Depth Mapping on Arducam Stereo Camera HAT with OpenCV

Table Of Contents

Feedback

Introduction

The Arducam Stereo Camera HAT is a revolutionary product that enables you to use two synchronized Pi camera modules on a single standard Raspberry Pi model. The stereo camera setup could be used in lots of applications that are hard to realize in the past. This documentation will show a demo of using the Arducam stereo camera HAT for depth mapping on a

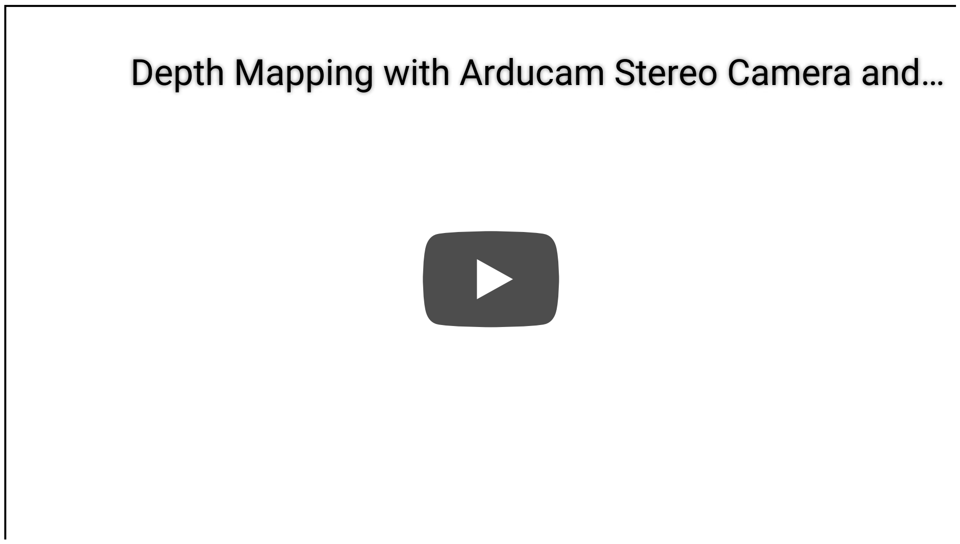
Raspberry Pi 3 with the help of our OV5647 Stereo Camera Board.

Credits to StereoPi, on which this tutorial and the codes used are based. Thanks to them for the work they have done.

> Note
>
> All of the following examples are for OpenCV novices, which are intended to be used in real production.

Video Tutorial

We also provide a video demo of this tutorial, and you can check that below:



Hardware setup

- Arducam Stereo Camera HAT, OV5647 Stereo Camera Board, and Raspberry Pi 3B are used.

Software Prerequisite

- Raspbian Buster (kernel 4.19.57-v7+), latest version recommended
- Python 3.5.3
- OpenCV 3.4.4 (pre-compiled, 'pip' from Python Wheels)
- StereoVision lib 1.0.3 (`https://github.com/erget/StereoVision`)

Installing Dependencies and Libraries

- Installing the latest Raspbian

- `sudo apt-get update && sudo apt-get install -y libhdf5-dev libhdf5-serial-dev libatlas-base-dev libjasper-dev libqtgui4 libqt4-test && sudo pip3 install opencv-python==3.4.6.27`
- `sudo pip3 install stereovision`
- `sudo pip3 install matplotlib`
- `git clone https://github.com/ArduCAM/MIPI_Camera.git`

Notice

Key stroke processing is supported for all. Press `Q` key to stop them. If `Ctrl+C` is used to stop the script, the program results could be incorrect.

Step 1:Capture Images

The `1_test.py` script will be used in this step.

Open the console and go to the examples folder:

```
cd MIPI_Camera/RPI/stereo_depth_demo
```

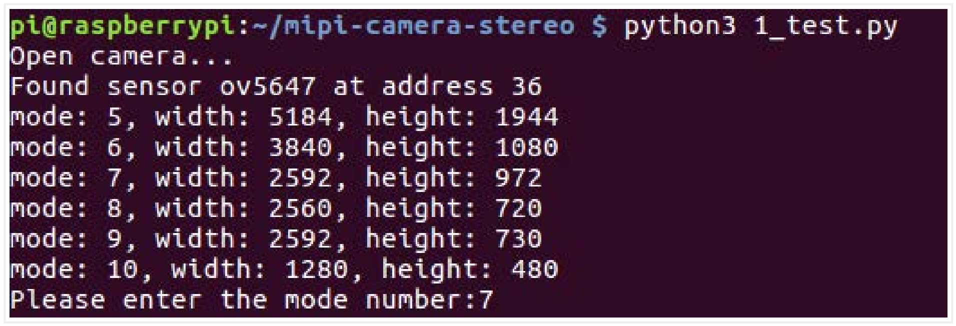
Console Command:

```
python3 1_test.py
```

Select the camera mode after running this program, and then you will see a preview window. Press `Q` to stop the process and save the last image captured, which will be used in the next scripts for tuning the parameters.

It helps you make sure the hardware is working and captures the first picture. Camera modes and resolutions(width and height) will be recorded for later uses.

Feedback



```
pi@raspberrypi:~/mipi-camera-stereo $ python3 1_test.py
Open camera...
Found sensor ov5647 at address 36
mode: 5, width: 5184, height: 1944
mode: 6, width: 3840, height: 1080
mode: 7, width: 2592, height: 972
mode: 8, width: 2560, height: 720
mode: 9, width: 2592, height: 730
mode: 10, width: 1280, height: 480
Please enter the mode number:7
```

- How the first script works

You can also see from the video above about how the first script works:

Note

By default, the script will scale the width to 1280 while keeping the aspect ratio, regardless of the resolutions chosen.

Step 2: Collect Images for Calibration

A perfect depth mapping sets a high standard for the camera setup, where two identical cameras have their optical, vertical and horizontal axis all in parallel. However, you can reach that perfection in real life, so calibration is required. The calibration software samples pictures of these two cameras on a specific object, analyzes them, and generates parameters for the correction. A chessboard is used in this demo.

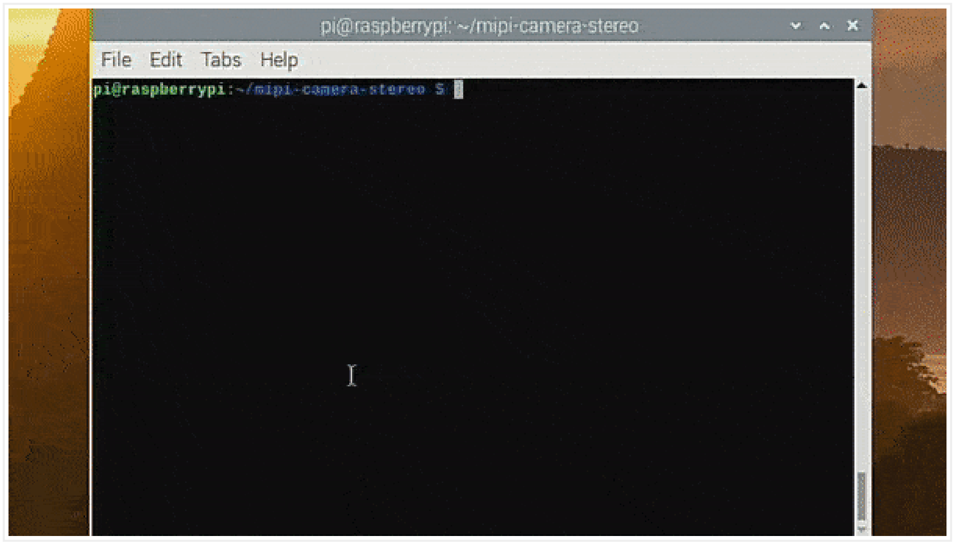
The program starts with a 5-second countdown for positioning the object and then takes the picture. This process will be looped for 30 times by default, and repositioning is required between each countdown.

Note

Try to hold the object steady and keep it within both camera's sight.

Console command:

```
python3 2_chess_cycle.py
```



```
pi@raspberrypi:~/mipi-camera-stereo $
```

- Collect Images for Calibration

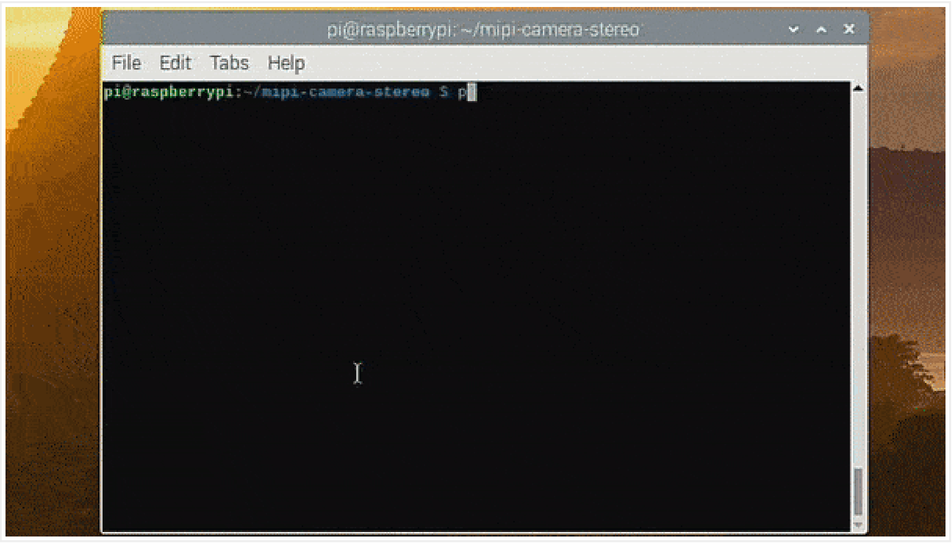
You can see from the video above to see more about this.

30 stereoscopic photos will then be saved in `/scenes` folder.

Step 3: Separate Captured Images

The `3_pairs_cut.py` will separate the photos captured in the last step into left and right halves and save them in the `/pairs` folder. This separating process can run on-the-fly without saving, but this step could be helpful otherwise for subsequent uses. Image pairs could be saved from different capture series. You can process these images with your own codes or putting other stereoscopic images in this folder. You can also go through every stereo pair before the separation and sort the bad frames out. Console Command:

```
python3 3_pairs_cut.py
```



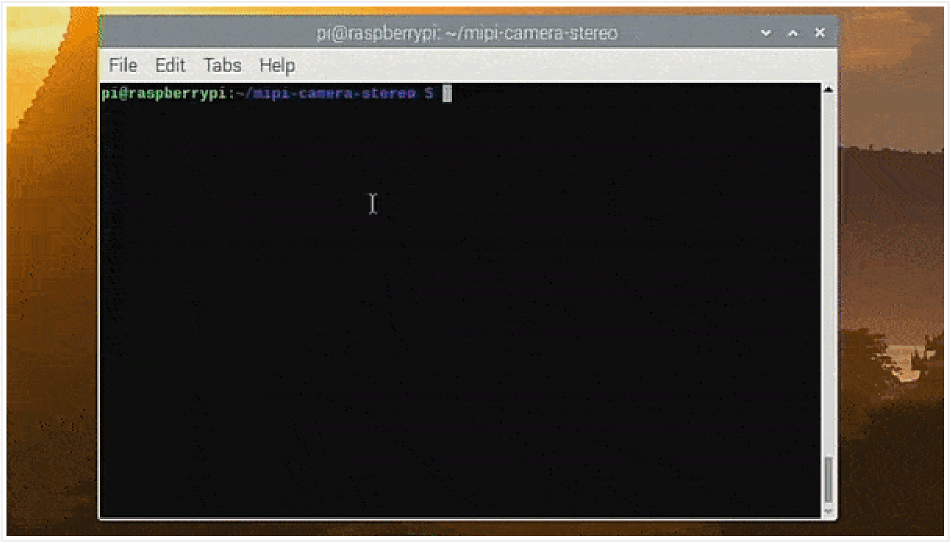
```
pi@raspberrypi:~/mipi-camera-stereo $ p
```

- Separate Captured Images

You can see from the video above to see more about this.

Step 4: Calibration

`4_calibration.py` loads all previously saved pairs and figures out the correction matrices. It searches for the chessboard on the photo, and move on if the chessboard is not found. Therefore, the bad photos in the series won't break the script. After the calculation is finished, it will rectify the last image and show the resulting "fixed" images. You can check the quality of the calibration in this step, which might take 1-2 minutes.



```
pi@raspberrypi:~/mipi-camera-stereo $
```

- calibration

You can see from the video above to see more about this.

Console Command:

```
python3 4_calibration.py
```

Calibration script doing his job:

> **Note**
>
> 1.If the calibration result is not correct, please delete the pictures under `/scenes` and `/pairs`, and capture the pictures again.
> 2. Please modify the **chessboard parameters** in this script if other chess boards are used.

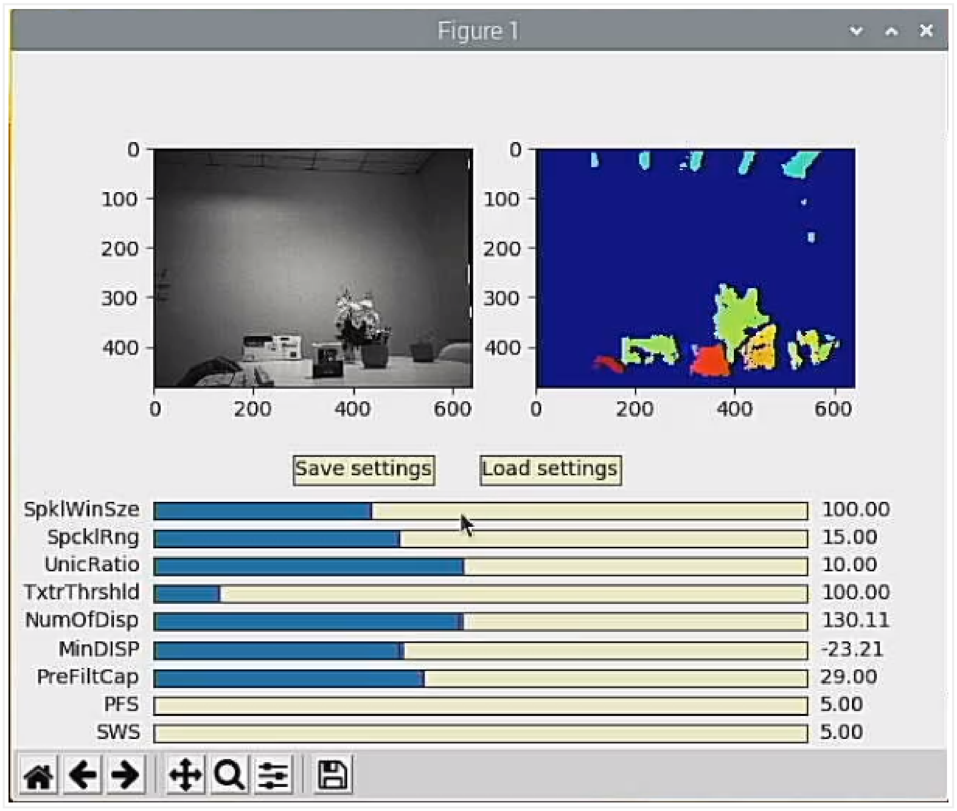
Step 5: Depth Map Tuning

The image in script 1 and the calibration results from step 4 will be loaded in `5_dm_tune.py`. It shows a depth map with an interface for fine-tuning. Press the `Load Settings` button to use the default parameters. It's recommended to take a photo with 3 objects from different distances to find the right settings before you tune the parameters. The closest will be red, and the furthest far away.

You can see from the video above to see more about this.

```
python3 5_dm_tune.py
```

Here is how this looks like:



```
SpklWinSze   100.00
SpcklRng     15.00
UnicRatio    10.00
TxtrThrshld  100.00
NumOfDisp    130.11
MinDISP      -23.21
PreFiltCap   29.00
PFS          5.00
SWS          5.00
```

- depth map tuning result

Step 6: Real-Time Depth Map Using Video

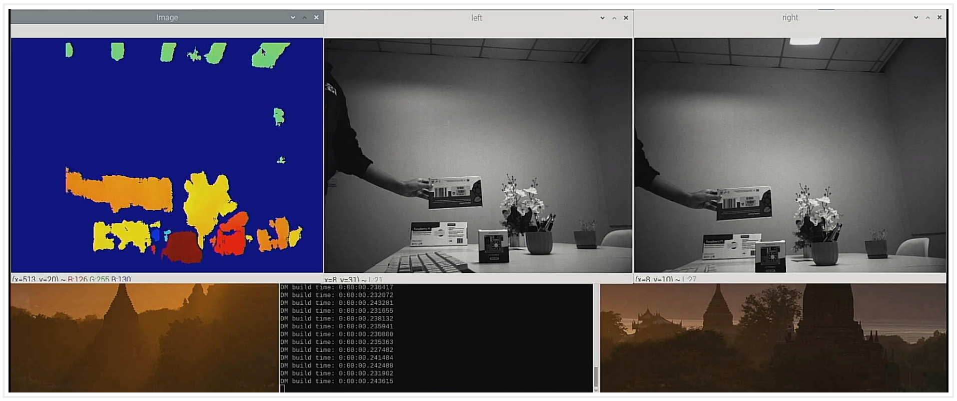
`6_dm_video.py` uses the results from the above steps to generate a depth map in realtime.

You can see from the video above to see more about this.

Console Command:

```
python3 6_dm_video.py
```

The result:



```
DM build time: 0:00:00.236417
DM build time: 0:00:00.232072
DM build time: 0:00:00.243281
DM build time: 0:00:00.231655
DM build time: 0:00:00.238132
DM build time: 0:00:00.235941
DM build time: 0:00:00.230800
DM build time: 0:00:00.235363
DM build time: 0:00:00.227482
DM build time: 0:00:00.241484
DM build time: 0:00:00.242488
DM build time: 0:00:00.231902
DM build time: 0:00:00.243615
```

- Real-Time Depth Map Using Video Result

Updated on August 29, 2021

← Sync Stereo Camera HAT User Manual

Camarray – Arducam 5MP Synchronized Stereo Camera Bundle Kit →

Was this article helpful to you? Yes 12 No 1

Feedback

Information

CONTACT US

PRIVACY POLICY

ABOUT US

SUBSCRIBE TO OUR NEWSLETTER

FOCAL LENGTH GENERATOR

Documentation

Contact Us

Feedback