

3

Building Your Own Machine Vision

Eyes are important for humans to see a beautiful world. In this chapter, we will explore how to make a machine see by deploying a camera. We will start to understand machine vision by detecting and tracking an object model by training our machine. Several camera modules will also be reviewed:

We explore the following topics:

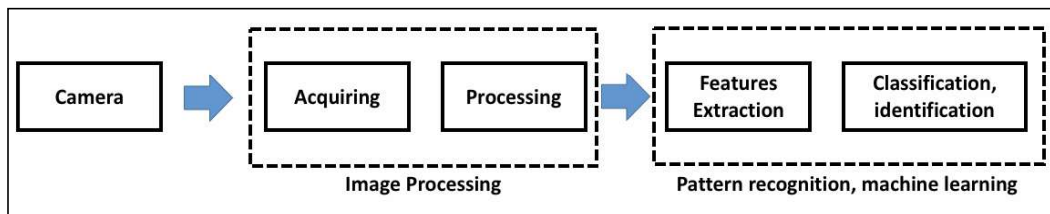
- Introducing machine vision
- Introducing OpenCV library
- Deploying OpenCV library to Raspberry Pi
- Building a simple program with OpenCV
- Working with camera modules
- Introducing pattern recognition for machine vision
- Building a tracking vision system for moving objects
- Building your own IoT machine vision

Introducing machine vision

A machine vision is a machine with camera capabilities and an understanding of what objects are. The machine uses its camera to sense physical objects around its environment. Machine vision or computer vision is a field where a machine acquires, analyzes, and understands a still image or video. This field involves knowledge such as image processing, pattern recognition, and machine learning.

The pattern recognition and machine learning fields help us to teach our machine to understand images. For instance, when we show a still image with people inside a car to the machine, then the machine should identify which are the people. Furthermore, in some cases, the machine also should guess the person in an image. From a pattern recognition and machine learning view, we should register the person so the machine can know the person in the image after identifying the person in an existing image.

To build a machine vision, we use the general design that is shown in the following figure:



Firstly, we acquire image collection from a camera. Each image will be processed for image processing tasks such as removing noise, filtering or transforming. Then, we do feature extraction for each image.

There are various feature extraction techniques depending on your purposes. After obtaining the features of images, we identify and recognize objects in an image. Pattern recognition and machine learning take part in this process.

I won't explain more about pattern recognition and machine learning. I recommend you to read a textbook related to pattern recognition and machine learning. In this chapter, I'm going to show you how to achieve machine vision by applying pattern recognition and machine learning into IoT devices.

Introducing the OpenCV library

The **OpenCV (Open Computer Vision)** library is an open source library that is designed for computational efficiency and with a strong focus on real-time applications. This library is written in C/C++ and also provides several bindings for other programming languages. The official website for OpenCV is <http://www.opencv.org>.

The OpenCV library provides a complete library starting from basic computation and image processing to pattern recognition and machine learning. I notice several research papers use this library for simulation and experiments, so this library is a good point for starting our project in machine vision/computer vision.

Currently, the OpenCV library is available for Windows, Linux, Mac, Android and iOS. You can download this library at <http://opencv.org/downloads.html>. I'll show you how to deploy OpenCV on Raspberry Pi with Raspbian OS.

Deploying OpenCV on Raspberry Pi

In this section, we will deploy the OpenCV library on Raspberry Pi. I use Raspbian Jessie for testing. We're going to install OpenCV from source code in Raspberry Pi board.ss

Let's start to build the OpenCV library from source code on Raspberry Pi. Firstly, we install development libraries. Type these commands on the Raspberry Pi terminal:

```
$ sudo apt-get update
$ sudo apt-get install build-essential git cmake pkg-config libgtk2.0-dev
$ sudo apt-get install python2.7-dev python3-dev
```

We also need to install the required matrix, image and video libraries. You can type these commands:

```
$ sudo apt-get install libjpeg-dev libtiff5-dev libjasper-dev libpng12-dev
$ sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev
$ sudo apt-get install libxvidcore-dev libx264-dev
$ sudo apt-get install libatlas-base-dev gfortran
```

The next step is to download the OpenCV source code via Git. You can type these commands:

```
$ mkdir opencv
$ cd opencv
$ git clone https://github.com/Itseez/opencv.git
$ git clone https://github.com/Itseez/opencv_contrib.git
```

We use a Python virtual environment to deploy OpenCV on Raspberry Pi using `virtualenv`. The benefit of this approach is that it isolates our existing Python development environment.

If your Raspbian hasn't installed it yet, you can install it using `pip`.

```
$ sudo pip install virtualenv virtualenvwrapper
$ sudo rm -rf ~/.cache/pip
```

After that, you configure `virtualenv` in your bash profile:

```
$ nano ~/.profile
```

Then, add the following scripts:

```
export WORKON_HOME=$HOME/.virtualenvs
source /usr/local/bin/virtualenvwrapper.sh
```

Save your bash profile file if finished.

To create a Python virtual environment, you can type this command:

```
$ mkvirtualenv cv
```

This command will create a Python virtual environment, called `cv`.

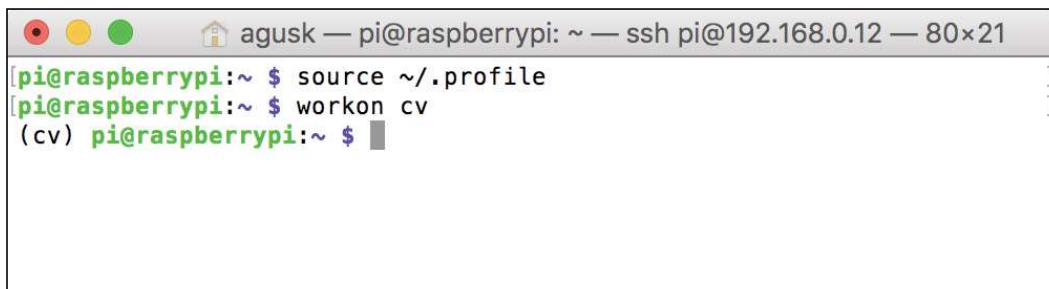
If you use Python 3, you can create it with the following command:

```
$ mkvirtualenv cv -p python3
```

You should see `(cv)` on your terminal. If you close the terminal or call a new terminal, you should activate your Python virtual environment again. Type these commands:

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

A sample of a form of Python virtual environment, called `cv`, can be seen in the following screenshot:



```
agusk — pi@raspberrypi: ~ — ssh pi@192.168.0.12 — 80x21
pi@raspberrypi:~ $ source ~/.profile ]
pi@raspberrypi:~ $ workon cv ]
(cv) pi@raspberrypi:~ $ █
```

Inside Python virtual terminal, we continue to install NumPy as the required library for OpenCV Python. We can install this library using `pip`.

```
$ pip install numpy
```

Now we're ready to build and install OpenCV from source. After cloning the OpenCV library, you can build it by typing the following commands:

```
$ cd ~/opencv/  
$ mkdir build  
$ cd build  
$ cmake -D CMAKE_BUILD_TYPE=RELEASE \  
-D CMAKE_INSTALL_PREFIX=/usr/local \  
-D INSTALL_C_EXAMPLES=ON \  
-D INSTALL_PYTHON_EXAMPLES=ON \  
-D OPENCV_EXTRA_MODULES_PATH=~/opencv/opencv_contrib/modules \  
-D BUILD_EXAMPLES=ON ..
```

Furthermore, we install the OpenCV library on our internal system from Raspbian OS.

```
$ make -j4  
$ sudo make install  
$ sudo ldconfig
```

If done, we should configure the library so Python can access it through Python binding. The following is a list of command steps for configuring with Python 2.7:

```
$ ls -l /usr/local/lib/python2.7/site-packages/  
$ cd ~/.virtualenvs/cv/lib/python2.7/site-packages/  
$ ln -s /usr/local/lib/python2.7/site-packages/cv2.so cv2.so
```

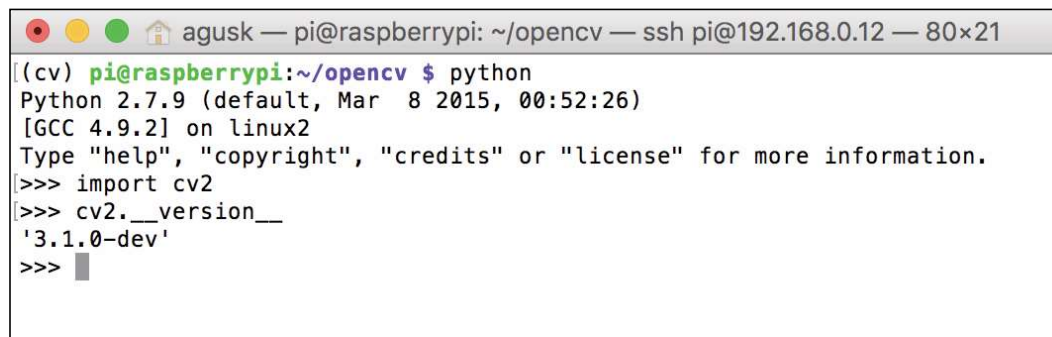
If you use Python 3.x, for instance Python 3.4, you do the following steps on the terminal. Consider if you use Python 3.4.x:

```
$ ls /usr/local/lib/python3.4/site-packages/  
$ cd /usr/local/lib/python3.4/site-packages/  
$ sudo mv cv2.cpython-34m.so cv2.so  
$ cd ~/.virtualenvs/cv/lib/python3.4/site-packages/  
$ ln -s /usr/local/lib/python3.4/site-packages/cv2.so cv2.so
```

The installation process is over. Now we need to verify whether our OpenCV installation is correct by checking the OpenCV version.

```
$ workon cv
$ python
>>> import cv2
>>> cv2.__version__
```

You should see the OpenCV version on the terminal. A sample of program output is shown in the following screenshot:

A screenshot of a terminal window on a Raspberry Pi. The window title is "agusk — pi@raspberrypi: ~/opencv — ssh pi@192.168.0.12 — 80x21". The terminal shows a Python prompt where the user runs 'python', then 'import cv2', and finally 'cv2.__version__'. The output shows the version '3.1.0-dev'.

```
(cv) pi@raspberrypi:~/opencv $ python
Python 2.7.9 (default, Mar 8 2015, 00:52:26)
[GCC 4.9.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import cv2
>>> cv2.__version__
'3.1.0-dev'
>>>
```

The next demo displays an image file using OpenCV. For this scenario, we can use the `cv2.imshow()` function to display a picture file.

For testing, log into the Raspberry Pi desktop to execute the program. Using a text editor, you can type the following scripts:

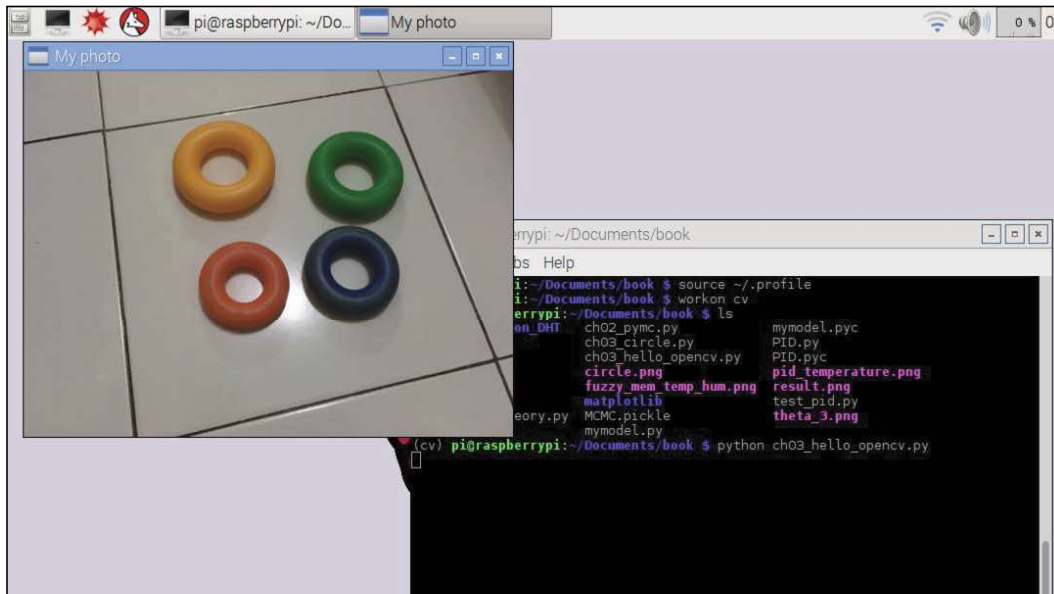
```
import numpy as np
import cv2

img = cv2.imread('circle.png')
cv2.imshow('My photo', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

I use `circle.png` file as a picture source. You can find it in this book's source codes. Save these scripts into a file called `ch03_hello_opencv.py`. Then, open the terminal inside your Raspberry Pi desktop and type this command:

```
$ python ch03_hello_opencv.py
```

If successful, you should see a dialog that displays a picture:



The picture dialog shows up because we called `cv2.waitKey(0)` in our code. Press any key on the picture dialog if you want to close the dialog.

After received a clicked key event, we close the dialog by calling the `cv2.destroyAllWindows()` function.

Building a simple program with OpenCV

There are many program samples that show how to use OpenCV using Python. In our case, we build a simple program to detect a circle in a still image.

Consider we have the following image, which is used for testing. You can find the image file in the source code files, called `circle.png`.



To find a circle in a still image, we use **circle Hough Transform (CHT)**. A circle can be defined as follows:

$$(x - a)^2 + (y - b)^2 = r^2$$

(a, b) is the center of a circle with radius r . These parameters will be computed using the CHT method.

Let's build a demo!

We will build a program to read an image file. Then, we will detect a circle form in an image using the `cv2.HoughCircles()` function.

Let's start to write these scripts:

```
import cv2
import numpy as np

print('load image')
orig = cv2.imread('circle.png')
processed = cv2.imread('circle.png', 0)
processed = cv2.medianBlur(processed, 19)

print('processing...')
circles = cv2.HoughCircles(processed, cv2.HOUGH_GRADIENT, 1, 70,
                           param1=30,
                           param2=15,
                           minRadius=0,
                           maxRadius=50)

circles = np.uint16(np.around(circles))
for (x, y, r) in circles[0, :]:
    cv2.circle(orig, (x, y), r, (0, 255, 0), 2)

print('completed')
print('writing to a file..')
cv2.imwrite('circle_process.png', orig)
print('done')
```

Save these scripts into a file called `ch03_circle.py`.

To run the program, type this command on the Raspberry Pi terminal:

```
$ python ch03_circle.py
```

Make sure `circle.png` and `ch03_circle.py` are located in the same folder.

You should see some text on the terminal. You can see the sample of program output in the following screenshot:

```
agusk — pi@raspberrypi: ~/Documents/book — ssh pi@192.168.0.12 — 80x21
[(cv) pi@raspberrypi:~/Documents/book $ ls
Adafruit_Python_DHT  ch02_pymc.py          mymodel.pyc
alpha.png            ch03_circle.py        PID.py
beta.png             ch03_hello_opencv.py PID.pyc
ch01_dht22.py        circle.png             pid_temperature.png
ch01_led.py          fuzzy_mem_temp_hum.png result.png
ch01_pid.py          matplotlib             test_pid.py
ch02_bayes_theory.py MCMC.pickle           theta_3.png
ch02_fuzzy.py        mymodel.py
[(cv) pi@raspberrypi:~/Documents/book $ python ch03_circle.py
load image
processing...
completed
writing to a file..
done
(cv) pi@raspberrypi:~/Documents/book $
```

This program will detect a circle form in an image file. After finishing the detection process, the program will generate a new image file, called `circle_process.png`.

If you open `circle_process.png` file, you should see four circle drawings in the image file, shown in the following figure:



How does it work?

Firstly, we load OpenCV and NumPy libraries into our program:

```
import cv2
import numpy as np
```

We read the image file using `cv2.imread()` into two variables, `orig` and `processed`. The `processed` variable is used to manipulate to find a circle. The image in `processed` variable will be changed due to the blurring process.

```
orig = cv2.imread('circle.png')
processed = cv2.imread('circle.png', 0)
processed = cv2.medianBlur(processed, 19)
```

`cv2.medianBlur()` is used to blur an image by defining a median value. The parameter value should be an odd value, such as 1, 3, 5, 7.

To find circles in an image, we can use `cv2.HoughCircles()`. `param1` and `param2` values, which are defined based on this paper <http://www.bmva.org/bmvc/1989/avc-89-029.pdf>.

```
circles = cv2.HoughCircles(processed, cv2.HOUGH_GRADIENT, 1, 70,
                           param1=30,
                           param2=15,
                           minRadius=0,
                           maxRadius=50)
```

Draw all circles found on the original image, the `orig` variable:

```
circles = np.uint16(np.around(circles))
for (x, y, r) in circles[0, :]:
    cv2.circle(orig, (x, y), r, (0, 255, 0), 2)
```

The last step is to save our computation result into a file called `circle_process.png`, using `cv2.imwrite()`:

```
cv2.imwrite('circle_process.png', orig)
```

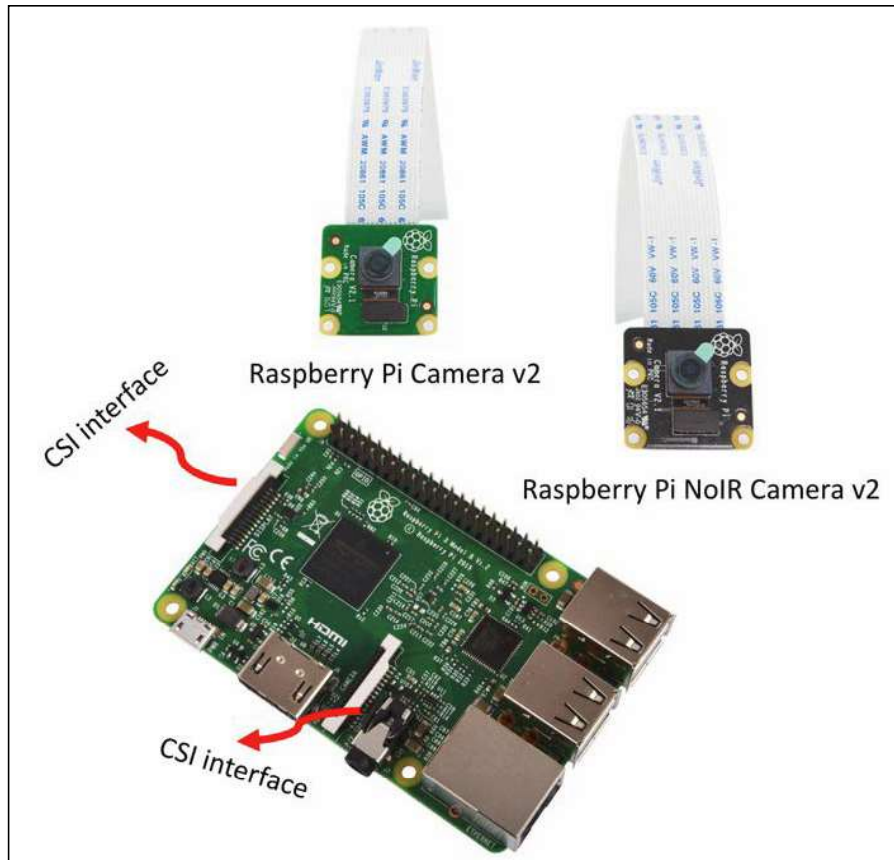
Working with camera modules

In this section, we explore various camera modules for the Raspberry Pi board. There are many camera models that fit your projects. Camera modules can be reviewed based on what kind of Raspberry Pi interface is used to attach the modules.

Let's explore.

Camera modules based on the CSI interface

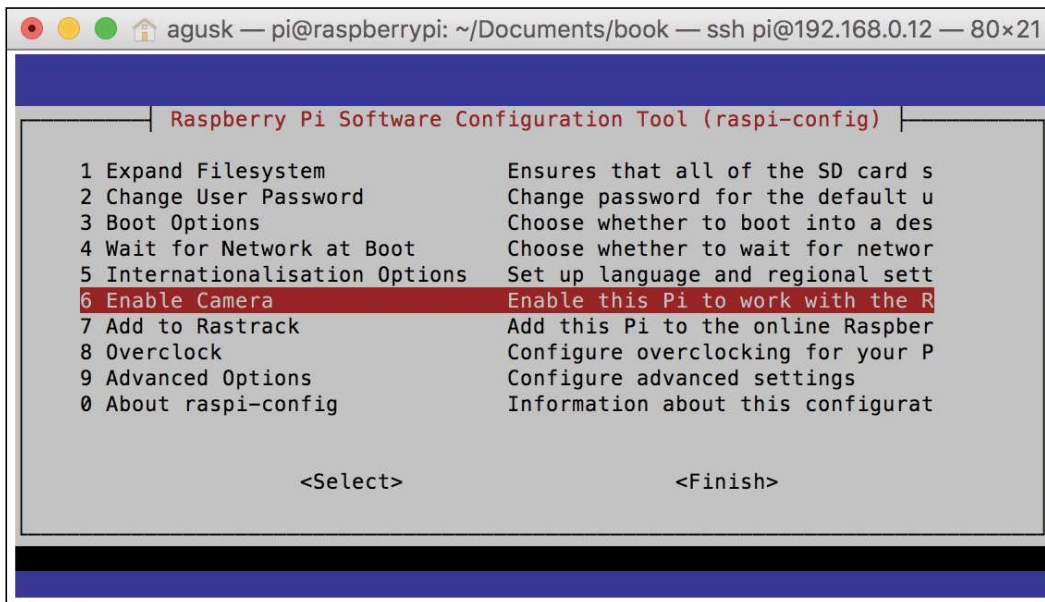
The Raspberry Pi camera is the official camera board released by the Raspberry Pi Foundation. This camera can be attached to the Raspberry Pi board through the CSI interface. The Raspberry Pi Foundation also provides another camera model, the Raspberry Pi NoIR Camera. This can work in low light (twilight) environments. A form of Raspberry Pi Camera v2 and NoIR camera v2 can be seen in the following figure:



These modules are official camera devices for Raspberry Pi. To use a camera over the CSI interface, we should enable it on Raspbian. You can configure it using the `raspi-config` tool. Just type command on the Raspberry Pi terminal.

```
$ sudo raspi-config
```

After execution, you should see the `raspi-config` program, which is shown in the following screenshot:



Select the **6 Enable Camera** option on the `raspi-config` tool. Then, click on **Enable Camera** to activate the camera modules. If finished, you should be asked to restart Raspbian. Restart Raspbian to complete the changed configuration.

Now you can use this camera with your program.

Camera modules based on USB interface

Camera modules with a USB interface are common camera devices. This device is usually called a webcam. You can easily find them in your local stores.



Image source: <http://www.amazon.com/Microsoft-LifeCam-Cinema-Webcam-Business/dp/B004ABQAFO/>

A camera module with USB can be attached to the Raspberry Pi board through the USB. For Raspberry Pi 3, you have four USB adapters, so you can attach four camera modules based on USB.

Several camera module-based USBs can be recognized by Raspberry Pi including the OpenCV library. You can find compatible USB webcams for Raspberry Pi on this site, http://elinux.org/RPi_USB_Webcams.

Camera modules-based serial interface

If your IoT boards don't have a USB interface but do have UART/serial pins, you can use a camera modules-based serial interface. The Grove-Serial Camera kit is one of these, which is shown in the following figure:

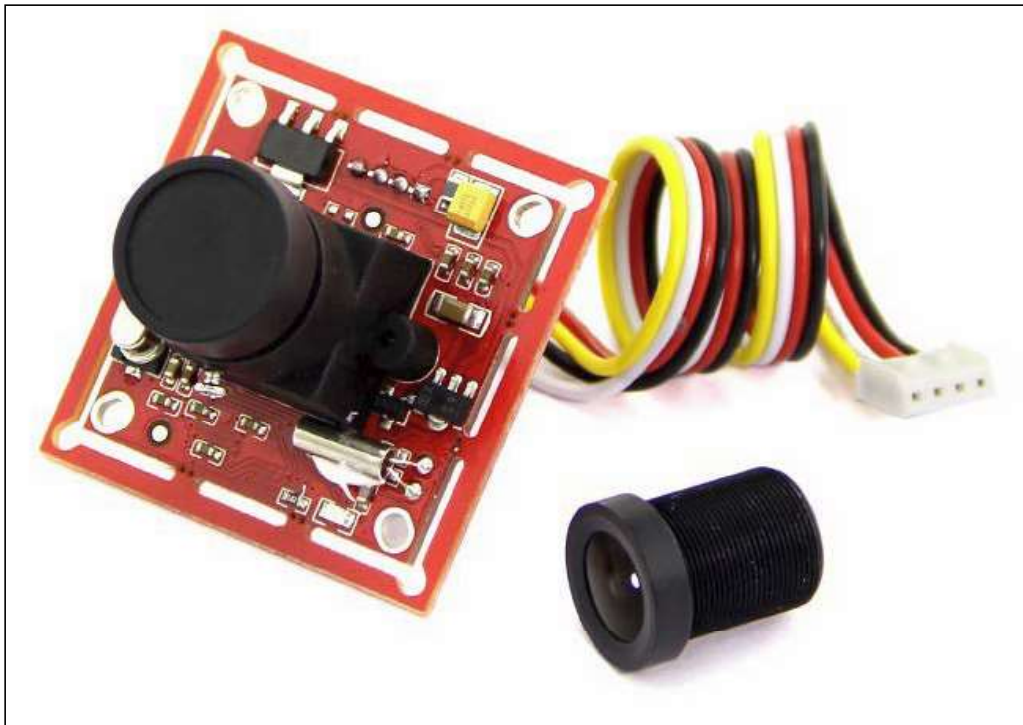


Image source: http://www.seeedstudio.com/item_detail.html?p_id=1608

A camera module with UART interface can be attached to Raspberry Pi boards through UART GPIO pins.

Camera modules with multi-interfaces

I found several camera module devices with multi-interfaces that support serial, USB, SPI and I2C interfaces. This is a good point, because we can attach them in our favorite boards.

Pixy CMUcam is one of the camera modules with a multi-interface. You can read and buy this module on the official website, <http://cmucam.org>. Some online stores also sell this module. I got Pixy CMUcam5 board and pan/tilt module from SeeedStudio, <http://www.seeedstudio.com>.



I'm going to share how to use the Pixy CMUcam5 module with a Raspberry Pi board in the last section.

Accessing camera modules from the OpenCV library

In the previous section, we used a still image as a source for the OpenCV library. We can use a camera as the source of a still image. A camera generates video data, which is a collection of still images. To access camera modules from the OpenCV library, follow these steps:

1. To access a camera from OpenCV, we can use the `VideoCapture` object. We call `read()` to read a frame, which is a still of a frame.
2. For a demo, we use the camera USB drive. Just connect this device to the Raspberry Pi board through the USB drive. Then, we write the following scripts with your text editor:

```
import numpy as np
import cv2

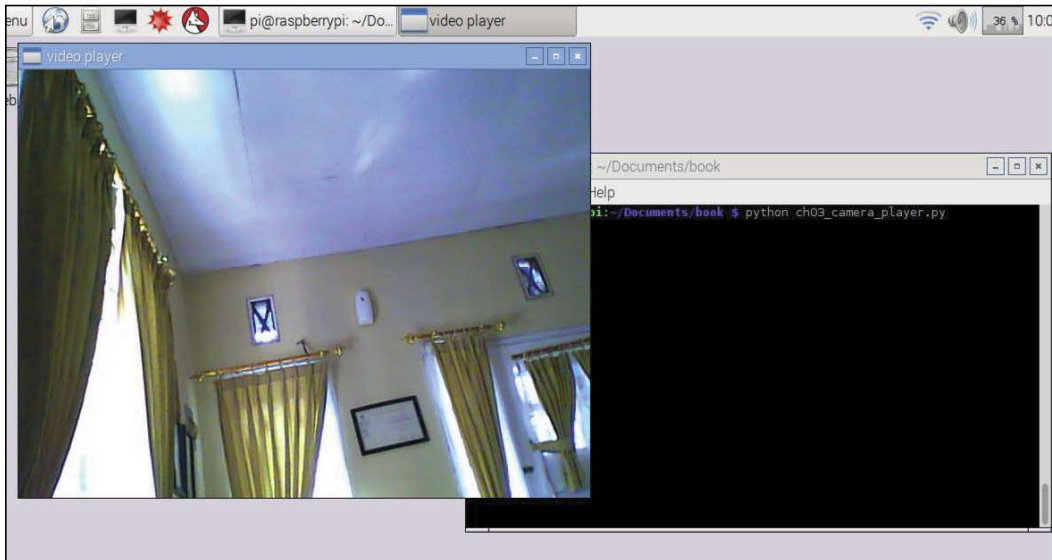
cap = cv2.VideoCapture(0)
while True:
    # Capture frame-by-frame
    ret, frame = cap.read()

    # Display the resulting frame
    cv2.imshow('video player', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

3. Save these scripts into a file, called `ch03_camera_player.py`.
4. To run this program, you should enter the Python virtual environment, which already deployed the OpenCV library.
5. Type this command:
`$ python ch03_camera_player.py`

6. If you succeed, you will have a dialog that shows streaming video from a camera. A sample of program output can be seen in the following screenshot:



7. To exit or close the dialog, you can press a key *Q*.
8. If you see the function, `cv2.VideoCapture(0)`, call the attached camera. If you attached more than one camera, call `cv2.VideoCapture(1)` for the second camera.

Introducing pattern recognition for machine vision

Pattern recognition is an important part of machine vision or computer vision, to teach a machine to understand the object in an image.

In this section, we explore a paper by Paul Viola and Michael Jones about *Rapid Object Detection using a Boosted Cascade of Simple Features*. This paper describes a machine learning approach for visual object detection.

In general, the Viola and Jones approach is known as Haar Cascades. Their algorithm uses AdaBoost algorithm with the following classifier:

$$h(x) = \begin{cases} 1 & \sum_{t=1}^T \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^T \alpha_t \\ 0 & \text{otherwise} \end{cases}$$

Fortunately, the OpenCV library has implemented Viola and Jones' approach to visual object detection. Other people also contributed to data training from Haar Cascades. You can find training data files on the OpenCV source code, which is located on `<opencv_source_codes>/data/haarcascades/`.

You can now test detection of faces on an image using the Haar Cascades approach. You can write the following scripts:

```
import numpy as np
import cv2

face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

img = cv2.imread('children.png')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

faces = face_cascade.detectMultiScale(gray, 1.3, 5)
for (x, y, w, h) in faces:
    img = cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 255), 2)

cv2.imshow('img', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Save these scripts into a file called `ch03_faces.py`.

You also put files, `haarcascade_frontalface_default.xml` and `children.png`, on the same path with the program. The `haarcascade_frontalface_default.xml` file can be obtained from the `<opencv_source_codes>/data/haarcascades/` and `children.png` file is taken from the source code of this book.

Run this program on the terminal with the Raspberry Pi desktop by typing this command:

```
$ python ch03_faces.py
```

After running, you should get a dialog with a picture. There are three faces detected, but one is missing. In my opinion, Haar Cascades approach is still good even it's not the best method. A sample of program output can be seen in the following screenshot:



How does it work?

Firstly, we load the required libraries and training data for Haar Cascades:

```
import numpy as np
import cv2
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
```

We load a picture file for testing and then change the image color to gray:

```
img = cv2.imread('children.png')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

To detect a face, we call `face_cascade.detectMultiScale()` with passing image vector, scale factor and minimum neighbors. If a detected face is found, we draw a rectangle on the picture:

```
faces = face_cascade.detectMultiScale(gray, 1.3, 5)
for (x, y, w, h) in faces:
    img = cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 255), 2)
```

The last step is to show a picture and wait pressed key. If any key is pressed, a picture dialog is closed.

```
cv2.imshow('img', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Building a tracking vision system for moving objects

In this section, we build a simple tracking vision system. We already learned how to detect a face in an image. Now we try to detect faces on video.

The idea is simple. We change a still image as source to a frame image from a camera. After calling `read()` from the `VideoCapture` object, we pass the frame image into `face_cascade.detectMultiScale()`. Then, we show it a picture dialog. That's it.

For implementation, type these scripts:

```
import numpy as np
import cv2

face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
cap = cv2.VideoCapture(0)
while True:
    # Capture frame-by-frame
    ret, frame = cap.read()

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    faces = face_cascade.detectMultiScale(gray, 1.3, 5)
    for (x, y, w, h) in faces:
        img = cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 255), 2)

    cv2.imshow('face tracking', frame)
```

```
        if cv2.waitKey(1) & 0xFF == ord('q'):  
            break  
  
    cap.release()  
    cv2.destroyAllWindows()
```

Save this program into a file called `ch03_faces_camera.py`.

Now you can run this program on the terminal from the Raspberry Pi desktop.

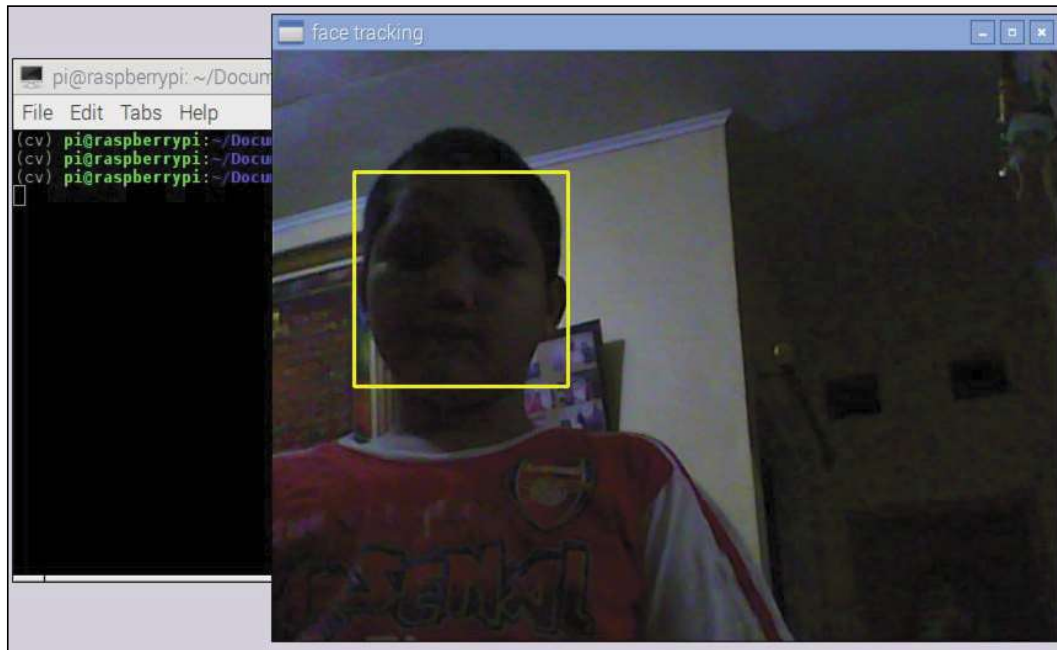
```
$ python ch03_faces_camera.py
```

After running, try to show your face. Then, the program should detect your face. You can see the program output in the following figure:

You can modify this project by adding an LED as an indicator for which a face is already detected.

How does it work?

This program runs as the previous program. We only change the image source from the camera.



Building your own IoT machine vision

We already know Pixy CMUcam5 as a camera module. In this section, we try to use this module in our IoT project.

The following is a list of the required modules:

- Pixy CMUcam5 Sensor, http://www.seeedstudio.com/item_detail.html?p_id=2048
- Pan/Tilt for Pixy, http://www.seeedstudio.com/item_detail.html?p_id=2048

You also can obtain these modules on other online stores.

Deploying Pixy CMUcam5 on Raspberry Pi

In order to use Pixy CMUcam5, you should install the required libraries and applications. Firstly, you can open the terminal on Raspberry Pi and type these commands:

```
$ sudo apt-get install libusb-1.0-0-dev
$ sudo apt-get install qt4-dev-tools
$ sudo apt-get install qt4-qmake qt4-default
$ sudo apt-get install g++
$ sudo apt-get install swig
$ sudo apt-get install libboost-all-dev
```

You need the Pixy library and application from source code. Firstly, you download the source code and then install PixyMon:

```
$ git clone https://github.com/charmedlabs/pixy.git
$ cd pixy/scripts
$ ./build_pixymon_src.sh
```

In order to use the Pixy library from Python, you can install Python binding. Type this command on the path `<pixy_library>/pixy/scripts`:

```
$ ./build_libpixyusb_swig.sh
```

You need to configure to access Pixy over USB without a non-root user. Type these commands:

```
$ cd ../src/host/linux/
$ sudo cp pixy.rules /etc/udev/rules.d/
```

Now you're ready to use Pixy CMUcam5.

Assembly

To setup Pixy CMUcam5 and Pan/Tilt, I recommend you read this guideline, http://cmucam.org/projects/cmucam5/wiki/Assembling_pantilt_Mechanism. The following is my assembly, shown in the following figure:



Updating the Pixy CMUcam5 firmware

Before you use the Pixy CMUcam5 module, I recommend you update the board firmware. You can download it on http://cmucam.org/projects/cmucam5/wiki/Latest_release. For instance, you can download Pixy firmware 2.0.19 directly on http://cmucam.org/attachments/download/1317/pixy_firmware-2.0.19-general.hex.

To update the firmware, you should run the PoxyMon application. Please unplug the Pixy CMUcam5 from Raspberry Pi. Then, press the white button on the top of the Pixy CMUcam5 board and plug the board in to Raspberry Pi through the USB. Please keep pressing the white button until you get a folder dialog. Then release the white button on Pixy CMUcam5, and select the Pixy firmware file. Wait until the flashing firmware is done.

Testing

We start to test Pixy CMUcam5 with Raspberry Pi. Several demos are provided to show how Pixy CMUcam5 works. Let's start!

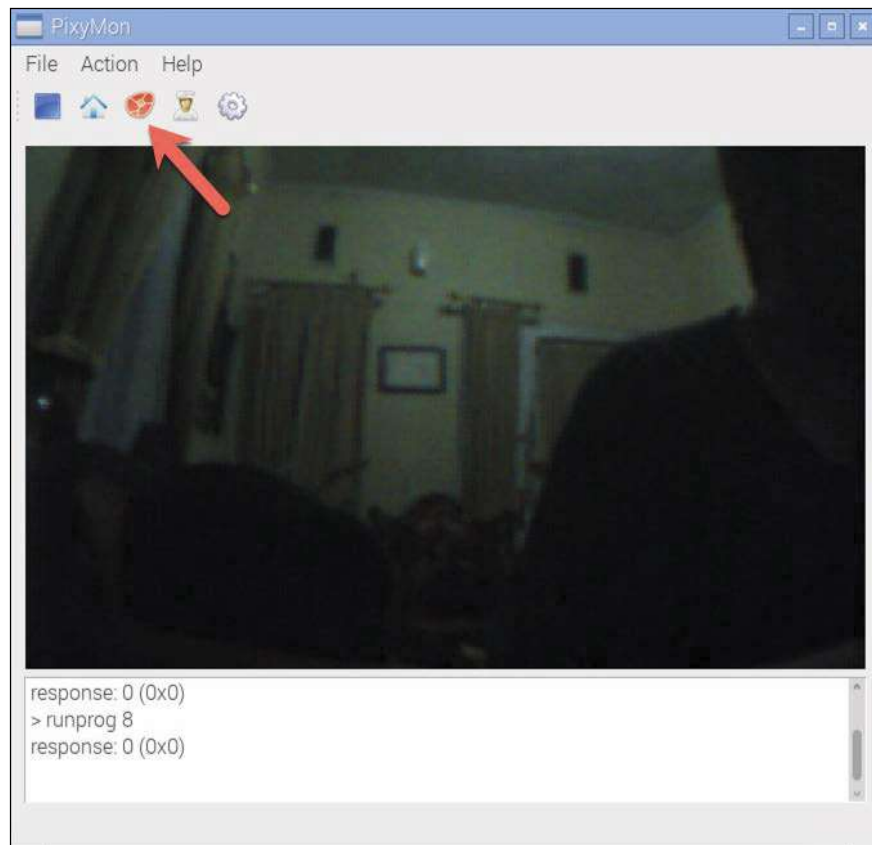
Loading streaming video

After we have deployed the Pixy CMUcam5 application and library, we will obtain the PixyMon application. It's a tool to manage our training data and can be found on `<pixy_codes>/build/pixymon/bin/`.

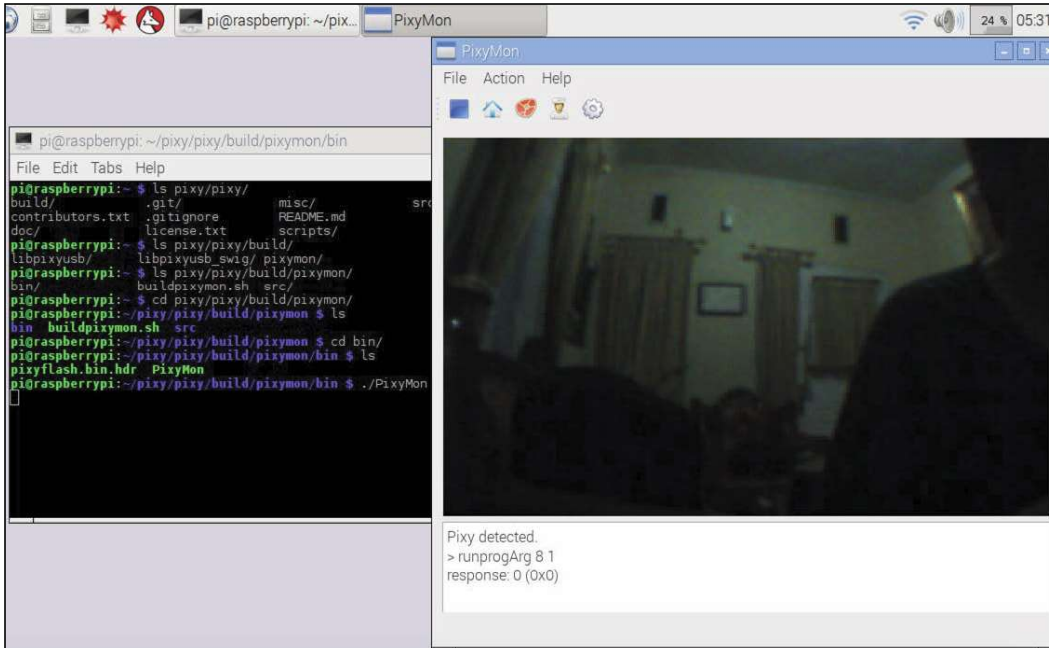
Navigate to `<pixy_codes>/build/pixymon/bin/`, then type this command on the terminal with Raspberry Pi in desktop mode.

```
$ ./PixyMon
```

If done correctly, you should see a **PixyMon** dialog, shown in the following screenshot:



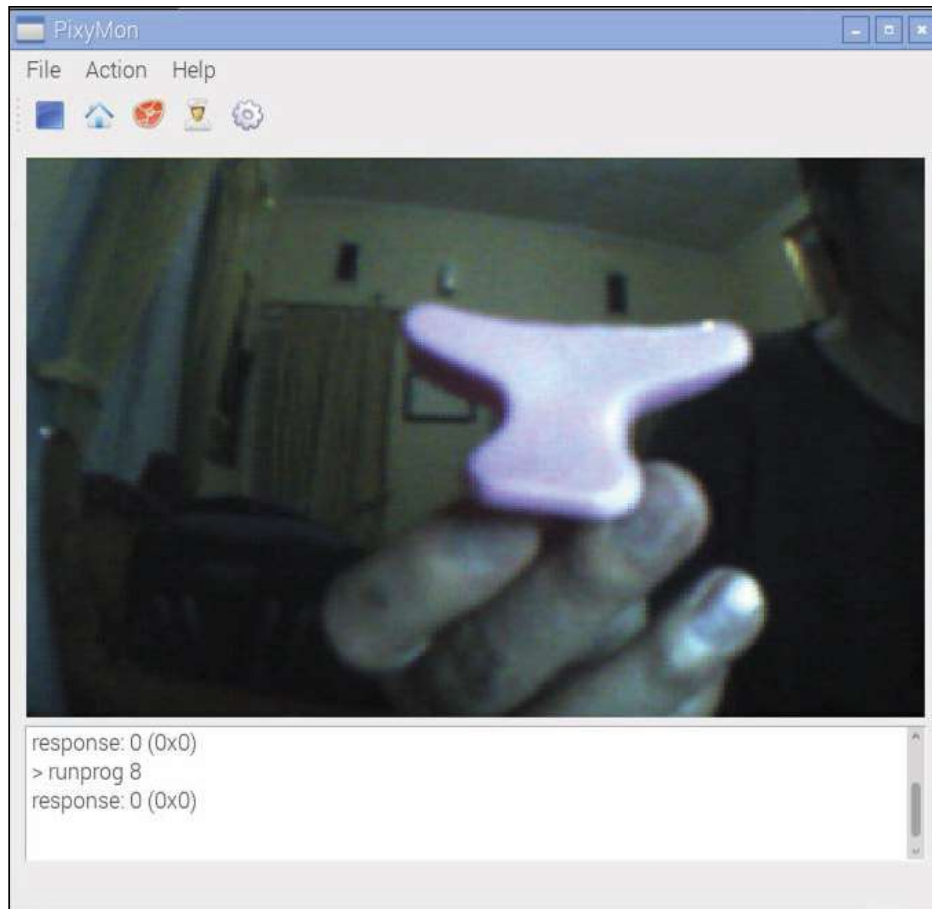
If you don't see any picture on the dialog, you can click the red circle icon, which is shown by the arrow. This puts the PixyMon application in streaming video mode. The following screenshot is a sample of my program output:



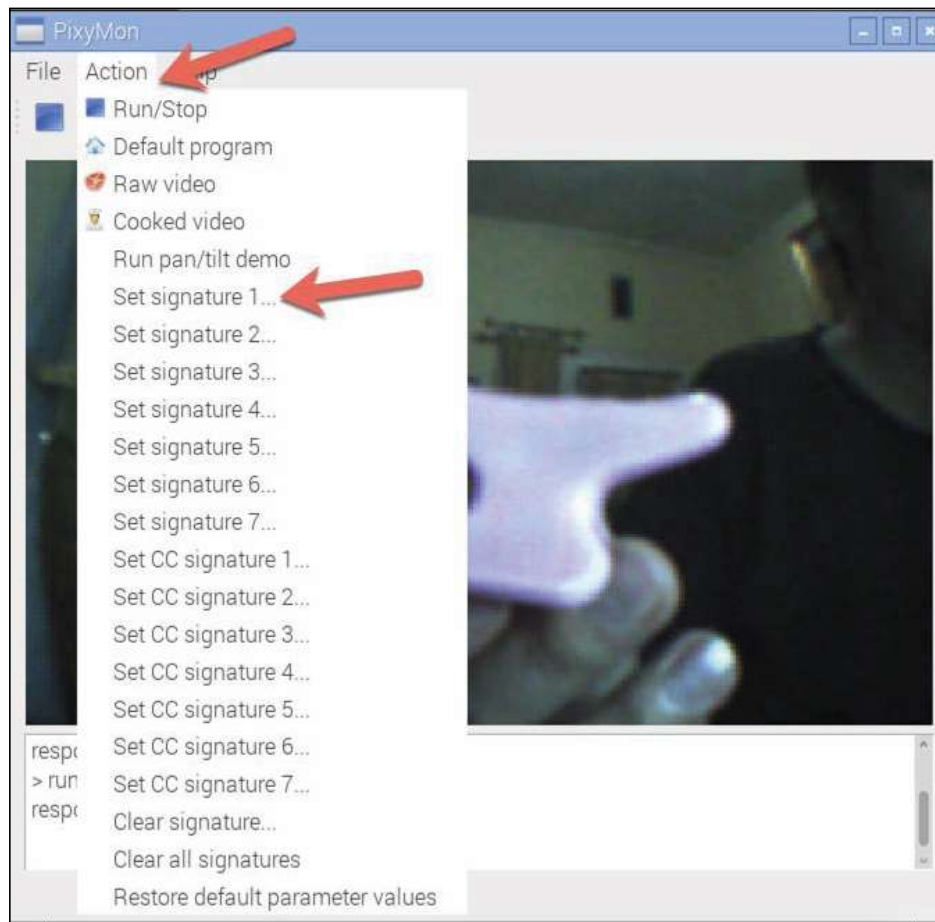
Tracking an object

Pixy CMUcam5 can track any object after the object is already registered. In this section, we explore how to register a new object and then track it.

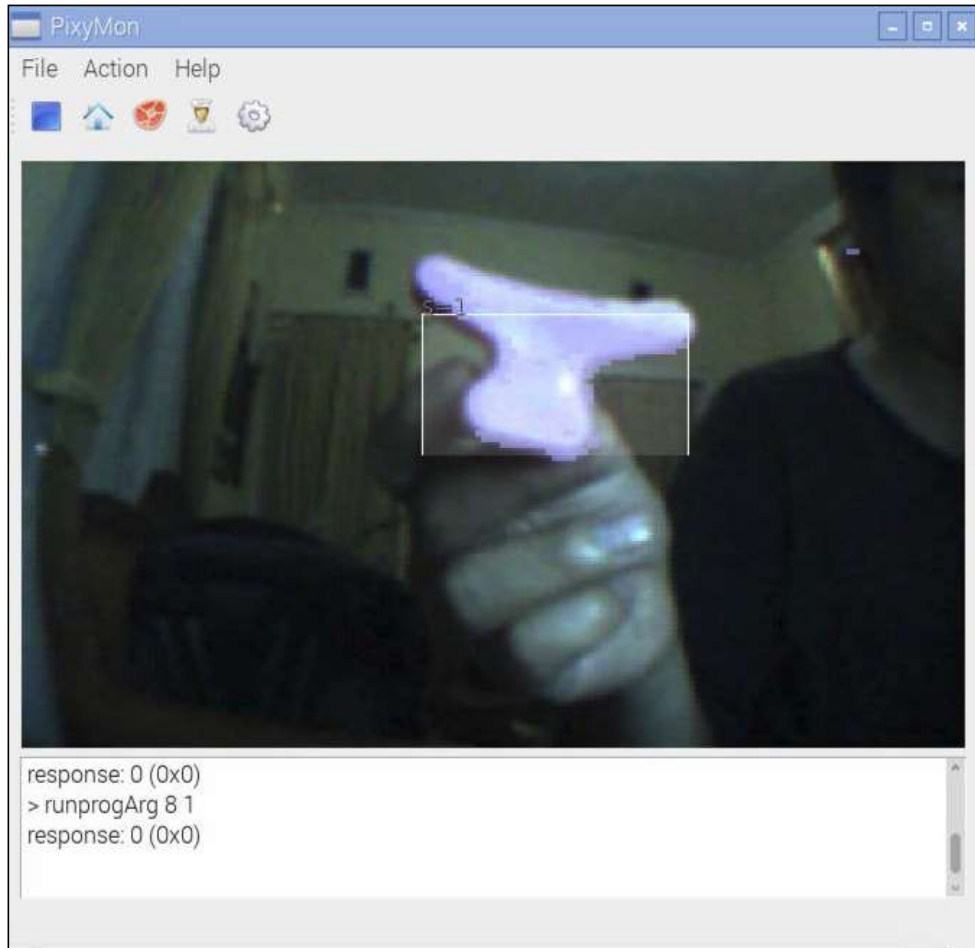
1. Plug Pixy CMUcam5 into the Raspberry Pi board. Open the **PixyMon** application. Show it any object you want to track.



2. Keep your target object on the camera. Then, click menu **Action | Set signature 1** on the PixyMon application.



3. This makes PixyMon freeze the image so you can set a region of your target object using a mouse:

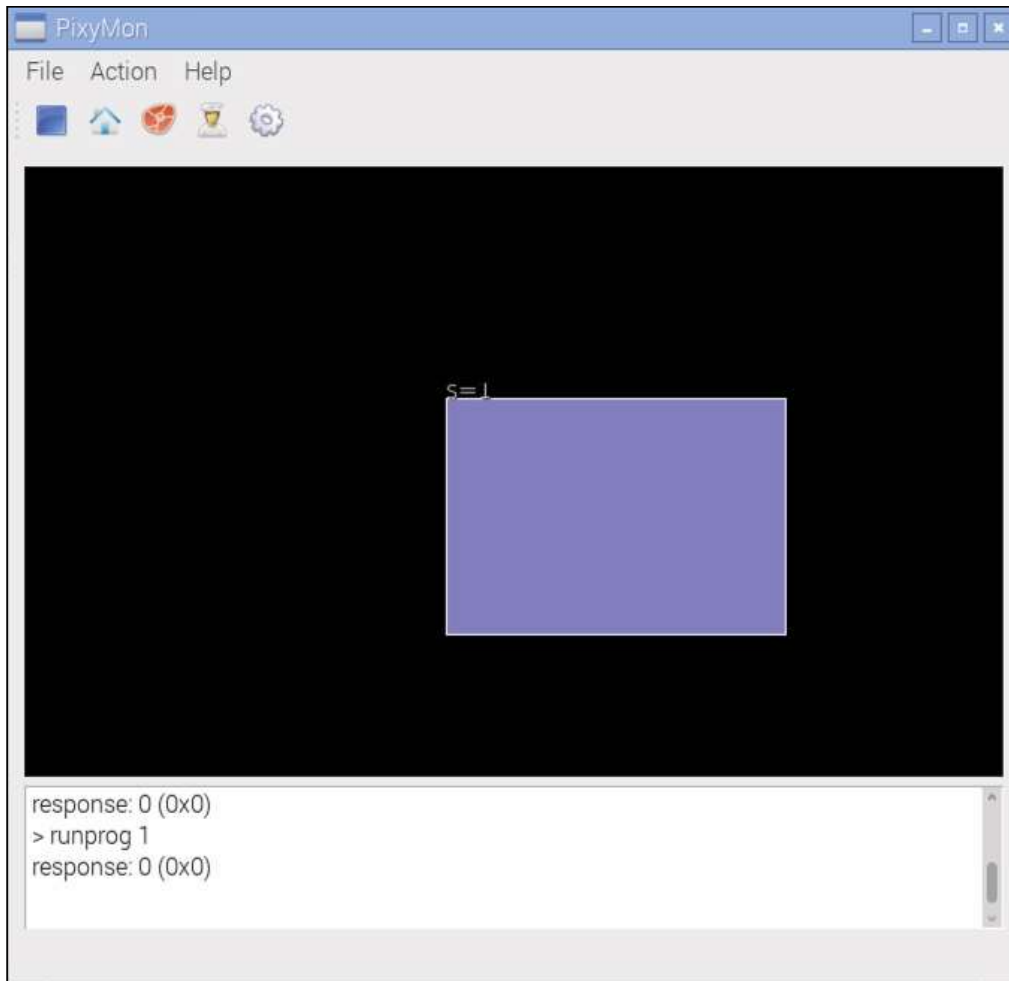


4. After that, PixyMon saves the signature data. Then, the application will track your object. Move your target object.

Tracking an object with a Pan/Tilt module

If you have a Pan/Tilt module already attached to Pixy CMUcam5, you can play a demo to track the object through Pan/Tilt.

1. Using your registered signature, you can activate **Pan/Tilt** by clicking menu **Action | Run Pan/Tilt demo** on the PixyMon application.



2. Try to move your target object. The Pan/Tilt module will move to where the target object is located.

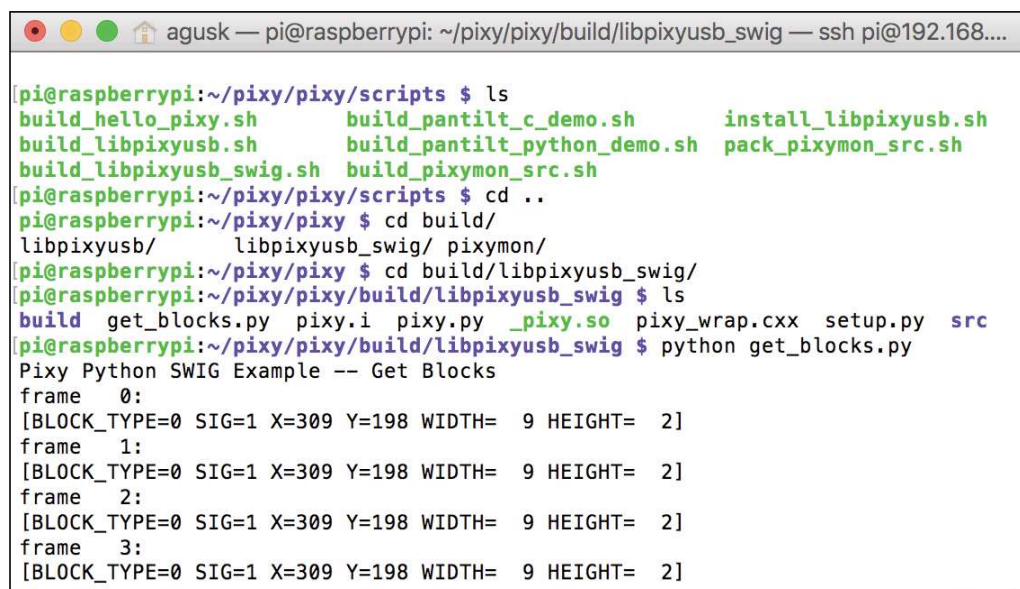
Running the Python application

Using the same registered signature, we can get a signature position. We can use a sample program written in Python.

You can find `get_blocks.py` file in the folder `<pixy_codes>/build/libpixyusb_swig/`. After that, you can add this file:

```
$ python get_blocks.py
```

The program will acquire the position of the signature if it's found. You can see the program output in the following screenshot:



```

agusk — pi@raspberrypi: ~/pixy/pixy/build/libpixyusb_swig — ssh pi@192.168...

[pi@raspberrypi:~/pixy/pixy/scripts $ ls
build_hello_pixy.sh      build_pantilt_c_demo.sh      install_libpixyusb.sh
build_libpixyusb.sh     build_pantilt_python_demo.sh  pack_pixymon_src.sh
build_libpixyusb_swig.sh build_pixymon_src.sh
[pi@raspberrypi:~/pixy/pixy/scripts $ cd ..
[pi@raspberrypi:~/pixy/pixy $ cd build/
libpixyusb/             libpixyusb_swig/ pixymon/
[pi@raspberrypi:~/pixy/pixy $ cd build/libpixyusb_swig/
[pi@raspberrypi:~/pixy/pixy/build/libpixyusb_swig $ ls
build get_blocks.py pixy.i pixy.py _pixy.so pixy_wrap.cxx setup.py src
[pi@raspberrypi:~/pixy/pixy/build/libpixyusb_swig $ python get_blocks.py
Pixy Python SWIG Example -- Get Blocks
frame 0:
[BLOCK_TYPE=0 SIG=1 X=309 Y=198 WIDTH= 9 HEIGHT= 2]
frame 1:
[BLOCK_TYPE=0 SIG=1 X=309 Y=198 WIDTH= 9 HEIGHT= 2]
frame 2:
[BLOCK_TYPE=0 SIG=1 X=309 Y=198 WIDTH= 9 HEIGHT= 2]
frame 3:
[BLOCK_TYPE=0 SIG=1 X=309 Y=198 WIDTH= 9 HEIGHT= 2]

```

What's next?

You can modify the program to control Pixy CMUcam5 module with the Raspberry Pi board.

Clearing all signatures

If you have finished all experiments and don't want to use signature data again, you can clear them by clicking menu **Action | Clear all signatures** on PixyMon application.

Summary

We have learned some basic machine vision using OpenCV. We also explored Python to access OpenCV and then practiced with them.

As the last topic, we deployed machine vision on a Raspberry Pi board to build face detection and track an object.

In the next chapter, we will learn how to build an autonomous car using machine learning.

References

The following is a list of recommended books where you can learn more about the topics in this chapter.

1. Richard Szeliski. *Computer Vision: Algorithms and Applications*, Springer. 2011.
2. P. Viola and M. Jones, *Rapid object detection using a boosted cascade of simple features*, *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on, 2001*, pp. I-511-I-518 vol. 1.
3. OpenCV library, <http://opencv.org>.