Step 1: What do I need?

To get started, you'll need a Raspberry Pi camera board module.

I got my 5MP Raspberry Pi camera board module from Amazon for under $30, with shipping. It's hard to believe that the camera board module is almost as expensive as the Raspberry Pi itself — but it just goes to show how much hardware has progressed over the past 5 years. I also picked up a camera housing to keep the camera safe, because why not?
Assuming you already have your camera module, you'll need to install it. Installation is very simple and instead of creating my own tutorial on installing the camera board, I'll just refer you to the official Raspberry Pi camera installation guide:

Assuming your camera board and properly installed and setup, it should look something like this:

Figure 1: Installing the Raspberry Pi camera board.

Step 2: Enable your camera module.

Now that you have your Raspberry Pi camera module installed, you need to enable it. Open up a terminal and execute the following command:

Accessing the Raspberry Pi Camera with OpenCV and Python
Shell CODE：

```
1 $ sudo raspi-config
```

This will bring up a screen that looks like this:

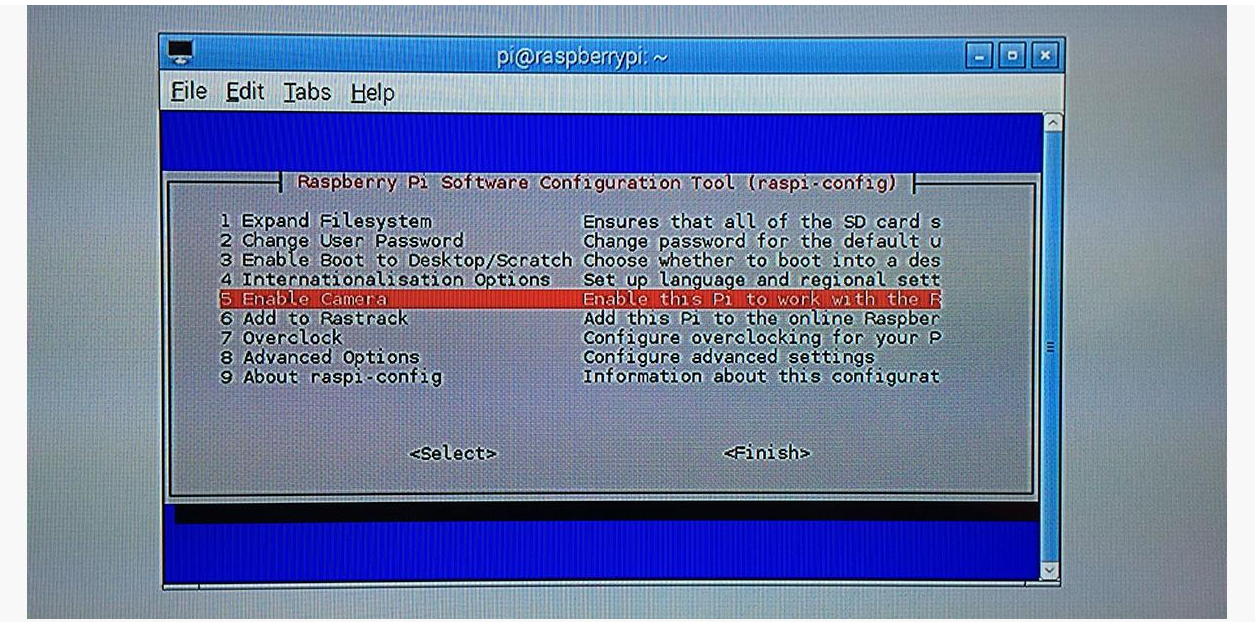

Figure 2: Enabling the Raspberry Pi camera module using the raspi-config command.

Use your arrow keys to scroll down to **Option 5: Enable camera**, **hit your enter key** to enable the camera, and then arrow down to the **Finish** button and hit enter again. Lastly, **you'll need to reboot your Raspberry Pi** for the configuration to take affect.

Step 3: Test out the camera module.

Before we dive into the code, let's run a quick sanity check to ensure that our Raspberry Pi camera is working properly.

Note: *Trust me, you'll want to run this sanity check before you start working with the code. It's* ***always good*** *to ensure that your camera is working prior to diving into OpenCV code, otherwise you could easily waste time wondering when your code isn't working correctly when it's simply the camera module itself that is causing you problems.*

Anyway, to run my sanity check I connected my Raspberry Pi to my TV and positioned it such that it was pointing at my couch:

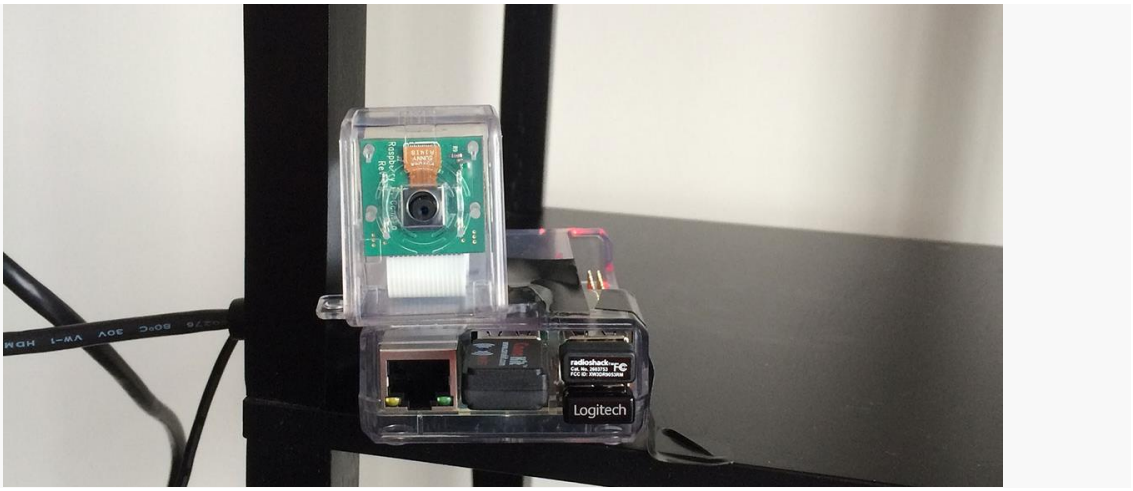

Figure 3: Example setup of my Raspberry Pi 2 and camera.

And from there, I opened up a terminal and executed the following command:


```
1 $ raspistill -o output.jpg
```

This command activates your Raspberry Pi camera module, displays a preview of the image, and then after a few seconds, snaps a picture, and saves it to your current working directory as output.jpg .

Here's an example of me taking a photo of my TV monitor (so I could document the process for this tutorial) as the Raspberry Pi snaps a photo of me:

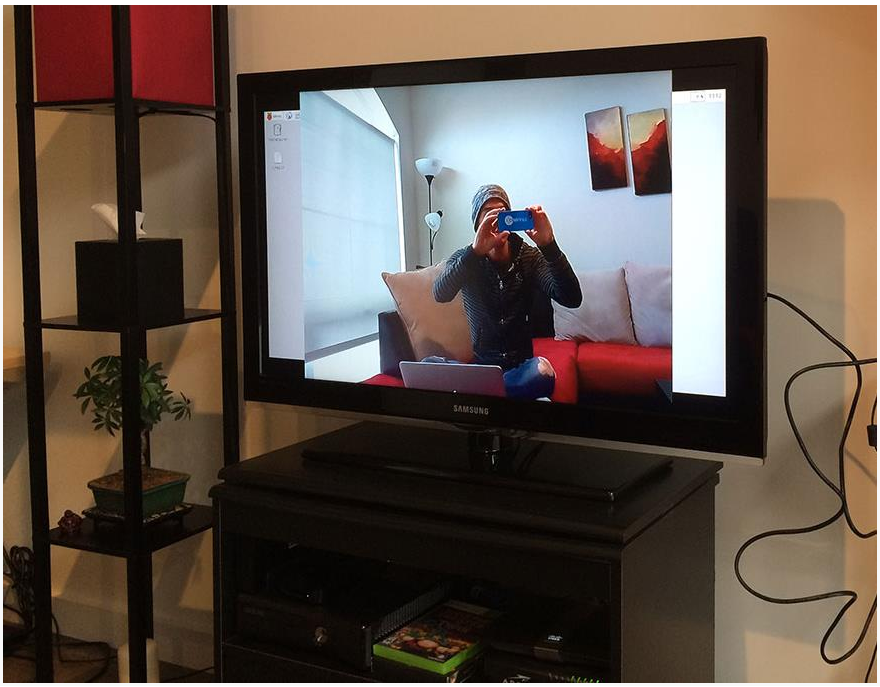

Figure 4: Sweet, the Raspberry Pi camera module is working!

And here's what output.jpg looks like:

Figure 5: The image captured using the raspi-still command.

Clearly my Raspberry Pi camera module is working correctly! Now we can move on to the some more exciting stuff.

Step 4: Installing picamera.

So at this point we know that our Raspberry Pi camera is working properly. But how do we interface with the Raspberry Pi camera module using Python?

Before installing picamera , be sure to activate our cv virtual environment:
Accessing the Raspberry Pi Camera with OpenCV and Python
Shell CODE：

```
1 $ source ~/.profile
2 $ workon cv
```

By sourcing our .profile file, we ensure that we have the paths to our virtual environments setup correctly. And from there we can access our cv virtual environment.
Note: *If you are installing the the picamera module system wide, you can skip the previous commands. However, if you are following along from the previous tutorial, you'll want to make sure you are in the cv virtual environment before continuing to the next command.*
And from there, we can install picamera by utilizing pip:
Accessing the Raspberry Pi Camera with OpenCV and Python
Shell CODE：

```
1 $ pip install "picamera[array]"
```

IMPORTANT: *Notice how I specified picamera[array] and not just picamera .*
Why is this so important?

While the standard picamera module provides methods to interface with the camera, we need the (optional) array sub-module so that we can utilize OpenCV. Remember, when using Python bindings, OpenCV represents images as NumPy arrays — and the array sub-module allows us to obtain NumPy arrays from the Raspberry Pi camera module.
Assuming that your install finished without error, you now have the picamera module (with NumPy array support) installed.

Step 5: Accessing a single image of your Raspberry Pi using Python and OpenCV.

Alright, now we can finally start writing some code!

Open up a new file, name it test_image.py , and insert the following code:
Accessing the Raspberry Pi Camera with OpenCV and Python

Python CODE:

```
 1  # import the necessary packages
 2  from picamera.array import PiRGBArray
 3  from picamera import PiCamera
 4  import time
 5  import cv2
 6
 7  # initialize the camera and grab a reference to the raw camera capture
 8  camera = PiCamera()
 9  rawCapture = PiRGBArray(camera)
10
11 # allow the camera to warmup
12 time.sleep(0.1)
13
14 # grab an image from the camera
15 camera.capture(rawCapture, format="bgr")
16 image = rawCapture.array
17
18 # display the image on screen and wait for a keypress
19 cv2.imshow("Image", image)
20 cv2.waitKey(0)
```

We'll start by importing our necessary packages on **Lines 2-5**.
From there, we initialize our PiCamera object on **Line 8** and grab a reference to the raw capture component on **Line 9**. This rawCapture object is especially useful since it (1) gives us direct access to the camera stream and (2) avoids the expensive compression to JPEG format, which we would then have to take and decode to OpenCV format anyway. I *highly recommend* that you use PiRGBArray whenever you need to access the Raspberry Pi camera — ***the performance gains are well worth it.***
From there, we sleep for a tenth of a second on **Line 12** — this allows the camera sensor to warm up.
Finally, we grab the actual photo from the rawCapture object on **Line 15** where we take special care to ensure our image is in BGR format rather than RGB. OpenCV represents images as NumPy arrays in BGR order rather than RGB — this little nuisance is subtle, but very important to remember as it can lead to some confusing bugs in your code down the line.
Finally, we display our image to screen on **Lines 19 and 20**.
To execute this example, open up a terminal, navigate to your test_image.py file, and issue the following command:
Accessing the Raspberry Pi Camera with OpenCV and Python

```
1 $ python test_image.py
```

If all goes as expected you should have an image displayed on your screen:

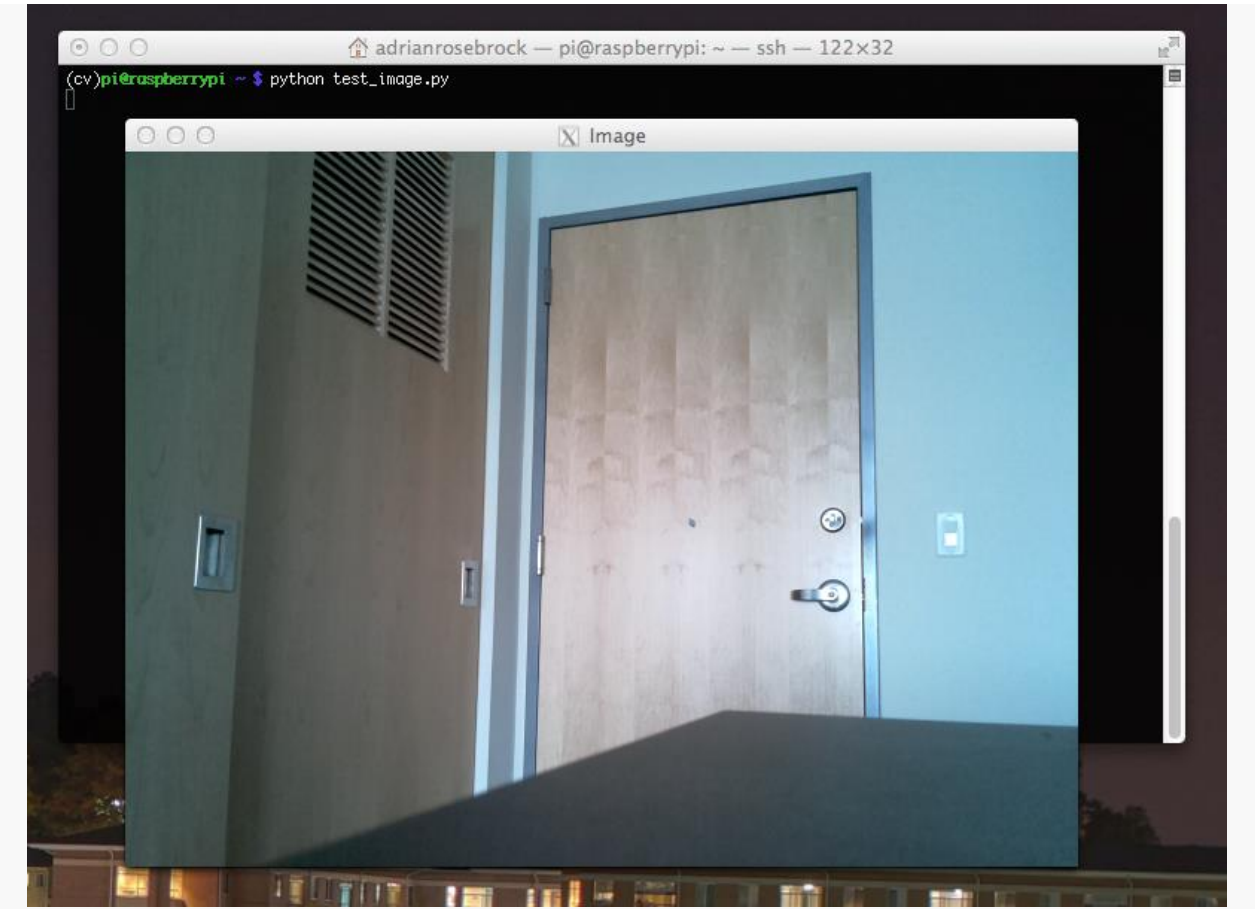

Figure 6: Grabbing a single image from the Raspberry Pi camera and displaying it on screen.

Step 6: Accessing the video stream of your Raspberry Pi using Python and OpenCV.

Alright, so we've learned how to grab a single image from the Raspberry Pi camera. But what about a video stream?

You might guess that we are going to use the `cv2.VideoCapture` function here — but I actually recommend *against* this. Getting `cv2.VideoCapture` to play nice with your Raspberry Pi is not a nice experience (you'll need to install extra drivers) and something you should generally avoid.

And besides, why would we use the cv2.VideoCapture function when we can *easily* access the raw video stream using the picamera module?

Let's go ahead and take a look on how we can access the video stream. Open up a new file, name it test_video.py , and insert the following code:

Accessing the Raspberry Pi Camera with OpenCV and Python

```
 1  # import the necessary packages
 2  from picamera.array import PiRGBArray
 3  from picamera import PiCamera
 4  import time
 5  import cv2
 6
 7  # initialize the camera and grab a reference to the raw camera capture
 8  camera = PiCamera()
 9  camera.resolution = (640, 480)
10 camera.framerate = 32
11 rawCapture = PiRGBArray(camera, size=(640, 480))
12
13 # allow the camera to warmup
14 time.sleep(0.1)
15
16 # capture frames from the camera
17 for frame in camera.capture_continuous(rawCapture, format="bgr", use_video_port=True):
18         # grab the raw NumPy array representing the image, then initialize the timestamp
19         # and occupied/unoccupied text
20         image = frame.array
21
22         # show the frame
23         cv2.imshow("Frame", image)
24         key = cv2.waitKey(1) & 0xFF
25
26         # clear the stream in preparation for the next frame
27         rawCapture.truncate(0)
28
29         # if the `q` key was pressed, break from the loop
30         if key == ord("q"):
31                 break
```

This example starts off similarly to the previous one. We start off by importing our necessary packages on **Lines 2-5**.

And from there we construct our camera object on **Line 8** which allows us to interface with the Raspberry Pi camera. However, we also take the time to set the resolution of our camera (640 x 480 pixels) on **Line 9** and the frame rate (i.e. frames per second, or simply FPS) on **Line 10**. We also initialize our PiRGBArray object on **Line 11**, but we also take care to specify the same resolution as on **Line 9**.

Accessing the actual video stream is handled on **Line 17** by making a call to thecapture_continuous method of our camera object.

This method returns a frame from the video stream. The frame then has anarray property, which corresponds to the frame in NumPy array format — all the hard work is done for us on **Lines 17 and 20!**

We then take the frame of the video and display on screen on **Lines 23 and 24**.

An important line to pay attention to is **Line 27: You *must* clear the current frame before you move on to the next one!**

If you fail to clear the frame, your Python script will throw an error — ***so be sure to pay close attention to this when implementing your own applications!***

Finally, if the user presses the `q` key, we break form the loop and exit the program.

To execute our script, just open a terminal (making sure you are in the `cv` virtual environment, of course) and issue the following command:


```
1 $ python test_video.py
```

Below follows an example of me executing the above command:

As you can see, the Raspberry Pi camera's video stream is being read by OpenCV and then displayed on screen! Furthermore, the Raspberry Pi camera shows no lag when accessing frames at 32 FPS. Granted, we are not doing any processing on the individual frames, but as I'll show in future blog posts, the Pi 2 can easily keep up 24-32 FPS even when processing each frame.