cable (included)
- USB-capable host

#### What's in the box?

- OAK-D / DepthAI USB3C
- USB3C cable (3 ft.)
- Power Supply
- Getting Started Card

#### Setup

Install the Python API.

#### Verify installation

We'll execute a DepthAI example Python script to ensure your setup is configured correctly. Follow these steps to test DepthAI:

1. Start a terminal session.
2. Clone the depthai example repository.

```
git clone https://github.com/luxonis/depthai.git
```

1. Access your local copy of depthai.

```
cd [depthai repo]
```

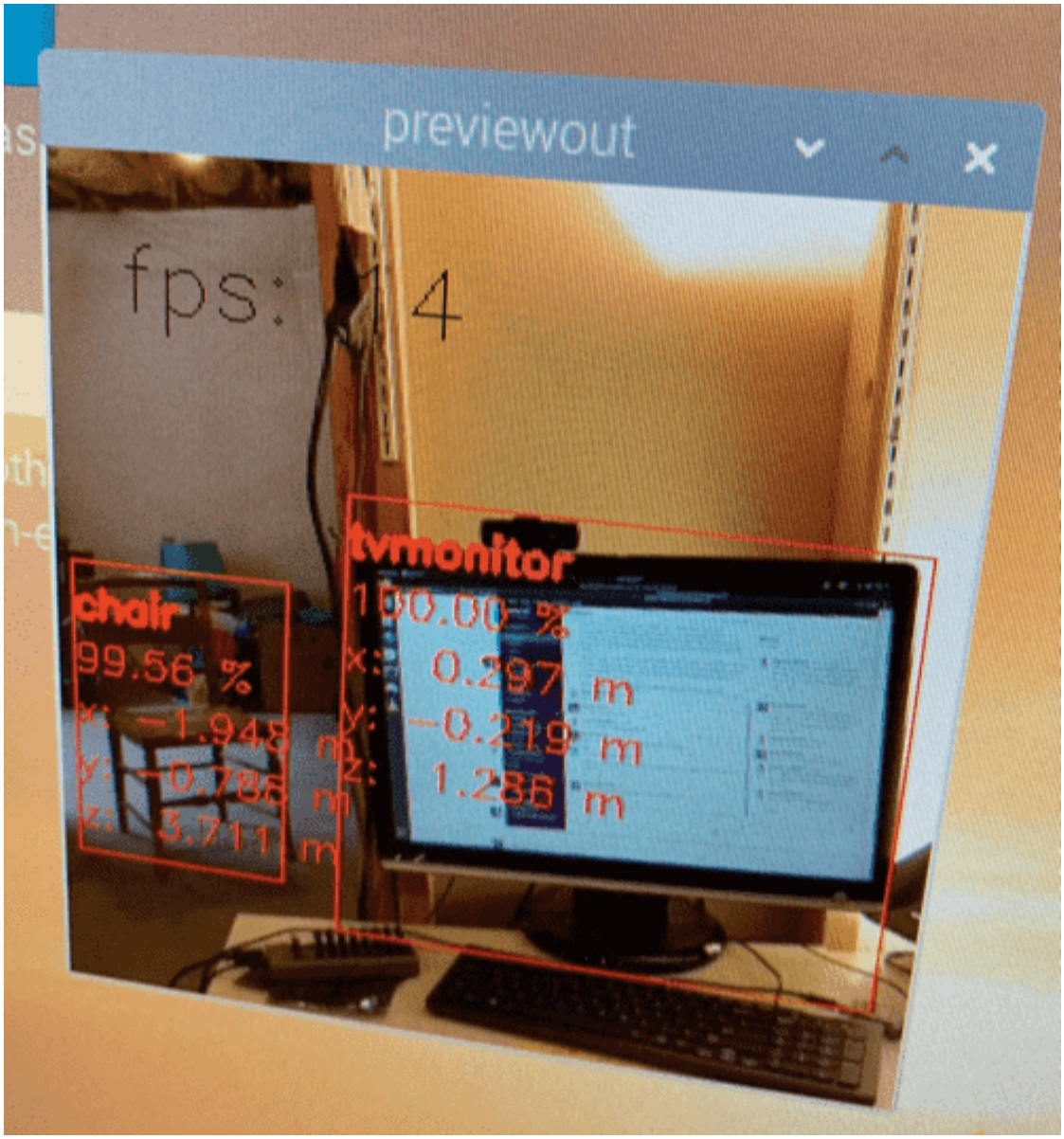
1. Install the example repository requirements.

```
python -m pip install -r requirements.txt
```

1. Run demo script.

```
python3 depthai_demo.py
```

The script launches a window, starts the cameras, and displays a video stream annotated with object localization metadata:




In the screenshot above, DepthAI identified a TV monitor (1.286 m from the camera) and a chair (3.711 m from the camera).

See the list of object labels in our pre-trained OpenVINO model tutorial.

### Camera Specs

Table 11: **Camera Specs**

| | **Color Camera** | **Stereo Camera Pair** |
|---|---|---|
| **Shutter Type** | Rolling Shutter | Sync Global Shutter |
| **Image Sensor** | IMX378 | OV9282 |
| **Max Framerate** | 60fps | 120fps |
| **H.265 Framerate** | 30fps | 120fps |
| **Resolution** | 12MP (4056x3040 px/1.55um) | 1MP (1280x800 px/3um) |
| **Field of View** | 81° DFoV - 68.8° HFoV | 81° DFoV - 71.8° HFoV |
| **Lens Size** | 1/2.3 Inch | 1/2.3 Inch |
| **Focus** | 8cm - $\infty$ (AutoFocus) | 19.6cm - $\infty$ (FixedFocus) |
| **F-number** | 2.0 | 2.2 |

### Compliance

Table 12: Compliance download & reference table

| | |
|---|---|
| megaAI / DepthAI BW1093 / OAK | `Declaration of conformity` |

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

## 2.4.7 BW1099 - System on Module

All DepthAI editions utilize the System on Module (SoM), which can also be used by itself to integrate into your own designs. The SoM allows the board that carries it to be a simple, easy four-layer standard-density board, as opposed

to the high-density-integration (HDI) stackup (with laser-vias and stacked vias) required to directly integrate the VPU itself.

### Specifications

- 2x 2-lane MIPI Camera Interface
- 1x 4-lane MIPI Camera Interface
- Quad SPI with 2 dedicated chip-selects
- $I^2C$
- UART
- USB2
- USB3
- Several GPIO (1.8 V and 3.3 V)
- Supports off-board eMMC or SD Card
- On-board NOR boot Flash (optional)
- On-board EEPROM (optional)
- All power regulation, clock generation, etc. on module
- All connectivity through single 100-pin connector (DF40C-100DP-0.4V(51))

**Datasheets** are available here and for EMB edition here

### Getting Started Integrating Into Your Products

All the boards based on the DepthAI System on Module are available on Github under MIT License here.

These are in Altium Designer format. So if you use Altium Designer, you're in luck! You can quickly/easily integrate the DepthAI SoM into your products with proven and up-to-date designs (the same designs you can buy here).

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

### 2.4.8 DepthAI Color Camera


4K, 60Hz video camera with 12 MP stills and 4056 x 3040 pixel resolution.

#### Specifications

- 4K, 60 Hz Video
- 12 MP Stills
- Same dimensions, mounting holes, and camera center as Raspberry Pi Camera v2.1
- 4056 x 3040 pixels
- 81 DFOV°
- Lens Size: 1/2.3 inch
- AutoFocus: 8 cm - $\infty$
- F-number: 2.0

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

### 2.4.9 DepthAI Mono Camera


For applications where Depth + AI are needed, we have modular, high-frame-rate, excellent-depth-quality cameras which can be separated to a baseline of up to 30 cm).

#### Specifications

- 720p, 120 Hz Video
- Synchronized Global Shutter
- Excellent Low-light
- Same dimensions, mounting holes, and camera center as Raspberry Pi Camera v2.1
- 1280 x 720 pixels
- 83 DFOV°
- Lens Size: 1/2.3 inch
- Fixed Focus: 19.6 cm - $\infty$
- F-number: 2.2

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

## 2.5 Calibration

**Note:** The *BW1097 - RaspberryPi Compute Module*, *BW1098OBC* and OAK-D are calibrated before shipment.

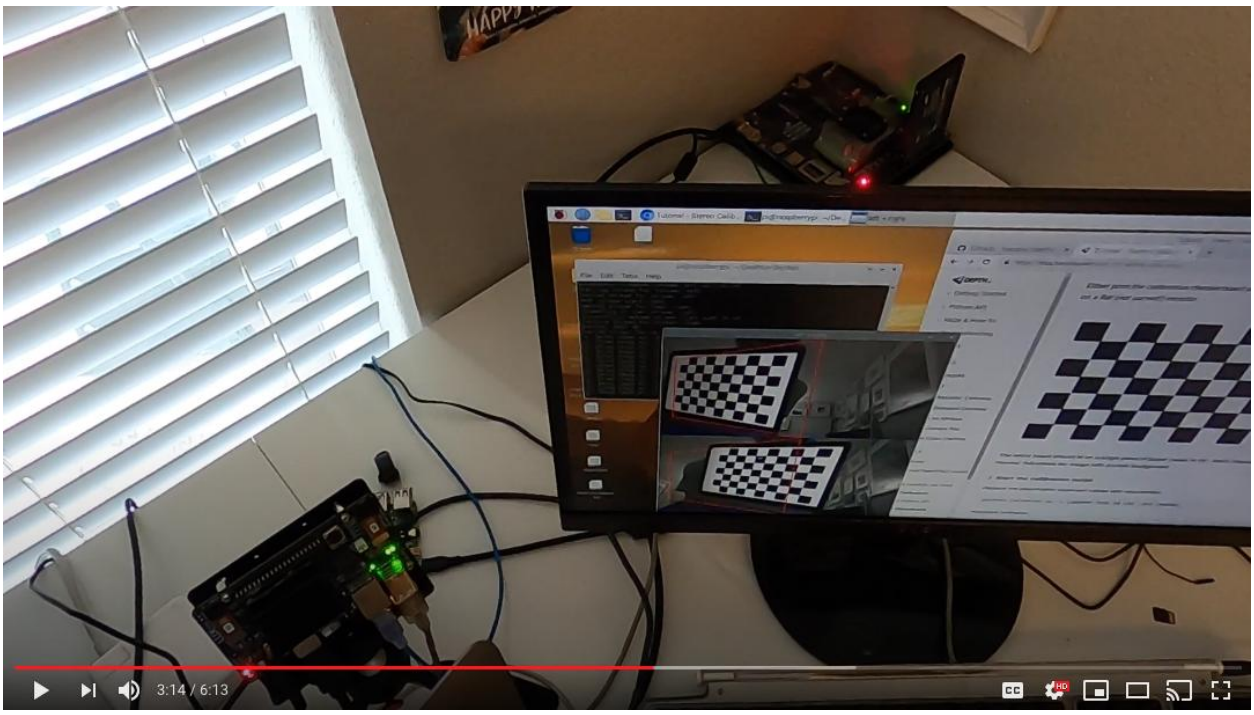
For the modular camera editions of DepthAI (*BW1098FFC - USB3 with Modular Cameras* and *BW1094 - RaspberryPi Hat*) it is necesssary to do a stereo camera calibration after mounting the cameras in the baseline/configuration for your application.

For the *BW1097 - RaspberryPi Compute Module* and *BW1098OBC*, the units come pre-calibrated - but you may want to re-calibrate for better quality in your installation (e.g. after mounting the board to something), or if the calibration quality has started to fade over use/handling.

Below is a quick video showing the (re-) calibration of the *BW1097 - RaspberryPi Compute Module*. In short, the calibration uses the intersections to determine the orientation and distance of the checkerboard. So the greatest accuracy will be obtained by a clear print or display of the provided checkerboard image on a flat plane.

The flatness of the calibration checkerboard is very important. Do not use curved monitors, or calibration targets that have any 'waviness'. So if you print the checkerboard, please make sure to affix the sheet to a known flat surface, without any waves. That said, using a laptop with a flat monitor is usually the easiest technique.

Watching the video below will give you the steps needed to calibrate your own DepthAI. For more information/details on calibration options, please see the steps below and also `./calibrate.py --help` which will print out all of the calibration options.




1. **Checkout the depthai GitHub repo.**

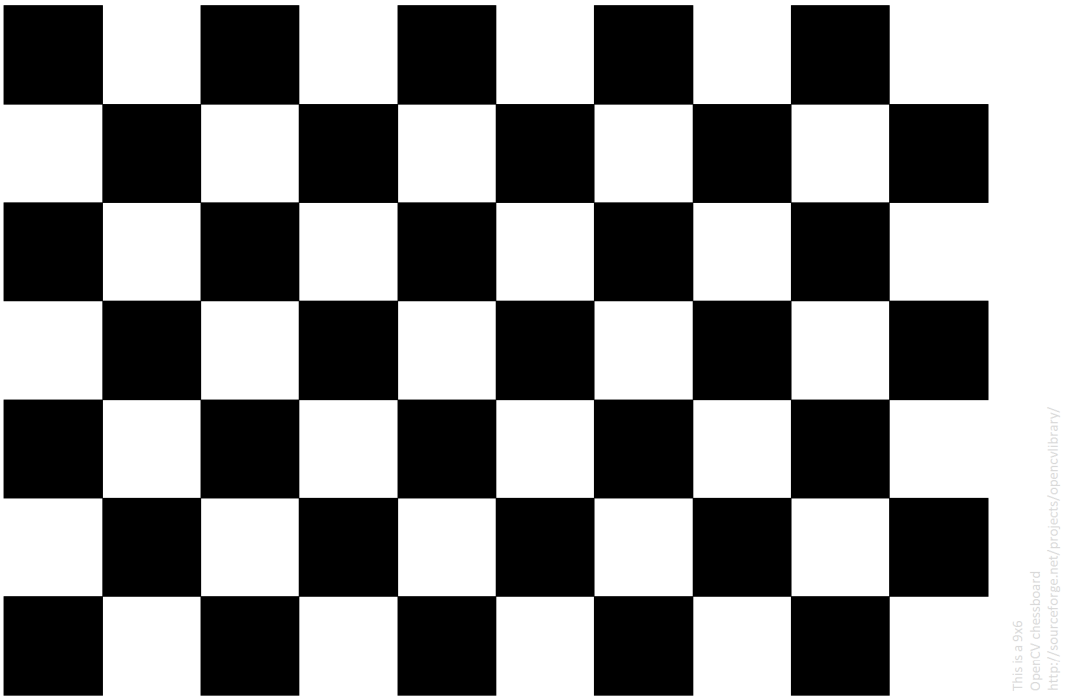
   **Warning:** Already checked out depthai? **Skip this step.**

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

2. Print checkerboard calibration image.

   Either print the calibration checkerboard onto a flat surface, or display the checkerboard on a flat (not curved!) monitor. Note that if you do print the calibration target, take care to make sure it is attached to a flat surface and is flat and free of wrinkles and/or 'waves'.

   Often, using a monitor to display the calibration target is easier/faster.




   The entire board should fit on a single piece of paper (scale to fit). And if displaying on a monitor, full-screen the image with a white background.

3. Start the calibration script.

   Replace the placeholder argument values with valid entries:

```
python3 calibrate.py -s [SQUARE_SIZE_IN_CM] -brd [BOARD]
```

   Argument reference:

   - `-s SQUARE_SIZE_IN_CM`, `--square_size_cm SQUARE_SIZE_IN_CM`: Measure the square size of the printed checkerboard in centimeters.
   - `-brd BOARD`, `--board BOARD`: BW1097, BW1098OBC - Board type from resources/boards/ (not case-sensitive). Or path to a custom .json board config. Mutually exclusive with [-fv -b -w], which allow manual specification of field of view, baseline, and camera orientation (swapped or not-swapped).

   Retrieve the size of the squares from the calibration target by measuring them with a ruler or calipers and enter that number (in cm) in place of [SQUARE_SIZE_IN_CM].

   For example, the arguments for the *BW1098OBC* look like the following if the square size is 2.35 cm:

```
python3 calibrate.py -s 2.35 -brd bw1098obc
```

And note that mirroring the display when calibrating is often useful (so that the directions of motion don't seem backwards). When seeing ourselves, we're used to seeing ourselves backwards (because that's what we see in a mirror), so do so, use the `-ih` option as below:

```
python3 calibrate.py -s 2.35 -brd bw1098obc -ih
```

So when we're running calibration internally we almost always use the `-ih` option, so we'll include it on all the following example commands:

- **BW1098OBC (USB3 Onboard Camera Edition)):**

```
python3 calibrate.py -s [SQUARE_SIZE_IN_CM] -brd bw1098obc -ih
```

- **BW1097 (Raspberry Pi Compute Module Edition):**

```
python3 calibrate.py -s [SQUARE_SIZE_IN_CM] -brd bw1097 -ih
```

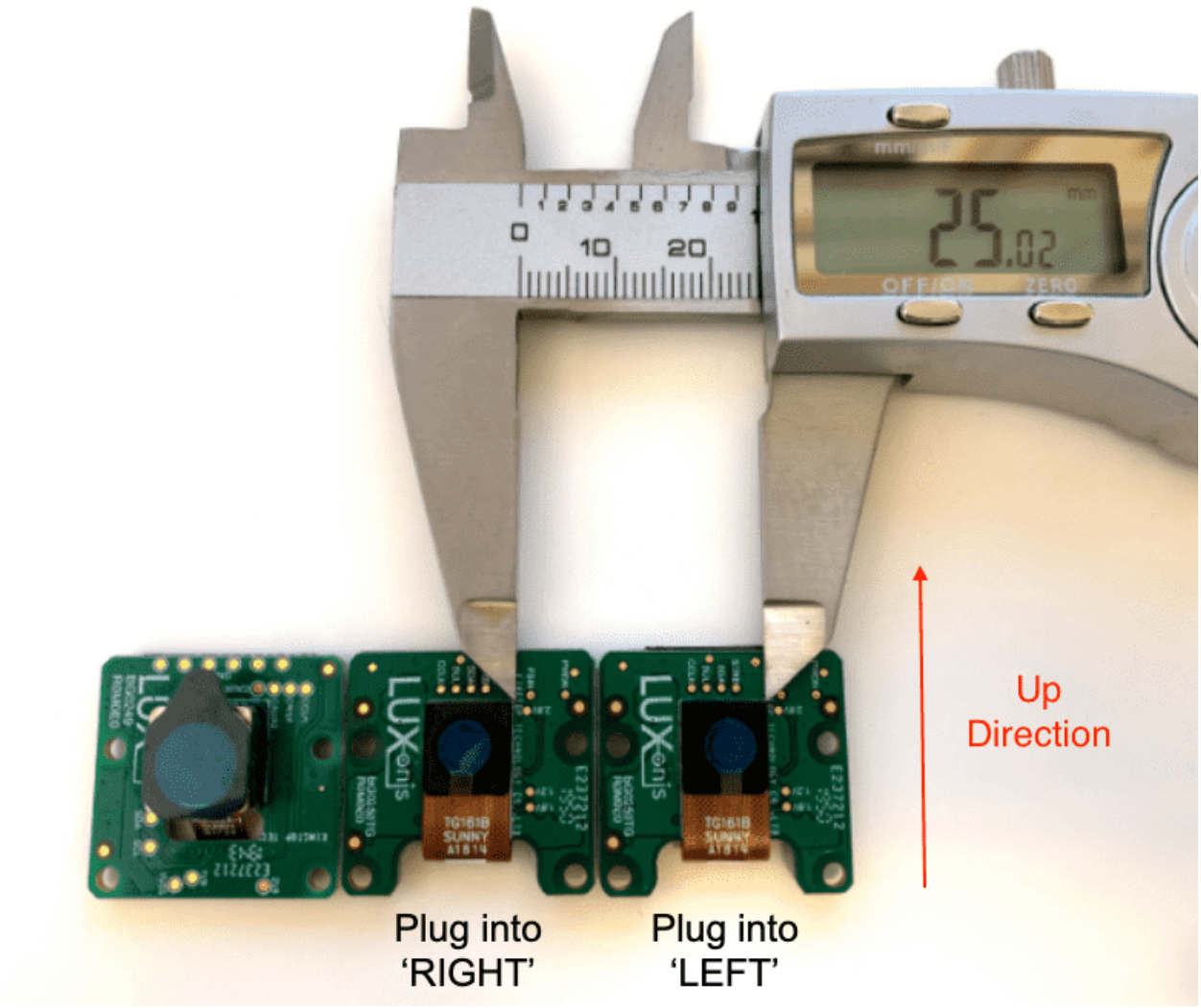
### 2.5.1 Modular cameras calibration

Use one of the board `*.json` files from here to define the baseline between the stereo cameras, and between the left camera and the color camera, replacing the items in brackets below.

- Swap left/right (i.e. which way are the cameras facing, set to `true` or `false`)
- The `BASELINE` in centimeters between grayscale left/right cameras
- The distance `RGBLEFT` separation between the `Left` grayscale camera and the color camera, in centimeters.

```
{
    "board_config":
    {
        "name": "ACME01",
        "revision": "V1.2",
        "swap_left_and_right_cameras": [true | false],
        "left_fov_deg": 73.5,
        "rgb_fov_deg": 68.7938,
        "left_to_right_distance_cm": [BASELINE],
        "left_to_rgb_distance_cm": [RGBLEFT]
    }
}
```

So for example if you setup your BW1098FFC with a stereo baseline of 2.5cm, with the color camera exactly between the two grayscale cameras, as shown below, use the JSON further below:




```
{
    "board_config":
    {
        "name": "ACME01",
        "revision": "V1.2",
        "swap_left_and_right_cameras": true,
        "left_fov_deg": 73.5,
        "rgb_fov_deg": 68.7938,
        "left_to_right_distance_cm": 2.5,
        "left_to_rgb_distance_cm": 5.0
    }
}
```

Note that in this orientation of of the cameras, `"swap_left_and_right_cameras"` is set to true.

Then, run calibration with this board name:

```
python3 calibrate.py -s [SQUARE_SIZE_IN_CM] -brd ACME01 -ih
```

Run `python3 calibrate.py --help` (or `-h`) for a full list of arguments and usage examples.

### 2.5.2 Position the checkerboard and capture images

Left and right video streams are displayed, each containing a polygon overlay.

Hold up the printed checkerboard (or laptop with the image displayed on the screen) so that the whole of the checkerboard is displayed within both video streams.

Match the orientation of the overlayed polygon and press [SPACEBAR] to capture an image. The checkerboard pattern does not need to match the polygon exactly, but it is important to use the polygon as a guideline for angling and location relative to the camera. There are 13 required polygon positions.

After capturing images for all of the polygon positions, the calibration image processing step will begin. If successful, a calibration file will be created at `depthai/resources/depthai.calib`. This file is loaded by default via the `calib_fpath` variable within `consts/resource_paths.py`.

### 2.5.3 Test depth

We'll view the depth stream to ensure the cameras are calibrated correctly:

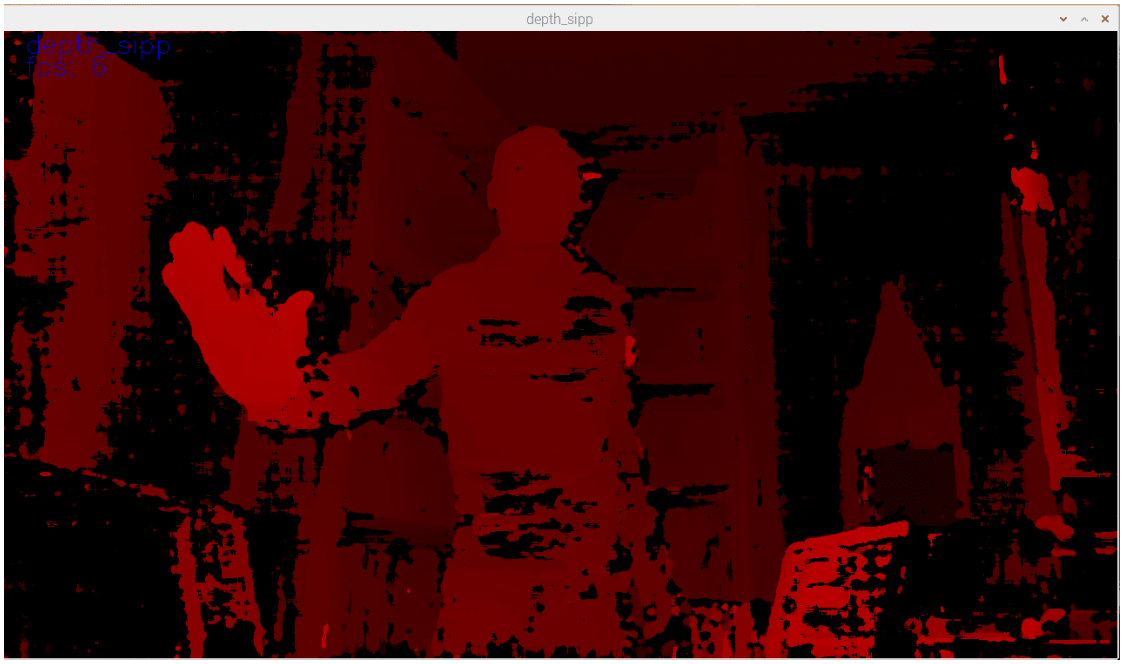
1. Start a terminal session.
2. Access your local copy of depthai.

```
cd [depthai repo]
```

3. Run test script.

```
python3 depthai_demo.py -s depth_raw -o
```

The script launches a window, starts the cameras, and displays a depth video stream:




In the screenshot above, the hand is closer to the camera.

### 2.5.4 Write calibration and board parameters to on-board eeprom

If your are happy with the depth quality above, you can write it to the on-board eeprom on DephtAI so that the calibration stick with DepthAI (all designs which have stereo-depth support have on-board eeprom for this purpose).

To write the calibration and associated board information to to EEPROM on DepthAI, use the following command:

```
python3 depthai_demo.py -brd [BOARD] -e
```

Where `[BOARD]` is either `BW1097` (Raspberry Pi Compute Module Edition), `BW1098OBC` (USB3 Onboard Camera Edition) or a custom board file (as in *here*), all case-insensitive.

So for example to write the (updated) calibration and board information to your BW1098OBC, use the following command:

```
python3 depthai_demo.py -brd bw1098obc -e
```

And to verify what is written to EEPROM on your DepthAI, you can see check the output whenever running DetphAI, simply with"

```
python3 depthai_demo.py
```

And look for `EEPROM data:` in the prints in the terminal after running the above command:

```
EEPROM data: valid (v2)
  Board name     : BW1098OBC
  Board rev      : R0M0E0
  HFOV L/R       : 73.5 deg
  HFOV RGB       : 68.7938 deg
  L-R   distance : 7.5 cm
  L-RGB distance : 3.75 cm
  L/R swapped    : yes
  L/R crop region: top
  Calibration homography:
    1.002324,   -0.004016,   -0.552212,
    0.001249,    0.993829,   -1.710247,
    0.000008,   -0.000010,    1.000000,
```

If anything looks incorrect, you can calibrate again and/or change board information and overwrite the stored eeprom information and calibration data using the `-brd` and `-e` flags as above.

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

## 2.6 Custom training

Here we have examples of Google Colab notebooks trained on various data sets. They are free GPU instances, so great for prototyping and even simple production models.

The below tutorials cover MobileNetv2-SSD, tiny-YOLOv3, tiny-YOLOv4, and Deeplabv3+ (semantic segmentation). A bunch of other object detectors and neural networks could be trained/supported on Colab and run on DepthAI, so if you have a request for a different object detector/network backend, please feel free to make a Github Issue!

And please feel free to work directly from our Github of `depthai-ml-training` for the latest models we support:

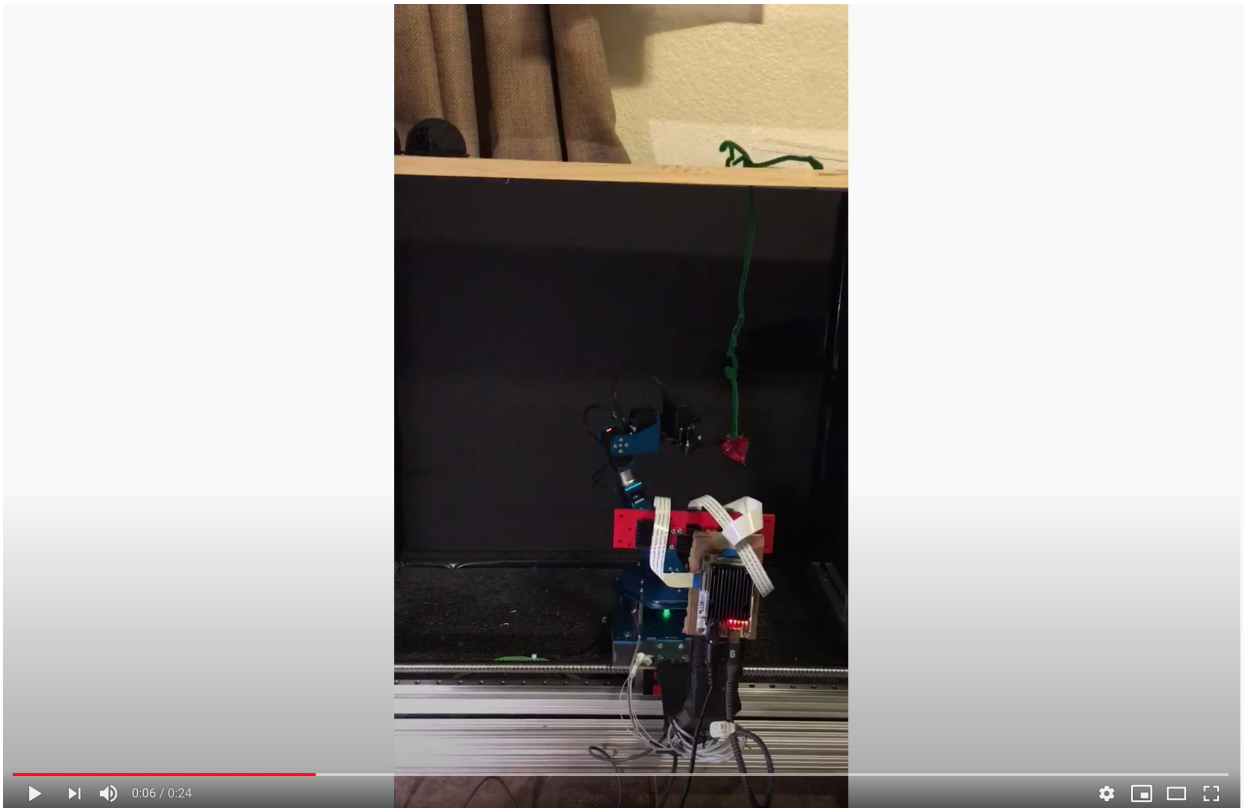
- depthai-ml-training

The tutorial notebook *Easy_Object_Detection_With_Custom_Data_Demo_Training.ipynb* shows how to quickly train an object detector based on the MobileNet SSDv2 network.

Optionally, see our documentation around this module (here) for of a guide/walk-through on how to use this notebook. Also, feel free to jump right into the Notebook, with some experimentation it's relatively straightforward to get a model trained.

After training is complete, it also converts the model to a .blob file that runs on our DepthAI platform and modules. First the model is converted to a format usable by OpenVINO called Intermediate Representation, or IR. The IR model is then compiled to a .blob file using a server we set up for that purpose. (The IR model can also be converted locally to a blob.)

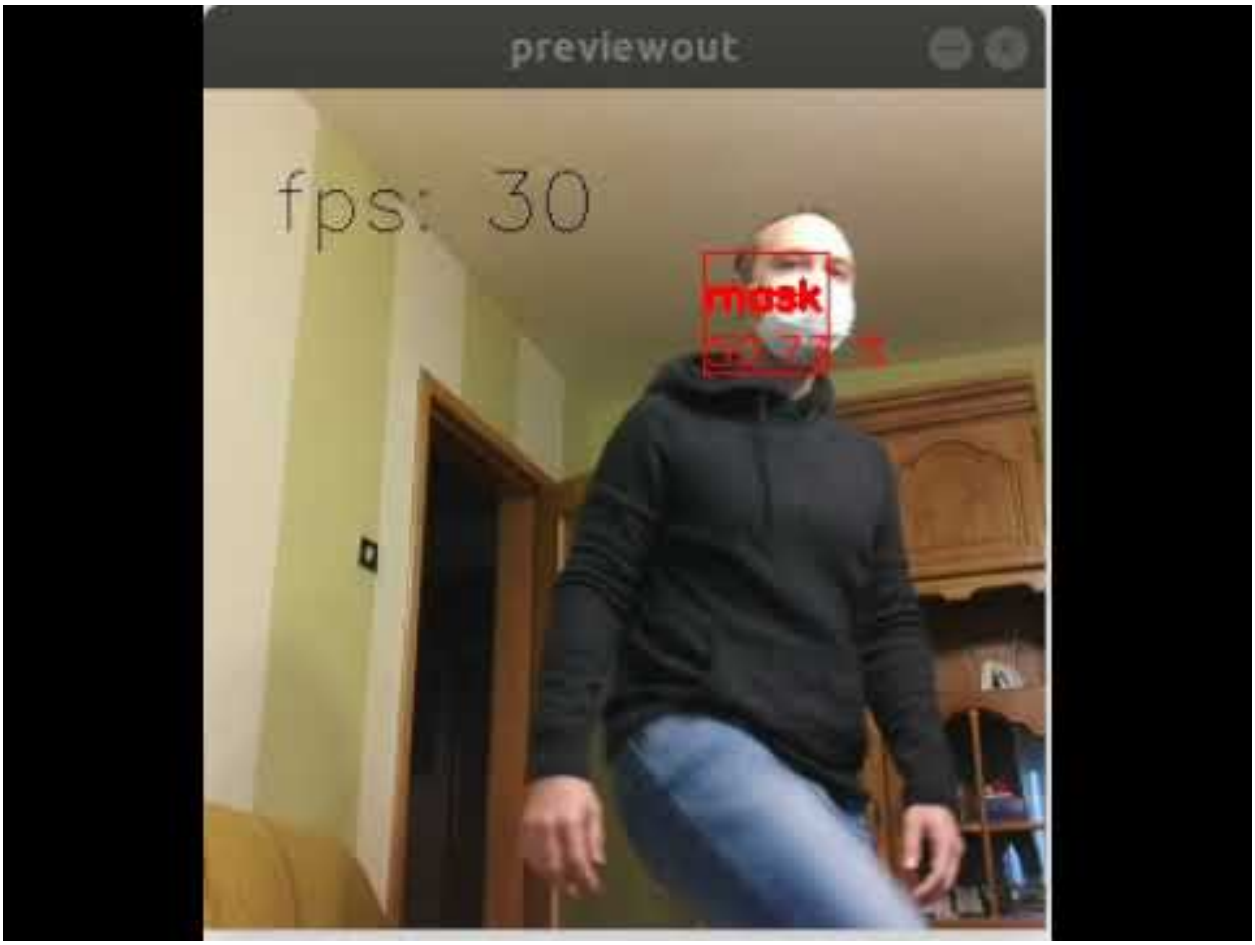
And that's it, in less than a couple of hours a fairly advanced proof of concept object detector can run on DepthAI to detect objects of your choice and their associated spatial information (i.e. X, Y, Z coordinates). For example this notebook was used to train DepthAI to locate strawberries in 3D space, see below:




The above example used a DepthAI Modular Cameras Edition (BW1098FFC).

The *Medical Mask Detection Demo Training.ipynb* training notebook shows another example of a more complex object detector. The training data set consists of people wearing or not wearing masks for viral protection. There are almost 700 pictures with approximately 3600 bounding box annotations. The images are complex: they vary quite a lot in scale and composition. Nonetheless, the object detector does quite a good job with this relatively small data set for such a task. Again, training takes around 2 hours. Depending on which GPU the Colab lottery assigns to the notebook instance, training 10k steps can take 2.5 hours or 1.5 hours. Either way, a short period for such a good quality proof of concept for such a difficult task. We then performed the steps above for converting to blob and then running it on our DepthAI module.

Below is a quick test of the model produced with this notebook on Luxonis DepthAI Onboard Cameras Edition (BW1098OBC):




This notebook operates on your set of images in Google Drive to resize them to the format needed by the training notebooks.

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

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