UNITED INTERNATIONAL UNIVERSITY (UIU)

Dept. of Computer Science & Engineering

Course No: CSE 4326

Course Title: Microprocessors and Microcontrollers Laboratory

Exp No. 04 : Image and Video Processing with Raspberry Pi by interfacing pi camera

Objective:

In this experiment, participants will learn to set up and utilize the Raspberry Pi Camera Module for capturing images and videos, both from the command line and programmatically using Python. Students will gain an understanding of the Camera Module's versatility and flexibility, also, they can experiment with various image and video settings, and explore different camera modes. Another main objective of this experiment is to learn how to solve computer vision problems using Raspberry Pi. We will see an example on object detection using a builtin machine learning model at the end of this experiment. By the end of this experiment, participants will be proficient in using the Camera Module, customizing its settings, and have hands-on experience capturing images and videos. They will also be prepared for more advanced projects involving image and video processing (such as object detection) with Raspberry Pi. Final outcomes of this experiment will be-

- Proficiency in setting up and using the Raspberry Pi Camera Module.
- Ability to capture images and record videos using both command-line tools and Python code.
- Understanding of customizable camera settings and their impact on image and video quality.
- Familiarity with various camera modes for specialized applications.
- Practical experience in capturing images and videos with the Camera Module.
- Preparedness for more advanced projects in image and video processing with Raspberry Pi.

Introduction:

The Raspberry Pi Camera Module is a compact and versatile camera accessory designed specifically for use with Raspberry Pi single-board computers. It provides an affordable and accessible solution for capturing images and videos directly from the Raspberry Pi platform, enabling users to explore a wide range of creative and practical applications.

Equipped with a high-quality image sensor, the camera module is capable of capturing still images with impressive resolution and detail. It also supports video recording, allowing users to create dynamic and engaging visual content. Whether you're a beginner experimenting with basic

photography or an experienced developer working on advanced computer vision algorithms, the Raspberry Pi Camera Module provides a powerful toolset to bring your ideas to life.

One of the standout features of the Raspberry Pi Camera Module is its flexibility. It supports various camera modes, including standard mode for capturing images, video mode for recording videos, and even a specialized mode for low-light photography. Additionally, the camera module offers programmable controls and interfaces, empowering users to customize and fine-tune their camera settings to suit their specific needs and applications.

At present, Raspberry Pi Foundation offers three camera modules:

- Camera Module 2
- Camera Module 2 NoIR
- Raspberry Pi High-Quality Camera

The camera module 2 is a replacement for the original camera module in April 2016. The V2 module has a Sony IMX219 8-Megapixel Sensor compared to the OmniVision OV5647 5-Megapixel sensor in the original module. The module is compatible with Raspberry Pi 1, 2, 3, and 4. It can be easily attached to the Camera Serial Interface (CSI port) of any Raspberry Pi. The V2 camera module supports 1080p30, 720p60 and VGA90 video modes. The new module is not just high in resolution. It is far better in image quality, low-light performance, and color fidelity. The infrared camera module 2 is the same as the regular module 2, except it does not employ an infrared filter. The Pi NoIR camera is very useful for night photography and video capturing. The high-quality camera is a 12.3 Megapixel Sony IMX477 sensor that supports C- and CS-mount lenses.

You can develop many exciting projects with the Raspberry Pi camera module by just installing a camera module with your tiny pocket computer, you can explore the vast world of image processing, video processing, and even machine learning.

In this experiment, we explore how to get started with the camera module and control it using Python.

What you will need?

Raspberry Pi computer with a Camera Module port:

All current models of Raspberry Pi have a port for connecting the Camera Module.

In this experiment, our exclusive focus is on the Camera Module port of the Raspberry Pi. As participants have previously learned about some of the other ports in a previous experiment 3, we will delve into the Camera Module port, as depicted in Figure 1.

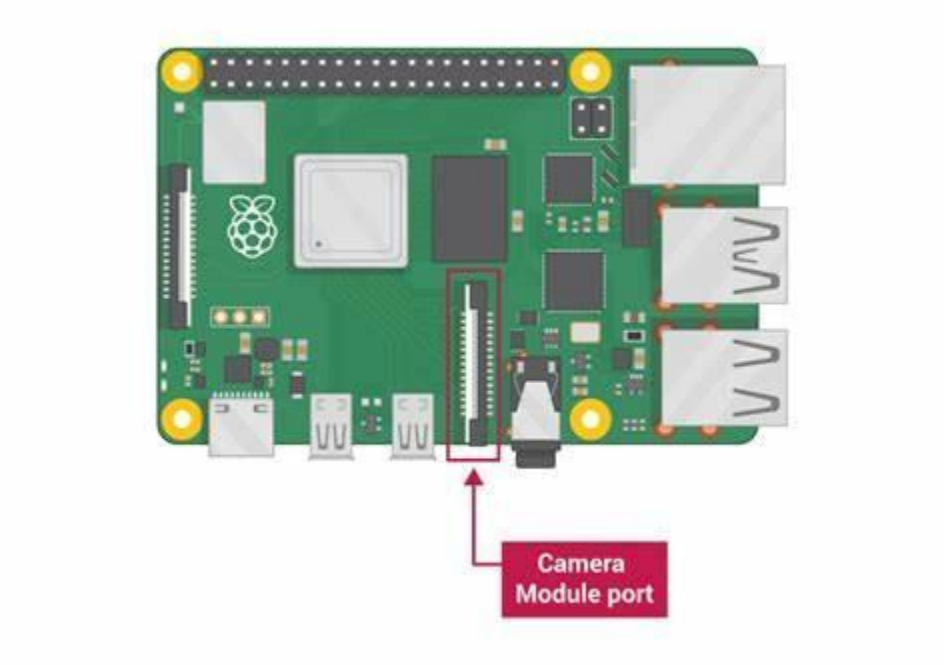


Fig. 1: The camera port in Raspberry Pi

Raspberry Pi Camera Modules:

Raspberry Pi, the versatile single-board computer, offers a range of camera modules that have evolved over the years. These modules allow users to capture images and videos directly using the Raspberry Pi's processing power and memory. This section provides an overview of the different Raspberry Pi camera modules available.

Original 5-Megapixel Model: In 2013, Raspberry Pi introduced the original 5-megapixel camera module (Fig. 2), a groundbreaking addition to the platform. This module served as the initial foray into Raspberry Pi-powered photography and video. It had a 5-megapixel OmniVision OV5647 sensor that could capture still images up to 2592 x 1944 pixels and record video up to 1080p30. It also had a fixed-focus lens that could be replaced with other lenses for different effects. The original camera module was compatible with all models of Raspberry Pi.

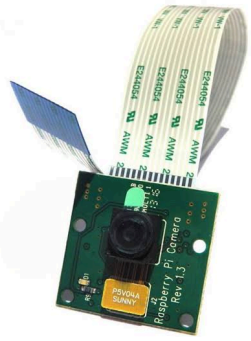


Fig. 2: 5 - Megapixel Model

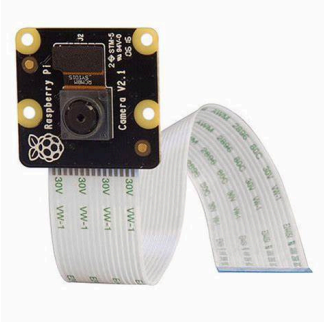


Fig. 3: 8-Megapixel Camera Module 2

8-Megapixel Camera Module 2: The Camera Module 2 followed in 2016, offering an upgraded 8-megapixel sensor, improved image quality, and versatility for a wide range of applications. The Camera Module 2 (Fig. 3) followed in 2016, offering an upgraded 8-megapixel sensor, improved image quality, and versatility for a wide range of applications. It had an 8-megapixel Sony IMX219 sensor that could capture still images up to 3280 x 2464 pixels and record video up to 1080p30, 720p60, or VGA90. It also had a fixed-focus lens that could be replaced with other lenses for different effects. The Camera Module 2 was compatible with all models of Raspberry Pi.

12-Megapixel Camera Module 3: The most recent addition to the lineup, the 12-megapixel Camera Module 3 (Fig. 4), was introduced in 2023. This high-resolution camera module opens up new possibilities for Raspberry Pi enthusiasts. It has a 12-megapixel Sony IMX477 sensor that can capture still images up to 4056 x 3040 pixels and record video up to 4K30, 1080p60, or 720p120. It also has an autofocus lens that can adjust the focus automatically or manually. The Camera Module 3 is compatible with all models of Raspberry Pi.

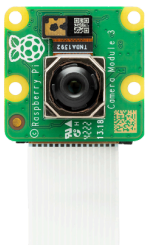


Fig. 4: 12-Megapixel Camera Module 3

These camera modules are available in both visible light and infrared versions, offering flexibility for various lighting conditions and applications. The infrared version, known as NoIR, is particularly useful for low-light and dark environments. The Camera Module 3 also comes in both standard and wide Field of View (FoV) variants, providing options for different scene-capturing needs.

All official Raspberry Pi Camera Modules connect to the CSI (Camera Serial Interface) port of the Raspberry Pi. The CSI port is typically situated near the HDMI port, positioned between the audio port and the HDMI port on most Raspberry Pi models. Connecting the camera module is a straightforward process using a ribbon cable, with the cable's shiny contacts facing away from the USB ports. We will see the CSI port and how to connect in the later part of the experiment. It's worth noting that the CSI port is compatible with all Raspberry Pi models except the Raspberry Pi 400 and the 2016 launch version of the Raspberry Pi Zero.

Global Shutter Camera:

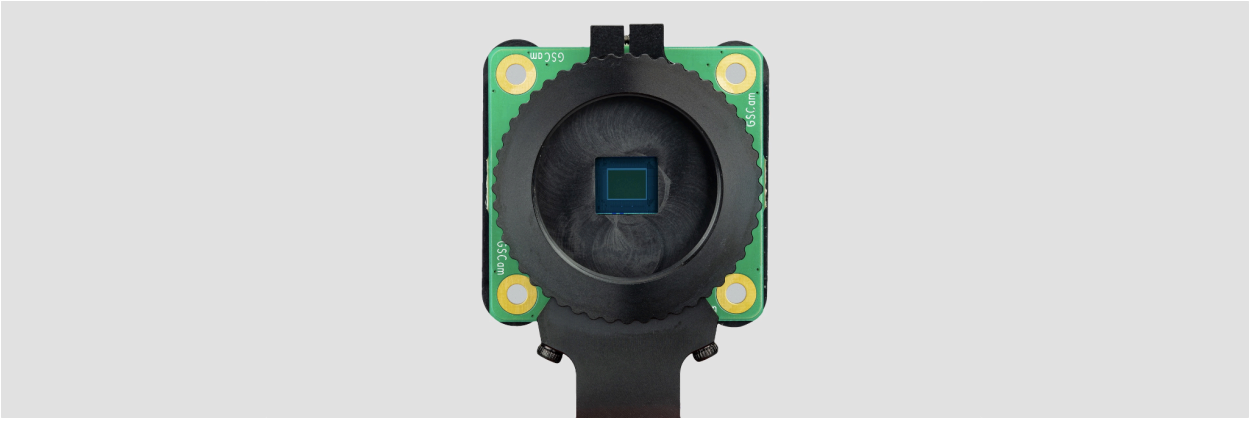


Fig. 5: Global Shutter Camera

A recent addition to the camera module lineup is the Global Shutter Camera (Fig. 5). This module employs a unique method of capturing images, allowing it to capture light from every pixel in the scene simultaneously, rather than scanning line by line. This approach minimizes distortion when capturing fast-moving objects or synchronizing multiple cameras. Despite its distinct image-capturing technique, the Global Shutter Camera also connects to the CSI port of the Raspberry Pi, but it utilizes a smaller connector than other camera modules. For more info you can visit here: [Raspberry Pi Documentation - Camera]

Raspberry Pi Camera Interface:

The Raspberry Pi's Camera Interface offers two primary connector variants: the 15-pin and 22-pin connectors. The 15-pin connector is commonly found on standard Raspberry Pi models like the A&B series and Pi camera modules, while the 22-pin connector is used by the Raspberry Pi Zero-W and Compute Module IO Board. Arducam also employs modified versions of these connectors to accommodate various camera board designs with different Flexible Printed Circuit (FPC) contact positions.

15-Pin Connector:

The 15-pin connector is the default choice for most Raspberry Pi camera applications as shown in Fig. 6. You may encounter at least three distinct variations of the 15-pin connectors in Pi-camera-related hardware:

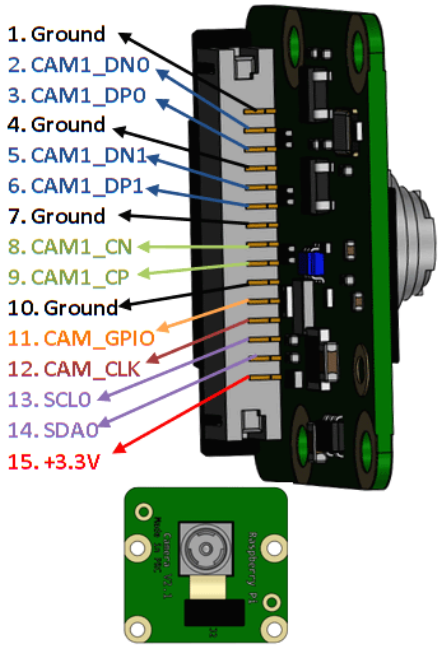


Fig. 6: 15-pin connector pin

Regardless of the connector model, the pinout remains consistent for these camera connectors. The following table outlines the pin definitions for the camera board, specifying whether the signals are input or output, with corresponding functions on the Raspberry Pi board:

Pin No.	Name	Description
1	GND	Ground
2	CAM_DN0	MIPI Data Lane 0 Negative
3	CAM_DP0	MIPI Data Lane 0 Positive
4	GND	Ground
5	CAM_DN1	MIPI Data Lane 1 Negative
6	CAM_DP1	MIPI Data Lane 1 Positive

7	GND	Ground
8	CAM_CN	MIPI Clock Lane Negative
9	CAM_CP	MIPI Clock Lane Positive
10	GND	Ground
11	CAM_IO0	Power Enable
12	CAM_IO1	LED Indicator
13	CAM_SCL	I2C SCL
14	CAM_SDA	I2C SDA
15	CAM_3V3	3.3V Power Input

Here's a simplified summary of the pins that share the same functions:

Data Lanes (Data Transmission):

- CAM_DN0 (MIPI Data Lane 0 Negative) and CAM_DP0 (MIPI Data Lane 0 Positive): These pins form a differential pair used for transmitting image data from the camera module to the Raspberry Pi. MIPI (Mobile Industry Processor Interface) is a standardized protocol for data transfer between cameras and processors, ensuring high-speed and reliable data transmission.
- CAM_DN1 (MIPI Data Lane 1 Negative) and CAM_DP1 (MIPI Data Lane 1 Positive): These pins represent an additional differential pair for transmitting image data, offering greater bandwidth and enabling the transfer of complex image information.

Clock Lanes (Synchronization):

- CAM_CN (MIPI Clock Lane Negative) and CAM_CP (MIPI Clock Lane Positive): These pins are responsible for transmitting clock signals that synchronize the data transmission between the camera module and the Raspberry Pi. Synchronization is crucial to ensure that the data is received and interpreted correctly.

Ground Pins (Electrical Stability):

- GND (Ground) pins: Ground pins serve as common reference points for electrical circuits. Multiple ground pins are provided to ensure stable electrical connections and to minimize noise interference.

Power and Indicator Pins (Control):

- CAM_IO0 (Power Enable): This pin allows the Raspberry Pi to control the power supply to the camera module. By toggling this pin, you can turn the camera module on and off, effectively managing its power consumption.

- CAM_IO1 (LED Indicator): Many camera modules feature an LED indicator for signaling various states or activities, such as capturing an image or recording video. CAM_IO1 can be used to control and interface with this indicator.

I2C Pins (Communication):

- CAM_SCL (I2C Serial Clock): This pin is part of the I2C (Inter-Integrated Circuit) communication bus, and it carries the clock signal for synchronous communication between the Raspberry Pi and the camera module.
- CAM_SDA (I2C Serial Data): CAM_SDA is another I2C pin responsible for transferring data between devices on the I2C bus, enabling configuration and control of the camera module.

Power Supply (Voltage Input):

- CAM_3V3 (3.3V Power Input): This pin provides a regulated 3.3-volt power supply input to the camera module. It ensures that the camera module receives the necessary voltage to operate reliably and capture images or video.

Raspberry Pi Camera Module:

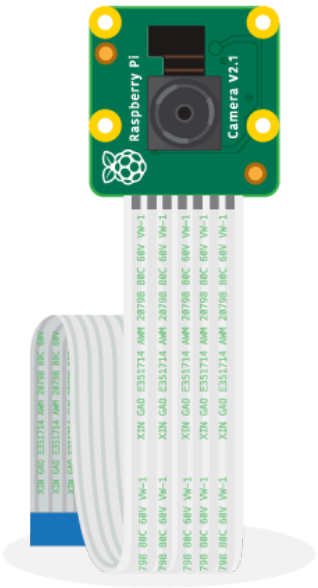


Fig. 7 : Raspberry Pi Camera Module

There are two versions of the Camera Module:

- **The standard version,** which is designed to take pictures in normal light

- **The NoIR version,** which doesn't have an infrared filter, so you can use it together with an infrared light source to take pictures in the dark.

Connecting the Camera Module:

Let's interface our pi camera with our raspberry Pi!! First, Ensure your Raspberry Pi is turned off. Now, follow the instructions below:

1. Locate the Camera Module port.
2. Gently pull up on the edges of the port's plastic clip.
3. Insert the Camera Module ribbon cable; make sure the connectors at the bottom of the ribbon cable are facing the contacts in the port.
4. Push the plastic clip back into place. A gif animation on how to interface pi cam is given below in Fig. 8.

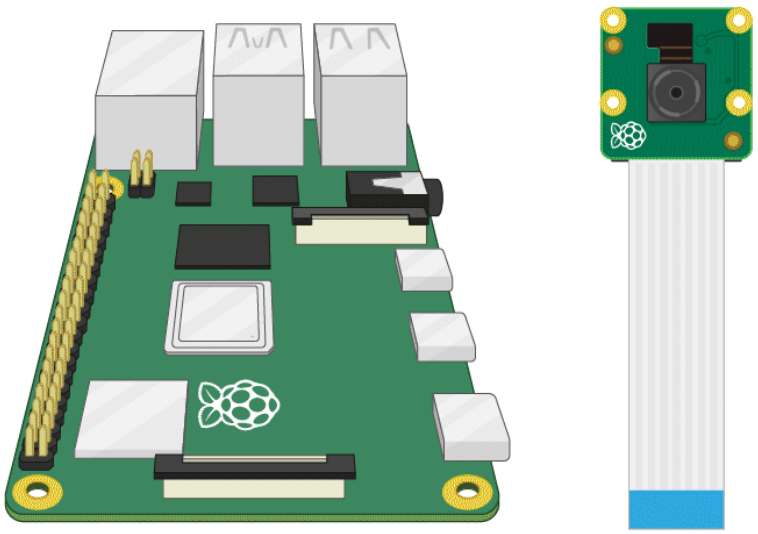


Fig. 8: Camera Module Connection Illustration

Link to above gif animation:
https://projects-static.raspberrypi.org/projects/getting-started-with-picamera/dbf2d9575be4756f79e4293a047a8a531d340710/en/images/connect-camera.gif

Now, as we have interfaced our pi camera, let's move on to the next part:

- Start up your Raspberry Pi.
- Go to the main menu and open the Raspberry Pi Configuration tool as shown in Fig. 9.

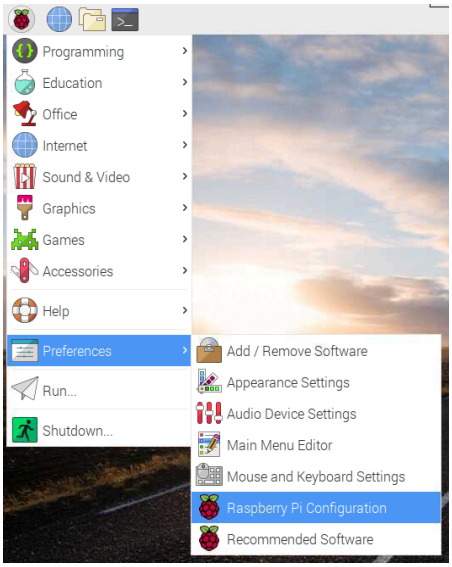


Fig. 9: Configuration tool

- Select the Interfaces tab and ensure that the camera is enabled.

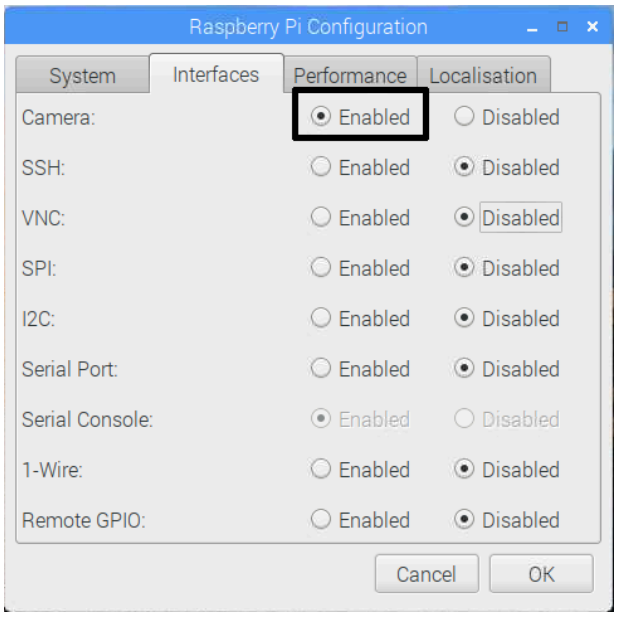


Fig. 10: Enabling the Camera Interface

If you do not see the option "Camera", then you need to open the terminal of your pi, then write

the following line: **sudo raspi-config**. A window will appear like shown below (Fig. 11):

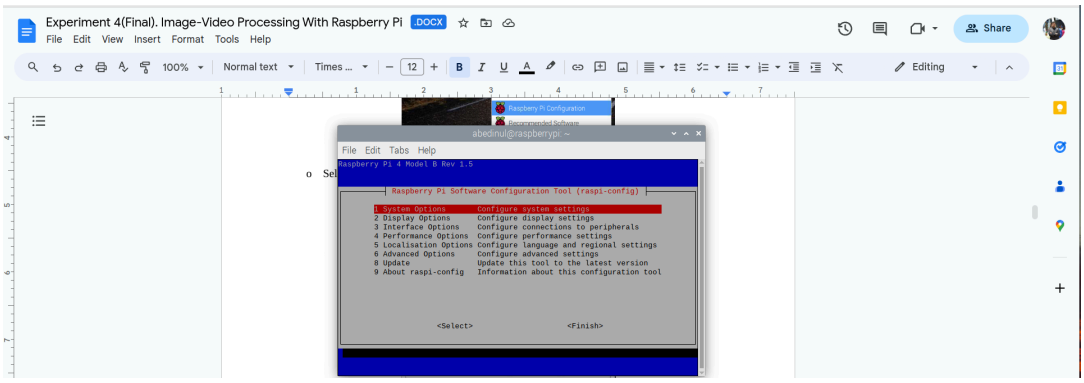


Fig. 11: Accessing Interface Options

- Select option 3: 'Interface options' and click enable camera to 'yes'. Then, click on 'Finish'.
- Reboot your Raspberry Pi.

How to control the Camera Module via the command line:

Now your Camera Module is connected and the software is enabled, try out the command line tools ***raspistill*** and ***raspivid.***

- Open a terminal window by clicking the black monitor icon in the taskbar (Fig. 12):

Fig. 12: Accessing the Terminal

- Type in the following command to take a still picture and save it to the Desktop:
  - raspistill -o Desktop/image.jpg

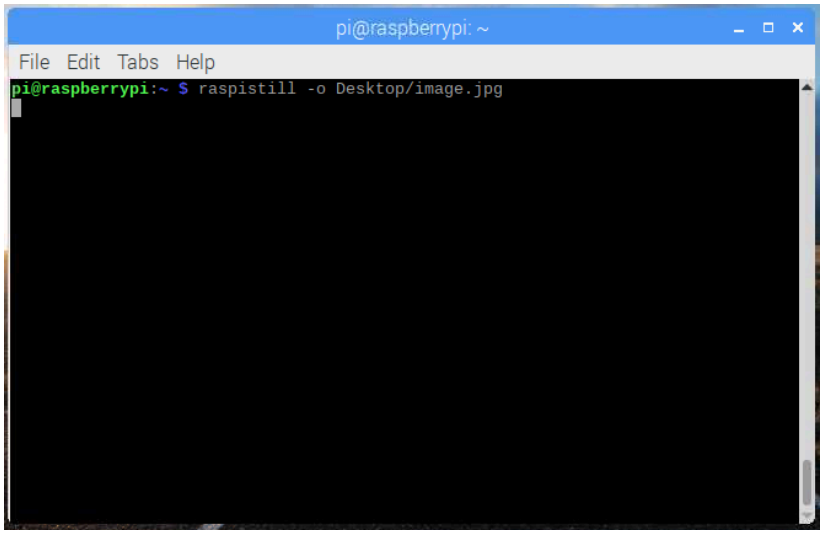


Fig. 13: Taking a Photo with raspistill

- Press Enter to run the command.

When the command runs, you can see the camera preview open for five seconds before a still picture is taken.

- Look for the picture file icon on the Desktop, and double-click the file icon to open the picture.

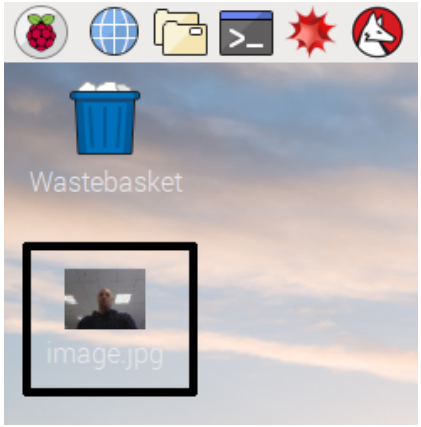


Fig. 14: Opening the Photo on the Desktop

By adding different options, you can set the size and look of the image the raspistill command takes.

- For example, add **-h** and **-w** to change the height and width of the image.
    - raspistill -o Desktop/image-small.jpg -w 640 -h 480

- Now record a video with the Camera Module by using the following raspivid command in the terminal:

    - raspivid -t 10000 -o Desktop/video.h264

  This records a ten-second video (10,000 milliseconds) at the 1920 × 1080 resolution.

- In order to play the video file, double-click the video.h264 file icon on the Desktop to open it in VLC Media Player.

For more information and other options you can use with these commands, read the documentation for raspistill and the documentation for raspivid. If you want to see all the command for camera you can click here: Raspberry Pi Camera Board - RaspiStill Command List | The Pi Hut

How to control the Camera Module with Python code:

The Python ***picamera*** library allows you to control your Camera Module and create amazing projects.

- Open a Python 3 editor, such as Thonny Python IDE:

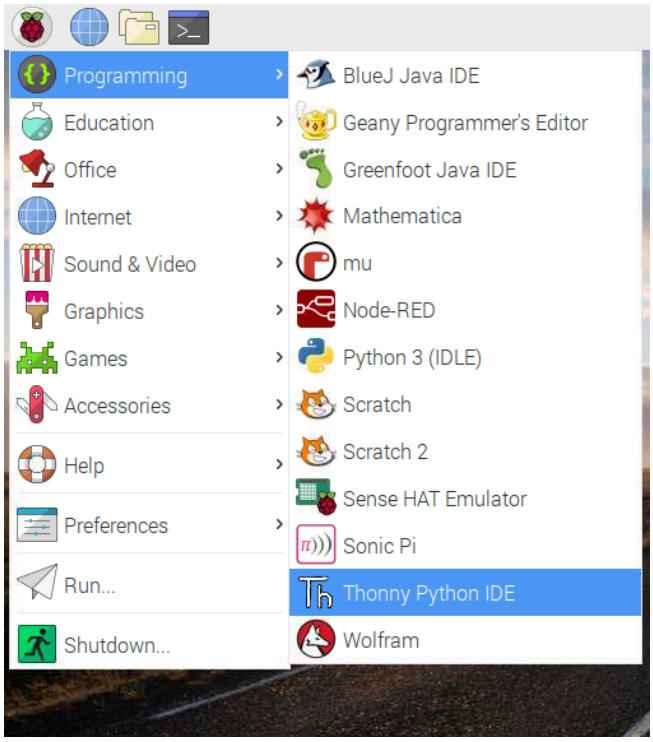


Fig. 15: Preparing Python 3 Environment

- Open a new file and save it as ***camera.py***

[Note: it's important that you never save the file as picamera.py.]

- Enter the following code:

```
from picamera import PiCamera
from time import sleep
camera = PiCamera()
camera.start_preview()
sleep(5)
camera.capture('/home/pi/Desktop/image.jpg')
camera.stop_preview()
```

- Save and run your program. The camera preview should be shown for five seconds, take an image and then close again.

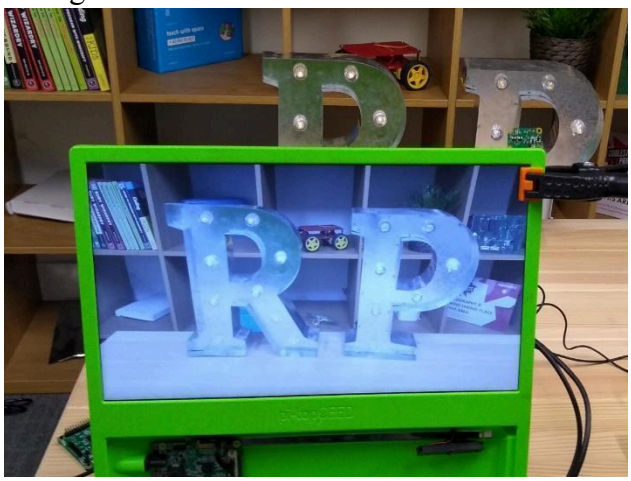


Fig. 16: Camera Preview

[Note: the camera preview only works when a monitor is connected to your Raspberry Pi. If you are using remote access (such as SSH or VNC), you won't see the camera preview.]

- If your preview is upside-down, you can rotate it by 180 degrees with the following code: `camera.rotation = 180`
- You can rotate the image by **90**, **180** or **270** degrees. To reset the image, set **rotation** to **0** degrees.

It's best to make the preview slightly see-through so you can see whether errors occur in your program while the preview is on.

- Make the camera preview see-through by setting an **alpha** level: `camera.start_preview(alpha=200)`

The **alpha** value can be any number between **0** and **255**.

*[Note: it's important to **sleep** for at least two seconds before capturing an image, because this gives the camera's sensor time to sense the light levels.]*

- Now add a loop to take five pictures in a row:

```
camera.start_preview()
for i in range(5):
    sleep(5)
    camera.capture('/home/pi/Desktop/image%s.jpg' % i)
camera.stop_preview()
```

The variable **i** counts how many times the loop has run, from **0** to **4**. Therefore, the images get saved as **image0.jpg**, **image1.jpg**, and so on.

- Run the code and hold the Camera Module in position.

The camera should take one picture every five seconds. Once the fifth picture is taken, the preview closes.

Look at your Desktop to find the five new pictures.

Record Video with Python:

Now let's record a video!

Amend your code to remove **capture()** and instead add **start_recording()** and **stop_recording()**.

Your code should look like this now:

```
camera.start_preview()
camera.start_recording('/home/pi/Desktop/video.h264')
sleep(5)
camera.stop_recording()
camera.stop_preview()
```

- Run the code.

Your Raspberry Pi should open a preview, record 5 seconds of video, and then close the preview

How to change the image settings and image effects:

The Python **picamera** software provides a number of effects and configurations to change how your images look.

[Note: some settings only affect the preview and not the captured image, some affect only the captured image, and many others affect both.]

Set the image resolution:

You can change the **resolution** of the image that the Camera Module takes. By default, the image resolution is set to the resolution of your monitor. The maximum resolution is 2592×1944 for still photos, and 1920×1080 for video recording.

- Use the following code to set the **resolution** to maximum and take a picture.

```
camera.resolution = (2592, 1944)
camera.framerate = 15
camera.start_preview()
sleep(5)
camera.capture('/home/pi/Desktop/max.jpg')
camera.stop_preview()
```

[Note: you also need to set the frame rate to 15 to enable this maximum resolution.]

The minimum resolution is 64×64.

- Try taking a picture with the minimum resolution.

Add text to your image:

You can add text to your image using the command **annotate_text**

Run this code to try it:

```
camera.start_preview()
camera.annotate_text = "Hello world!"
sleep(5)
camera.capture('/home/pi/Desktop/text.jpg')
camera.stop_preview()
```

Change the look of the added text:

Set the text size with the following code:

```
camera.annotate_text_size = 50
```

You can set the text size to anything between 6 to 160. The default size is 32. It's also possible to change the text color. First of all, add **Color** to your import line at the top of the program:

```
from picamera import PiCamera, Color
```

Then below the import line, amend the rest of your code so it looks like this:

```
camera.start_preview()
camera.annotate_background = Color('blue')
camera.annotate_foreground = Color('yellow')
camera.annotate_text = " Hello world "
sleep(5)
camera.stop_preview()
```

Change the brightness of the preview:

You can change how bright the preview appears. The default brightness is **50**, and you can set it to any value between **0** and **100.**

Run the following code to try this out:

```
camera.start_preview()
camera.brightness = 70
sleep(5)
camera.capture('/home/pi/Desktop/bright.jpg')
camera.stop_preview()
```

The following loop adjusts the brightness and also adds text to display the current brightness level:

```
camera.start_preview()
for i in range(100):
    camera.annotate_text = "Brightness: %s" % i
    camera.brightness = i
    sleep(0.1)
camera.stop_preview()
```

Change the contrast of the preview:

Similarly to the preview brightness, you can change the contrast of the preview. Run the following code to try this out:

```
camera.start_preview()
for i in range(100):
    camera.annotate_text = "Contrast: %s" % i
    camera.contrast = i
    sleep(0.1)
camera.stop_preview()
```

Add cool image effects:

You can use **camera.image_effect** to apply a particular image effect.

The image effect options are:

- none
- negative
- solarize
- sketch
- denoise
- emboss
- oilpaint
- hatch
- gpen
- pastel
- watercolor
- film
- blur
- saturation
- colorswap
- washedout
- posterise
- colorpoint
- colorbalance
- cartoon
- deinterlace1
- deinterlace2

The default effect is **none**. Pick an image effect and try it out:

```
camera.start_preview()
camera.image_effect = 'colorswap'
sleep(5)
camera.capture('/home/pi/Desktop/colorswap.jpg')
camera.stop_preview()
```

- Run this code to loop over all the image effects with **camera.image_effects**:

```
camera.start_preview()
for effect in camera.image_effects:
```

```
    camera.image_effect = effect
    camera.annotate_text = "Effect: %s" % effect
    sleep(5)
camera.stop_preview()
```

Fig. 17: Different image effect

Set the image exposure mode:

You can use **camera.exposure_mode** to set the exposure to a particular mode.

The exposure mode options are:

- ↪ off
- ↪ auto
- ↪ night
- ↪ nightpreview
- ↪ backlight
- ↪ spotlight
- ↪ sports
- ↪ snow
- ↪ beach
- ↪ verylong
- ↪ fixedfps
- ↪ antishake
- ↪ fireworks

The default mode is **auto**.

- Pick an exposure mode and try it out:

```
camera.start_preview()
camera.exposure_mode = 'beach'
sleep(5)
camera.capture('/home/pi/Desktop/beach.jpg')
```

```
camera.stop_preview()
```

You can loop over all the exposure modes with **camera.EXPOSURE_MODES,** like you did for the image effects.

Change the image white balance:

You can use **camera.awb_mode** to set the auto white balance to a preset mode.

The available auto white balance modes are:

↪ off	↪ auto	↪ sunlight	↪ cloudy	↪ shade
↪ tungsten	↪ fluorescent	↪ incandescent	↪ flash	↪ horizon

The default is **auto**.

- Pick an auto white balance mode and try it out:

```
camera.start_preview()
camera.awb_mode = 'sunlight'
sleep(5)
camera.capture('/home/pi/Desktop/sunlight.jpg')
camera.stop_preview()
```

You can loop over all the auto white balance modes with **camera.AWB_MODES**, like you did for the image effects.

What Next?

Now you know how to use your Camera Module, you could for example:

- ❖ Add buttons to control the camera with the help of GPIO Zero Python code
- ❖ Integrate the camera with Minecraft Pi
- ❖ Post the camera's pictures to Twitter automatically.

Try these Camera Module projects to learn more:

- Create a push button stop-motion film
- Make a Minecraft photobooth
- Get Babbage bear to tweet pictures
- Build a parent detector
- Use the NoIR Camera Module to create an infrared bird box

For more information about writing Python code to control the Camera Module, see the extensive picamera documentation.

Object Detection Using Raspberry Pi:

Fig. 18: Object detection algorithm running on Raspberry Pi

Now that you know how to set up and use the camera module in Raspberry Pi, it's time to use it for something more interesting!

Did you know that we can use the processing power of Raspberry pi to run Machine Learning algorithms? For instance, we may run an established object detection algorithm based on ML that runs onboard the raspberry Pi that can detect around 80 common objects!

Of course, with knowledge of ML and Digital Image Processing, we may also make our custom model that detects any object (or even your teammates' faces) and deploy it to the raspberry Pi. However, that would be easier after you complete your **ML/DIP** courses. For now, let's focus on deploying an already built (or trained) object detection model to our raspberry pi.

After you are done, your raspberry pi should be able to detect 80 common household objects (and even a human) as shown in Fig. 18.

Introduction:

As you should have already guessed, we will be using Python for implementing and deploying our object detection model. The great thing about working with python for ML is that we may use many built-in libraries and functions to make our life easier!

For this task, we'll need the following:

Tensorflow Lite: TensorFlow Lite is an open-source deep learning framework developed by Google that is designed for mobile and embedded devices (for instance, our raspberry Pi). It is a lightweight version of the popular deep learning framework TensorFlow, optimized for resource-constrained platforms

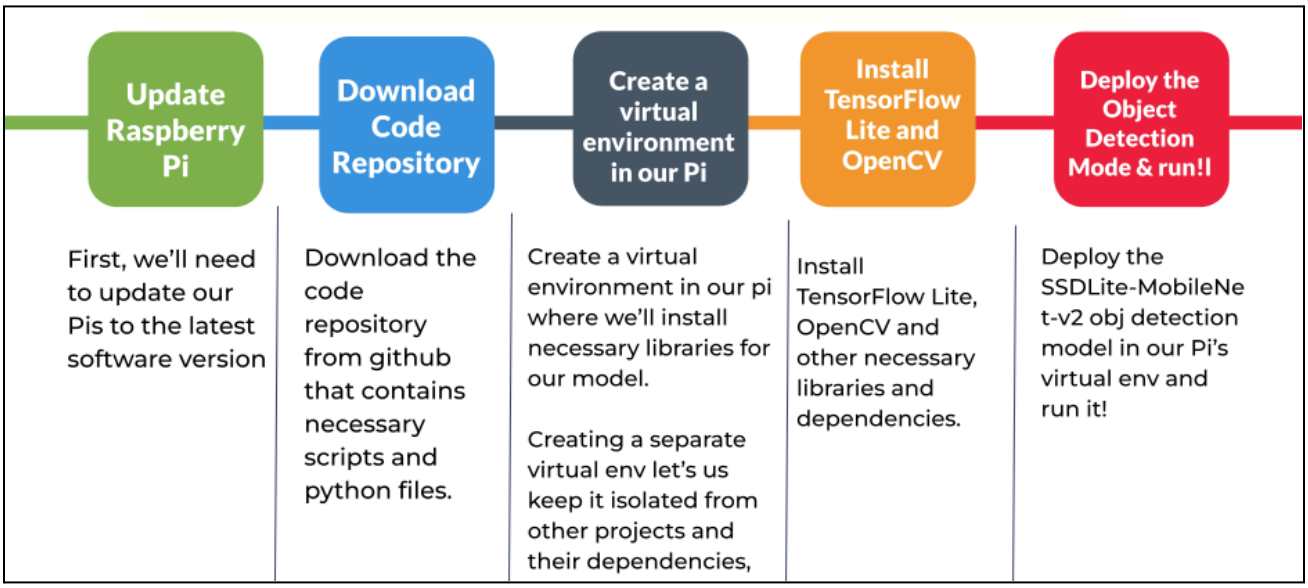
OpenCV: OpenCV, or Open Source Computer Vision Library, is an open-source computer vision and machine learning software library. We'll use this library to parse the images from the Pi camera's video and feed it to our object detection algorithm.

Raspberry Pi: Obviously, we'll need your favorite new single-board computer, Raspberry Pi to run the object detection model!

Pi cam: To get live video feed from the environment.

We'll also need various other python libraries that we'll get to in time. No need to worry about their particular use, for now.

Now, let's get started! Steps we need to follow is given below:



Step 1: Update the Raspberry Pi

First, we'll need to update our Pis to the latest software version. For this, open command Prompt in your Pi (keyboard shortcut **CTRL+ALT+T**) and issue the following commands.

```
sudo apt-get update
sudo apt-get dist-upgrade
```

Step 2: Download the Code Repository

Another great aspect of coding ML with python is that you'll get a lot of open source resources. For this, we'll use a github repository by EdjeElectronics. The repository contains scripts we'll use to run TensorFlow Lite, as well as a shell script that will make installing everything else we'll need easier. For this issue, the following command is in the same terminal.

```
git clone https://github.com/EdjeElectronics/TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi.git
```

This will download everything we need into a folder called "TensorFlow-Lite-Object-Detection -on-Android-and-Raspberry-Pi". Since the folder name is a little too long, let's rename it "objDetection" and then let's browse into that directory using our command promt so that we can work with the files inside. For this, issue the following commands,

```
mv TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi objDetection
cd objDetection
```

You should be familiar with these linux commands. "mv" is used here to rename the folder and "cd" is used to change directory to the newly renamed "objDetection" folder.

Step 3: Create a virtual environment in our Pi

In Linux, a virtual environment is a self-contained and isolated environment within which you can install and manage software packages and libraries independently of the system-wide packages. Virtual environments are particularly useful when you're working on multiple projects with different software requirements, and you want to avoid conflicts or dependencies on the system-wide software. They are commonly used in Python development.

Now, if all that was a little too hard to grasp, think of virtual environments as rooms in your home. You keep all the furniture you need for dining in your dining room so that you can dine there, right? Likewise, you have different rooms with different furniture for different work/purposes.

Think of the rooms as virtual environments, the furniture as necessary python libraries/ dependencies and the purpose of our virtual "room" is object detection!
For this let's first install virtualenv in our Pi, with the following command:

```
sudo pip3 install virtualenv
```

Then, let's create a new virtual environment (or a new room!) for our object detection model,

```
python3 -m venv objDetection-env
```

This will create a folder called objDetection-env inside the objDetection directory. The objDetection-env folder will hold all the package libraries for this environment.

Next, we'll have to activate the environment by issuing:

```
source objDetection-env/bin/activate
```

NOTE: You'll need to issue the source objDetection-env/bin/activate command from inside the /home/pi/objDetection (or wherever you've kept the git repository) directory to reactivate the environment every time you open a new terminal window (or you've shut down your Pi).

You can tell when the environment is active by checking if (objDetection-env) appears before the path in your command prompt.

Step 4: Install TensorFlow Lite and OpenCV

Next, let's install TensorFlow, OpenCV, and all other dependencies needed for the object detection mofel. OpenCV is not needed to run TensorFlow Lite, but the object detection scripts in this repository use it to grab images and draw detection results on them.

All the necessary requirements are given inside the "get_pi_requirements.sh" script. So, if we just run this shell script we should be good!

However, note that, just like time, python versions are always changing and you may need additional requirements. For instance, we'll need an additional library that isn't inside the git repository script.

Simply, add the command (sudo apt-get install libopenblas-dev) inside the "get_pi_requirements.sh" script or give it in the terminal window after running the "get_pi_requirements.sh" script.

For this, run the following commands,

```
bash get_pi_requirements.sh
sudo apt-get install libopenblas-dev
```

This downloads about 400MB worth of installation files, so it may take a while.

NOTE: If you get an error while running the bash get_pi_requirements.sh command, it's likely because your internet connection timed out, or because the downloaded package data was corrupted. If you get an error, try re-running the command a few more times.

Step 5: Set Up/Deploy the TensorFlow Lite Object Detection model

Now that we're done installing all necessary python libraries and downloading necessary git repositories, it's time to import the model itself!

We'll be using an already trained object detection model that can detect 80 objects (here's the list of objects this model can detect with class number). This model is provided by Google. It's a sample quantized SSDLite-MobileNet-v2 object detection model which is trained off the MSCOCO dataset and converted to run on TensorFlow Lite.

If you didn't understand all that, just note that we'll be using a model provided by google. We may also design/tweak our own model later (preferably after you do ML/DIP courses)!

Next, let's install TensorFlow, OpenCV, and all other dependencies needed for the object detection model. OpenCV is not needed to run TensorFlow Lite, but the object detection scripts in this repository use it to grab images and draw detection results on them.

We can download this model using the command:

```
wget https://storage.googleapis.com/download.tensorflow.org/models/tflite/coco_ssd_mobilenet_v1_1.0_quant_2018_06_29.zip
```

Then, unzip it to a folder called "Sample_TFLite_model" by issuing (this command automatically creates the folder):

```
unzip coco_ssd_mobilenet_v1_1.0_quant_2018_06_29.zip -d Sample_TFLite_model
```

Now, it's finally time to run the model!

First, check once again, that Pi cam is working properly (as per previous tutorial). Then, run the TFLite_detection_webcam.py (downloaded from the git repository we imported) by simply issuing the following command:

```
python3 TFLite_detection_webcam.py --modeldir=Sample_TFLite_model
```

If you've done everything correctly, after a few moments of initializing, a window will appear showing the webcam feed. Detected objects will have bounding boxes and labels displayed on them in real time.

But what if you've made some mistakes? Check out the common errors you may face at the end of this guide. Should help you resolve the errors.

Further Work:

Now that we've done some fun stuff with Raspberry Pi, let's get a little more comfortable! We can do some amazing things with a bit of python coding!

Option A: Make changes to an actuator/motor when you detect a certain object

Let's say you want to turn on an LED light when a human approaches your Pi cam. We can just add a bit of code inside the "**TFLite_detection_webcam.py**" inside our /home/pi/objDetection (or wherever you've kept the git repository) directory.

Humans are detected as **class number "1" (here's** the list of all classes) in our object detection model. So, we may add the following bit of code:

//to be added (or direct demo in class)

Option B: Make your own model that can detect a custom object which is not included in the 80 objects in the MSCOCO dataset.

As mentioned before. This is a bit more challenging if you haven't completed ML/DIP courses. You may not fully understand (conceptually) what you're doing. However, if you're interested, I recommended checking out the following resources:

1. Train, export, and deploy a TensorFlow Lite object detection model on the Raspberry Pi: Link How to Train TensorFlow Lite Object Detection Models Using Google Colab | SSD MobileNet

2. Custom Model Training with TensorFlow Lite: Link
   Train a custom object detection model using your data

In this case, you'll end up with a different model. So, since our model will have a different name than "Sample_TFLite_model", we'll use that name instead. For example, I would use --modeldir=My_Custom_Model_model to run my custom object detection model in **Step 5**. Step 1-4 will remain the same (you won;t need to do them again!, simply set up and deploy your new model to Pi).

Common Errors:

1. TypeError: int() argument must be a string, a bytes-like object or a number, not 'NoneType'

The 'NoneType' error means that the program received an empty array from the webcam, which typically means something is wrong with the webcam or the interface to the webcam. Try plugging and re-plugging the webcam a few times, and/or power cycling the Raspberry Pi, and see if that works. If not, you may need to try using a new webcam.

2. ImportError: No module named 'cv2'
This error occurs when you try to run any of the TFLite_detection scripts without activating the 'tflite1-env' first. It happens because Python cannot find the path to the OpenCV library (cv2) to import it.

Resolve the issue by closing your terminal window, re-opening it, and issuing:

cd objDetection
source objDetection-env/bin/activate
Then, try re-running the script as described in **Step 5**.

3. These Packages Do Not Match The Hashes From The Requirements File
This error can occur when you run the bash get_pi_requirements.sh command in Step 1c. It occurs because the package data got corrupted while downloading. You can resolve the error by re-running the bash get_pi_requirements.sh command a few more times until it successfully completes without reporting that error.

4. Unsupported data type in custom op handler: 6488064Node number 2 (EdgeTpuDelegateForCustomOp) failed to prepare.
This error occurs when trying to use a newer version of the libedgetpu library (v13.0 or greater) with an older version of TensorFlow (v2.0 or older). It can be resolved by uninstalling your current version of TensorFlow and installing the latest version of the tflite_runtime package. Issue these commands (make sure you are inside the tflite1-env virtual environment):

pip3 uninstall tensorflow
pip3 install
https://dl.google.com/coral/python/tflite_runtime-2.1.0.post1-cp37-cp37m-linux_armv7l.whl
(Or, if you're using Python 3.5, use pip3 install
https://dl.google.com/coral/python/tflite_runtime-2.1.0.post1-cp35-cp35m-linux_armv7l.whl
instead.)

Then, re-run the TFLite detection script. It should work now!

Note: the URLs provided in these commands may change as newer versions of tflite_runtime are released. Check the TFLite Python Quickstart page for download URLs to the latest version of tflite_runtime.

5. IndexError: list index out of range
This error usually occurs when you try using an "image classification" model rather than an "object detection" model. Image classification models apply a single label to an image, while

object detection models locate and label multiple objects in an image. The code in this repository is written for object detection models.

Many people run into this error when using models from Teachable Machine. This is because Teachable Machine creates image classification models rather than object detection models. To create an object detection model for TensorFow Lite, you'll have to follow the guide in this repository.

If you'd like to see how to use an image classification model on the Raspberry Pi, please see this example:
https://github.com/tensorflow/examples/tree/master/lite/examples/image_classification/raspberry_pi

Conclusion:

In conclusion, the experiment "Image/Video Processing with Raspberry Pi" has provided a solid foundation for understanding and utilizing the capabilities of the camera module with Raspberry Pi single-board computers. Through this experiment, we have explored the seamless integration of the camera module with the Raspberry Pi platform and witnessed its versatility in capturing high-quality images and videos.

Throughout the experiment, we have experienced the flexibility of the camera module, which supports different camera modes and customizable settings. This flexibility has allowed us to adapt the camera module to our specific needs and explore different applications, from capturing stunning photographs to developing advanced computer vision algorithms.

Furthermore, the experiment has emphasized the importance of the vibrant community surrounding the Raspberry Pi platform. The availability of extensive resources, tutorials, and sample projects online has facilitated our learning journey and inspired us to further explore and experiment with the camera module's capabilities.

Reference:

1. Raspberry Pi Documentation - Configuration
2. What you will need | Setting up your Raspberry Pi | Coding projects for kids and teens
3. Raspberry Pi Camera Pinout - Arducam