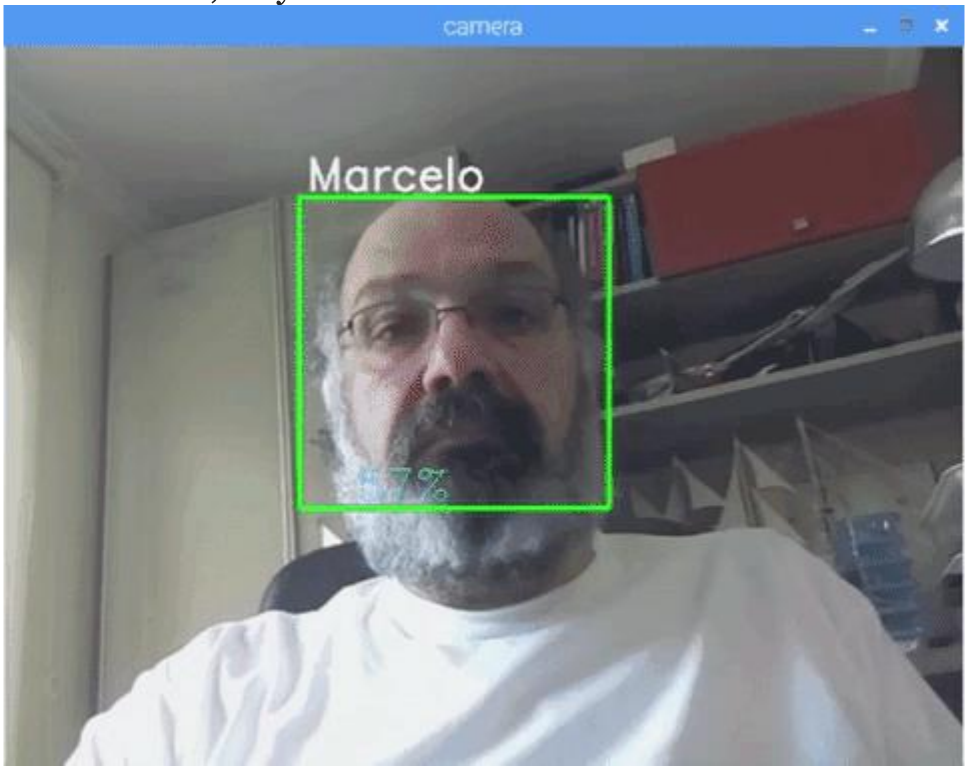
Real-Time Face Recognition: An End-To-End Project

Learn step by step, how to use a PiCam to recognize faces in real-time.

1. Introduction

On my tutorial exploring OpenCV, we learned AUTOMATIC VISION OBJECT TRACKING. Now we will use our PiCam to recognize faces in real-time, as you can see below:



This project was done with this fantastic “Open Source Computer Vision Library”, the OpenCV. On this tutorial, we will be focusing on Raspberry Pi (so, Raspbian as OS) and Python, but I also tested the code on my Mac and it also works fine.

"To run it on a Mac, there is a couple of changes that should be made on code. Do not worry, I will comment about it"

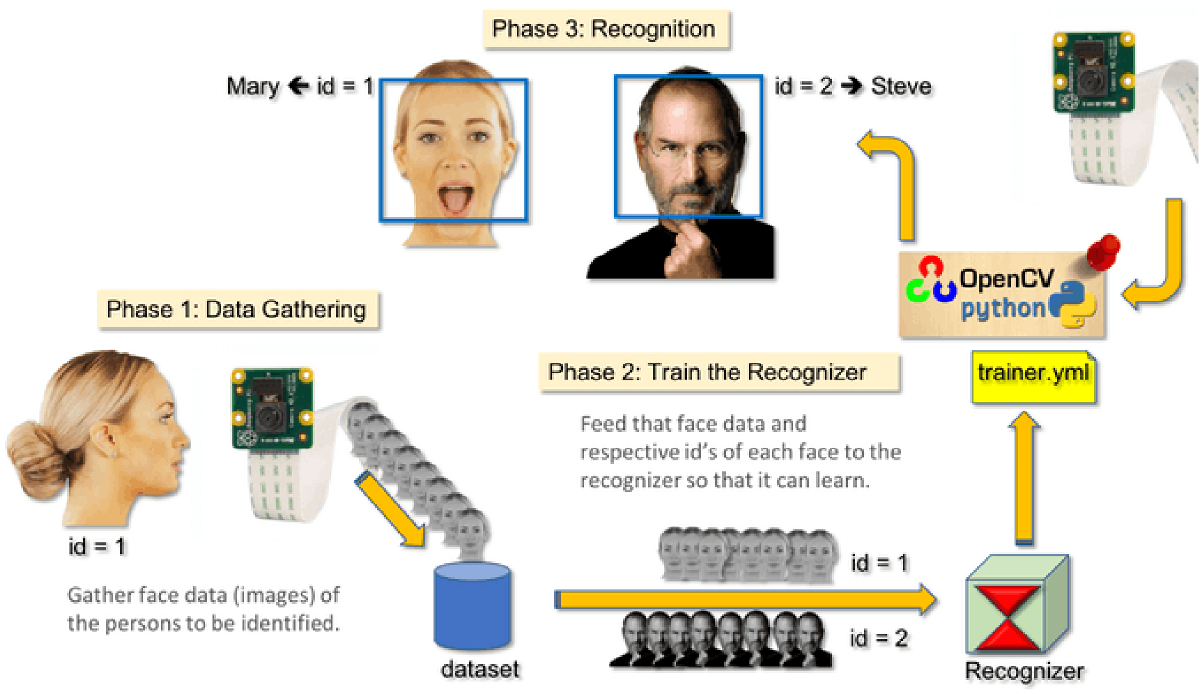
OpenCV was designed for computational efficiency and with a strong focus on real-time applications. So, it's perfect for real-time face recognition using a camera.

The 3 Phases

To create a complete project on Face Recognition, we must work on 3 very distinct phases:

- Face Detection and Data Gathering
- Train the Recognizer
- Face Recognition

The below block diagram resumes those phases:


Phase 3: Recognition
Mary ← id = 1
id = 2 → Steve
OpenCV
python
Phase 1: Data Gathering
Phase 2: Train the Recognizer
trainer.yml
Feed that face data and respective id's of each face to the recognizer so that it can learn.
id = 1
Gather face data (images) of the persons to be identified.
dataset
id = 1
id = 2
Recognizer

2. Installing OpenCV 3 Package

I am using a Raspberry Pi V3 updated to the last version of Raspbian (Stretch), so the best way to have OpenCV installed, is to follow the excellent tutorial developed by Adrian Rosebrock: Raspbian Stretch: Install OpenCV 3 + Python on your Raspberry Pi.

I tried several different guides to install OpenCV on my Pi. Adrian's tutorial is the best. I advise you to do the same, following his guideline step-by-step.

Once you finished Adrian's tutorial, you should have an OpenCV virtual environment ready to run our experiments on your Pi.

Let's go to our virtual environment and confirm that OpenCV 3 is correctly installed.

Adrian recommends run the command "source" each time you open up a new terminal to ensure your system variables have been set up correctly.

```
source ~/.profile
```

Next, let's enter on our virtual environment:

```
workon cv
```

If you see the text (cv) preceding your prompt, then you are in the *cv virtual*environment:

```
(cv) pi@raspberry:~$
```

Adrian calls the attention that the ***cv Python virtual environment*** is entirely independent and sequestered from the default Python version included in the download of Raspbian Stretch. So, any Python packages in the global site-packages directory will not be available to the cv virtual environment. Similarly, any Python packages installed in site-packages of cv will not be available to the global install of Python.

Now, enter in your Python interpreter:

```
python
```

and confirm that you are running the 3.5 (or above) version.

Inside the interpreter (the ">>>" will appear), import the OpenCV library:

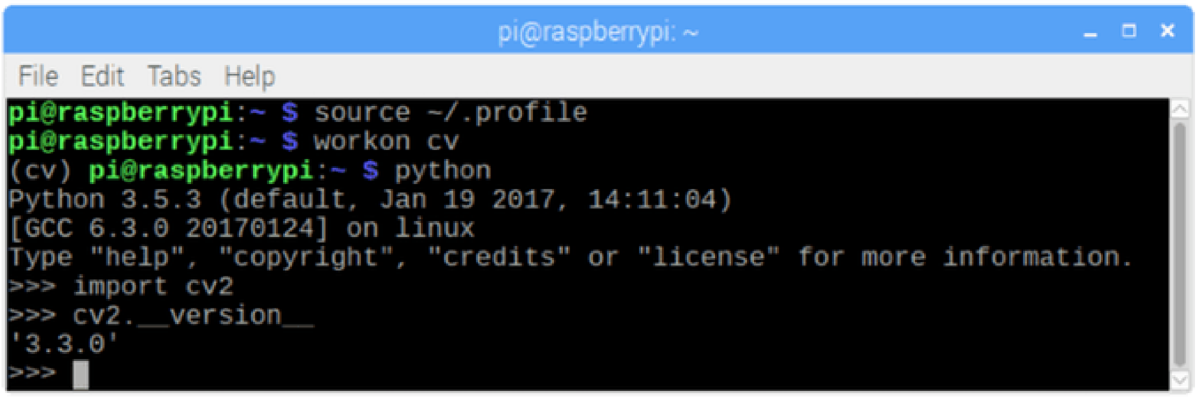
```
import cv2
```

If no error messages appear, the OpenCV is correctly installed ON YOUR PYTHON VIRTUAL ENVIRONMENT.

You can also check the OpenCV version installed:

```
cv2.__version__
```

The 3.3.0 should appear (or a superior version that can be released in future).



```
pi@raspberrypi:~ $ source ~/.profile
pi@raspberrypi:~ $ workon cv
(cv) pi@raspberrypi:~ $ python
Python 3.5.3 (default, Jan 19 2017, 14:11:04)
[GCC 6.3.0 20170124] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import cv2
>>> cv2.__version__
'3.3.0'
>>> 
```

The above Terminal PrintScreen shows the previous steps.

3. Testing Your Camera

Once you have OpenCV installed in your RPi let’s test to confirm that your camera is working properly.

I am assuming that you have a PiCam already installed and enabled on your Raspberry Pi.

Enter the below Python code on your IDE:

```
import numpy as np
import cv2cap = cv2.VideoCapture(0)
cap.set(3,640) # set Width
cap.set(4,480) # set Heightwhile(True):
    ret, frame = cap.read()
    frame = cv2.flip(frame, -1) # Flip camera vertically
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    cv2.imshow('frame', frame)
    cv2.imshow('gray', gray)

    k = cv2.waitKey(30) & 0xff
    if k == 27: # press 'ESC' to quit
        breakcap.release()
cv2.destroyAllWindows()
```

The above code will capture the video stream that will be generated by your PiCam, displaying both, in BGR color and Gray mode.

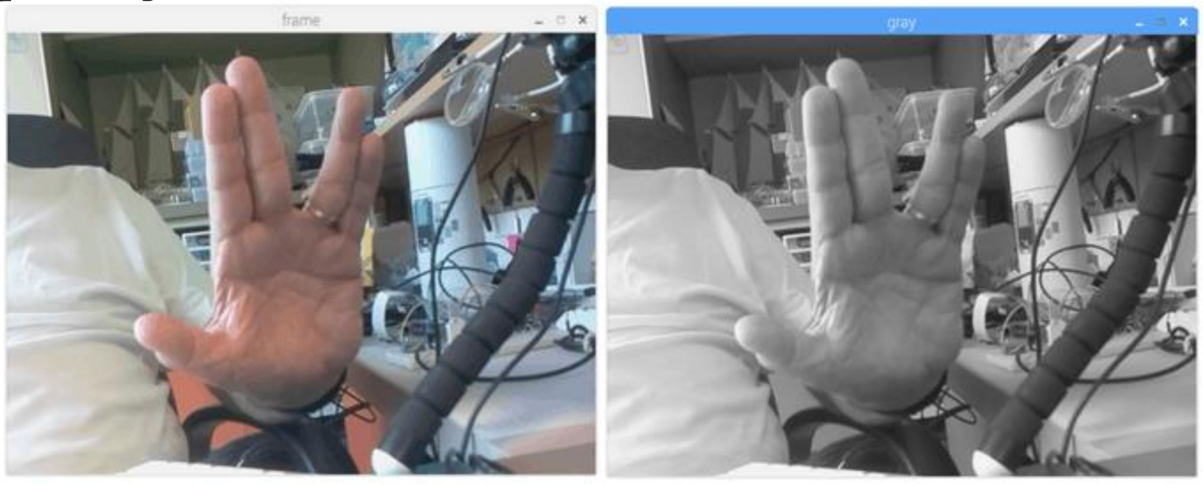
Note that I rotated my camera vertically due the way it is assembled. If it is not your case, comment or delete the “flip” command line.

You can alternatively download the code from my GitHub: simpleCamTest.py

To execute the script, enter the command:

```
python simpleCamTest.py
```

To finish the program, you must press the key [ESC] on your keyboard. Click your mouse on the video window, before pressing [ESC].



The above picture shows the result.

Some people found issues when trying to open the camera and got "Assertion failed" error messages. That could happen if the camera was not enabled during OpenCv installation and so, camera drivers did not install correctly. To correct, use the command:

```
sudo modprobe bcm2835-v4l2
```

You can also add bcm2835-v4l2 to the last line of the /etc/modules file so the driver loads on boot.

To know more about OpenCV, you can follow the tutorial: loading -video-python-opencv-tutorial

4. Face Detection

The most basic task on Face Recognition is of course, “Face Detecting”. Before anything, you must “capture” a face (Phase 1) in order to recognize it, when compared with a new face captured on future (Phase 3).

The most common way to detect a face (or any objects), is using the “Haar Cascade classifier”

Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones in their paper, “Rapid Object Detection using a Boosted Cascade of Simple Features” in 2001. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

Here we will work with face detection. Initially, the algorithm needs a lot of positive images (images of faces) and negative images (images without faces) to train the classifier. Then we need to extract features from it. The good news is that OpenCV comes with a trainer as well as a detector. If you want to train your own classifier for any object like car, planes etc. you can use OpenCV to create one. Its full details are given here: Cascade Classifier Training.

If you do not want to create your own classifier, OpenCV already contains many pre-trained classifiers for face, eyes, smile, etc. Those XML files can be download from haarcascades directory.

Enough theory, let’s create a face detector with OpenCV!

Download the file: faceDetection.py from my GitHub.

```
import numpy as np
import cv2faceCascade = 
cv2.CascadeClassifier('Cascades/haarcascade_frontalface_default.
xml')
cap = cv2.VideoCapture(0)
cap.set(3,640) # set Width
cap.set(4,480) # set Heightwhile True:
    ret, img = cap.read()
    img = cv2.flip(img, -1)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = faceCascade.detectMultiScale(
        gray,
        scaleFactor=1.2,
        minNeighbors=5,
        minSize=(20, 20)
    )
    for (x,y,w,h) in faces:
        cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
        roi_gray = gray[y:y+h, x:x+w]
        roi_color = img[y:y+h, x:x+w]
    cv2.imshow('video',img)
    k = cv2.waitKey(30) & 0xff
    if k == 27: # press 'ESC' to quit
        breakcap.release()
cv2.destroyAllWindows()
```

Believe it or not, the above few lines of code are all you need to detect a face, using Python and OpenCV.

When you compare with the last code used to test the camera, you will realize that few parts were added to it. Note the line below:

```
faceCascade = 
cv2.CascadeClassifier('Cascades/haarcascade_frontalface_default.
xml')
```

This is the line that loads the “classifier” (that must be in a directory named “Cascades/”, under your project directory).

Then, we will set our camera and inside the loop, load our input video in grayscale mode (same we saw before).

Now we must call our classifier function, passing it some very important parameters, as scale factor, number of neighbors and minimum size of the detected face.

```
faces = faceCascade.detectMultiScale(
        gray,
        scaleFactor=1.2,
        minNeighbors=5,
        minSize=(20, 20)
        )
```

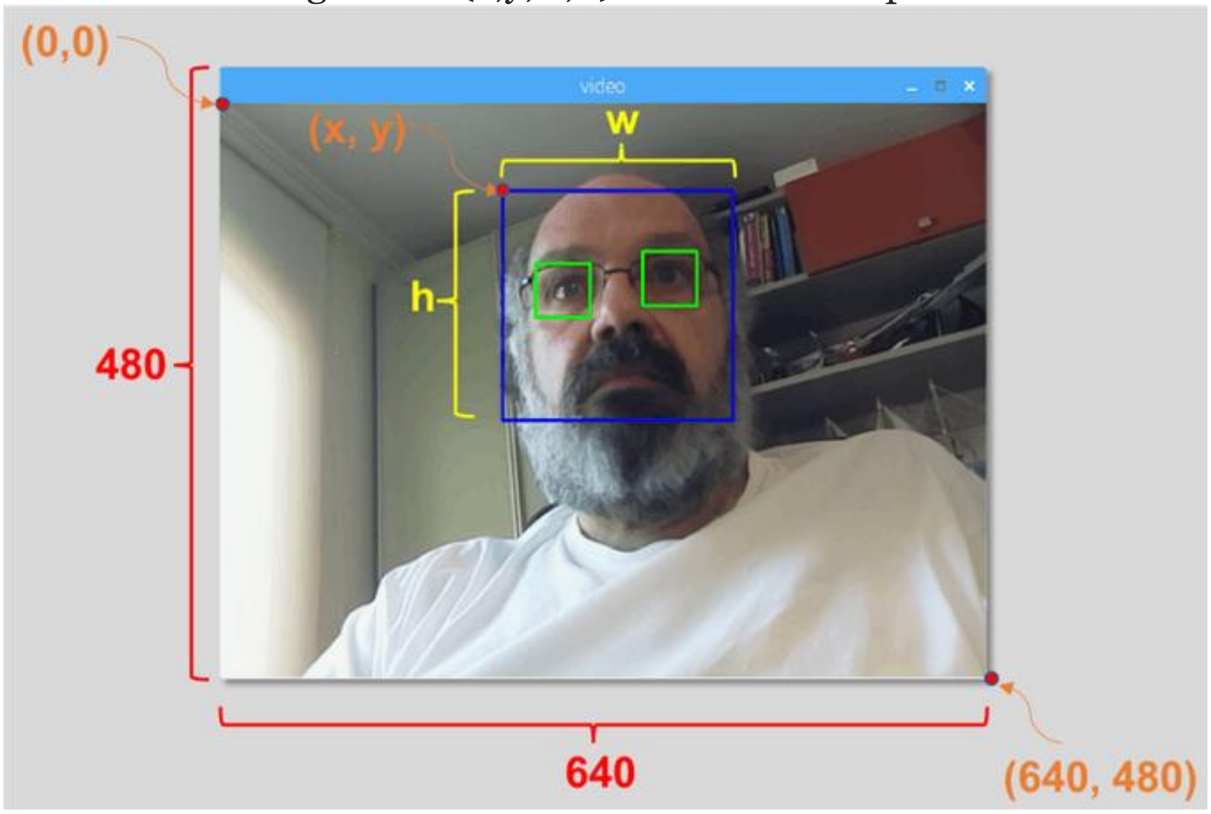
Where,

- **gray** is the input grayscale image.
- **scaleFactor** is the parameter specifying how much the image size is reduced at each image scale. It is used to create the scale pyramid.
- **minNeighbors** is a parameter specifying how many neighbors each candidate rectangle should have, to retain it. A higher number gives lower false positives.
- **minSize** is the minimum rectangle size to be considered a face.

The function will detect faces on the image. Next, we must “mark” the faces in the image, using, for example, a blue rectangle. This is done with this portion of the code:

```
for (x,y,w,h) in faces:
    cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
    roi_gray = gray[y:y+h, x:x+w]
    roi_color = img[y:y+h, x:x+w]
```

If faces are found, it returns the positions of detected faces as a rectangle with the left up corner (x,y) and having "w" as its Width and "h" as its Height ==> (x,y,w,h). Please see the picture.

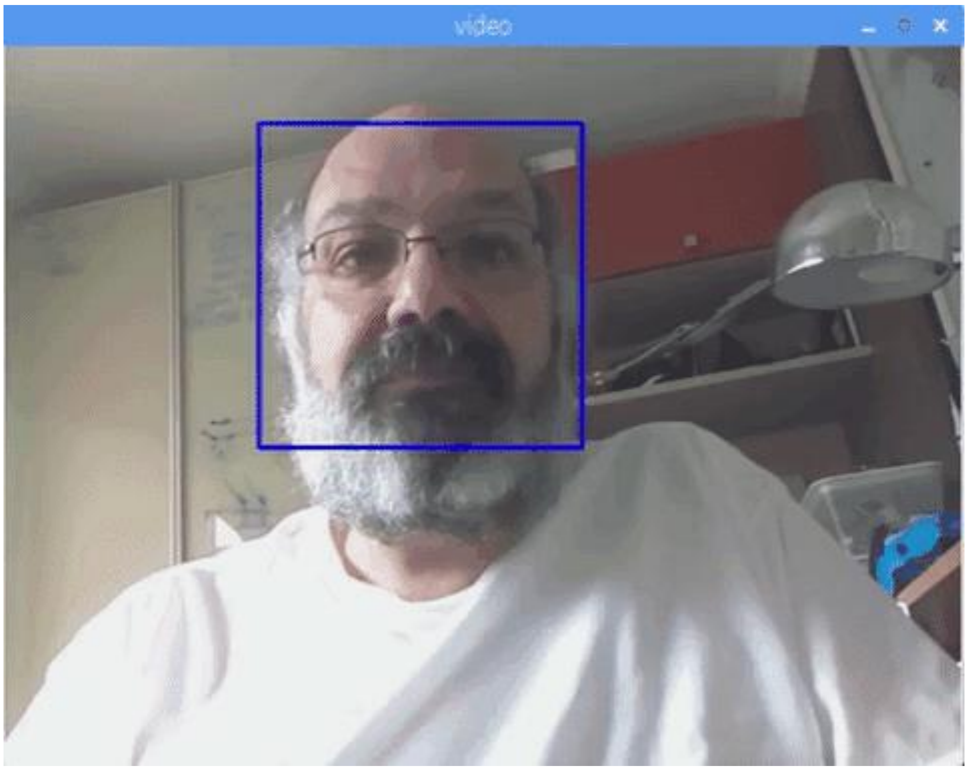


Once we get these locations, we can create an "ROI" (drawn rectangle) for the face and present the result with *imshow()* function.

Run the above python Script on your python environment, using the Rpi Terminal:

```
python faceDetection.py
```

The result:



You can also include classifiers for “eyes detection” or even “smile detection”. On those cases, you will include the classifier function and rectangle draw inside the face loop, because would be no sense to detect an eye or a smile outside of a face.

Note that on a Pi, having several classifiers at same code will slow the processing, once this method of detection (HaarCascades) uses a great amount of computational power. On a desktop, it is easer to run it.

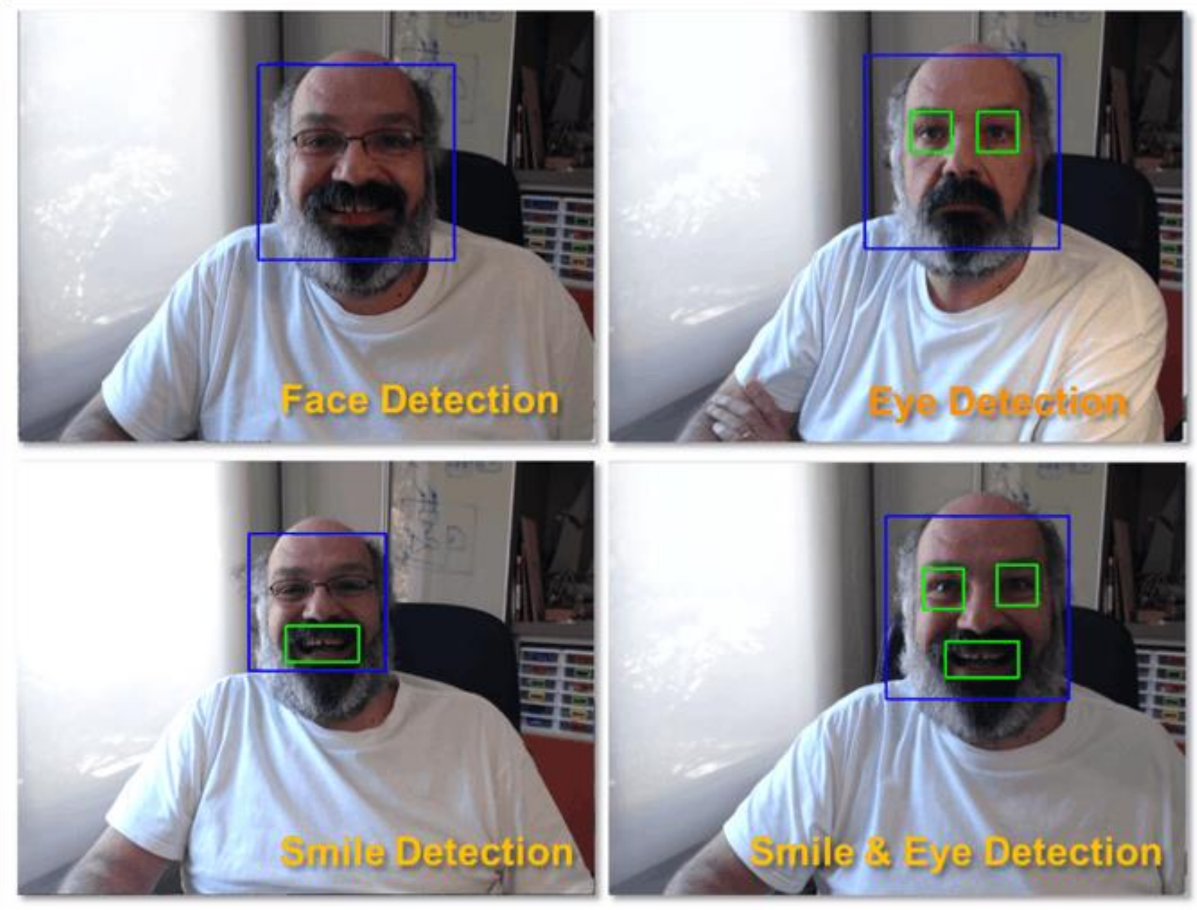
Examples

On my GitHub you will find other examples:

- faceEyeDetection.py
- faceSmileDetection.py

- faceSmileEyeDetection.py

And in the picture, you can see the result.



You can also follow the below tutorial to better understand Face Detection:

Haar Cascade Object Detection Face & Eye OpenCV Python Tutorial

5. Data Gathering

First of all, I must thank Ramiz Raja for his great work on Face Recognition on photos:

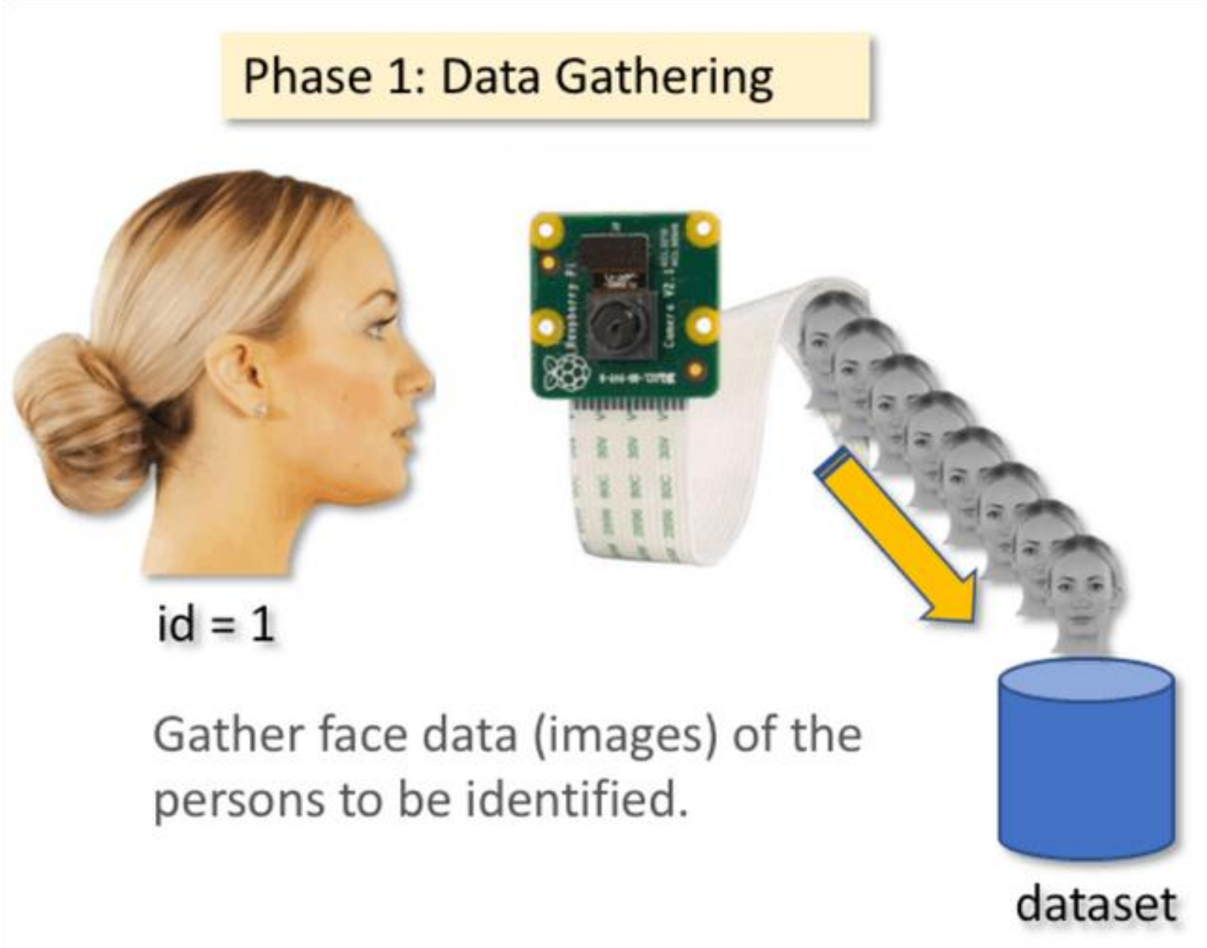
[FACE RECOGNITION USING OPENCV AND PYTHON: A BEGINNER'S GUIDE]

and also Anirban Kar, that developed a very comprehensive tutorial using video:

[FACE RECOGNITION — 3 parts]

I really recommend that you take a look at both tutorials.

Saying that, let's start the first phase of our project. What we will do here, is starting from last step (Face Detecting), we will simply create a dataset, where we will store for each id, a group of photos in gray with the portion that was used for face detecting.



First, create a directory where you develop your project, for example, FacialRecognitionProject:

```
mkdir FacialRecognitionProject
```

In this directory, besides the 3 python scripts that we will create for our project, we must have saved on it the Facial Classifier. You can download it from my GitHub: haarcascade_frontalface_default.xml

Next, create a subdirectory where we will store our facial samples and name it “dataset”:

```
mkdir dataset
```

And download the code from my GitHub: 01_face_dataset.py

```
import cv2
import oscam = cv2.VideoCapture(0)
cam.set(3, 640) # set video width
cam.set(4, 480) # set video height
face_detector = 
cv2.CascadeClassifier('haarcascade_frontalface_default.xml')#
For each person, enter one numeric face id
face_id = input('\n enter user id end press <return> ==>  ')
print("\n [INFO] Initializing face capture. Look the camera and
wait ...")# Initialize individual sampling face count
count = 0
while(True):
    ret, img = cam.read()
    img = cv2.flip(img, -1) # flip video image vertically
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_detector.detectMultiScale(gray, 1.3, 5)
    for (x,y,w,h) in faces:
        cv2.rectangle(img, (x,y), (x+w,y+h), (255,0,0), 2)
        count += 1
        # Save the captured image into the datasets folder
        cv2.imwrite("dataset/User." + str(face_id) + '.' +
                    str(count) + ".jpg", gray[y:y+h,x:x+w])
        cv2.imshow('image', img)
    k = cv2.waitKey(100) & 0xff # Press 'ESC' for exiting video
    if k == 27:
        break
    elif count >= 30: # Take 30 face sample and stop video
         break# Do a bit of cleanup
print("\n [INFO] Exiting Program and cleanup stuff")
```

```
cam.release()
cv2.destroyAllWindows()
```

The code is very similar to the code that we saw for face detection. What we added, was an “input command” to capture a user id, that should be an integer number (1, 2, 3, etc)

```
face_id = input('\n enter user id end press  ==>  ')
```

And for each one of the captured frames, we should save it as a file on a “dataset” directory:

```
cv2.imwrite("dataset/User." + str(face_id) + '.' + str(count) +
".jpg", gray[y:y+h,x:x+w])
```

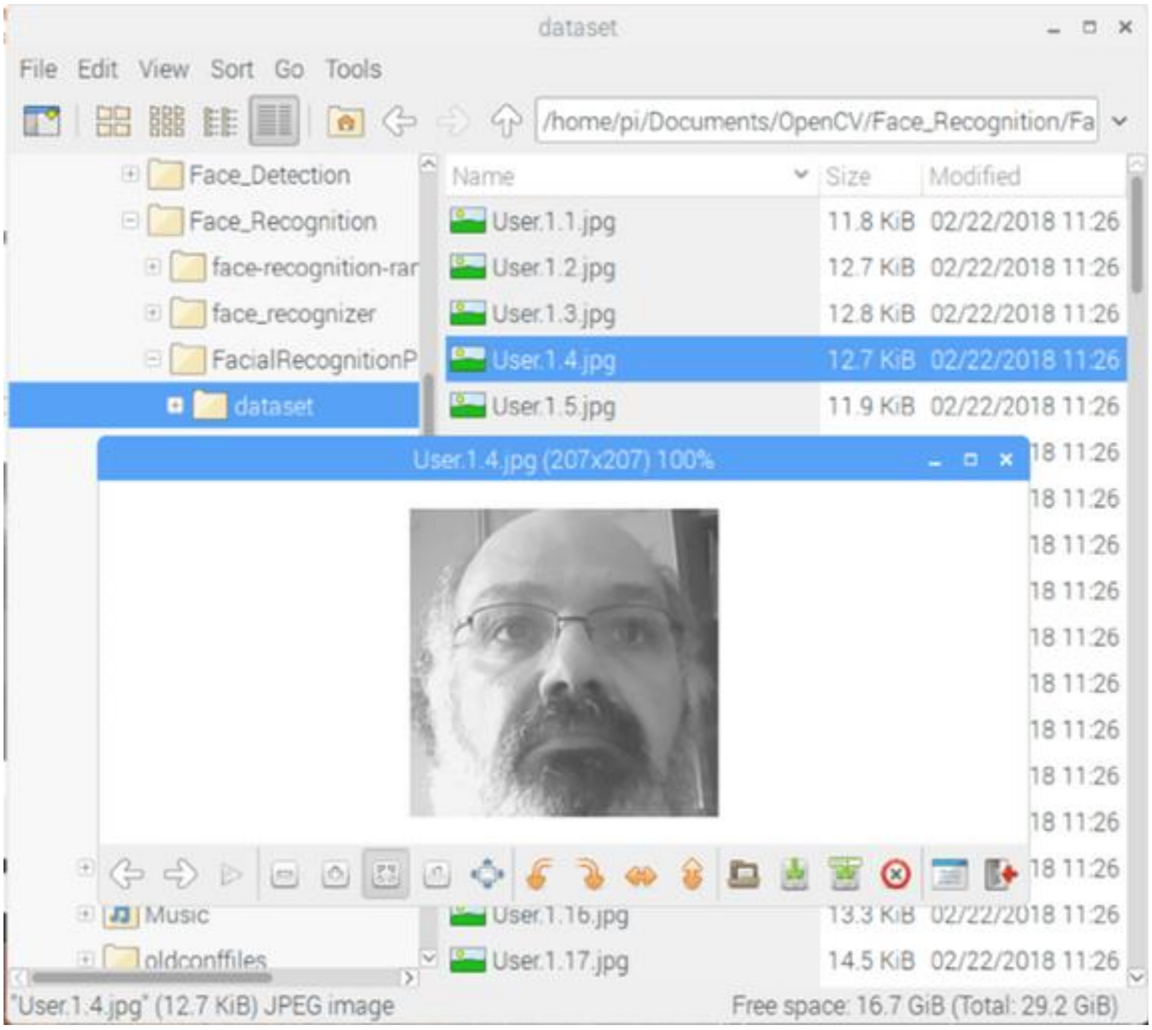
Note that for saving the above file, you must have imported the library “os”. Each file’s name will follow the structure:

```
User.face_id.count.jpg
```

For example, for a user with a face_id = 1, the 4th sample file on dataset/ directory will be something like:

```
User.1.4.jpg
```

as shown in the photo from my Pi:



Name	Size	Modified
User.1.1.jpg	11.8 KiB	02/22/2018 11:26
User.1.2.jpg	12.7 KiB	02/22/2018 11:26
User.1.3.jpg	12.8 KiB	02/22/2018 11:26
User.1.4.jpg	12.7 KiB	02/22/2018 11:26
User.1.5.jpg	11.9 KiB	02/22/2018 11:26
User.1.16.jpg	13.3 KiB	02/22/2018 11:26
User.1.17.jpg	14.5 KiB	02/22/2018 11:26

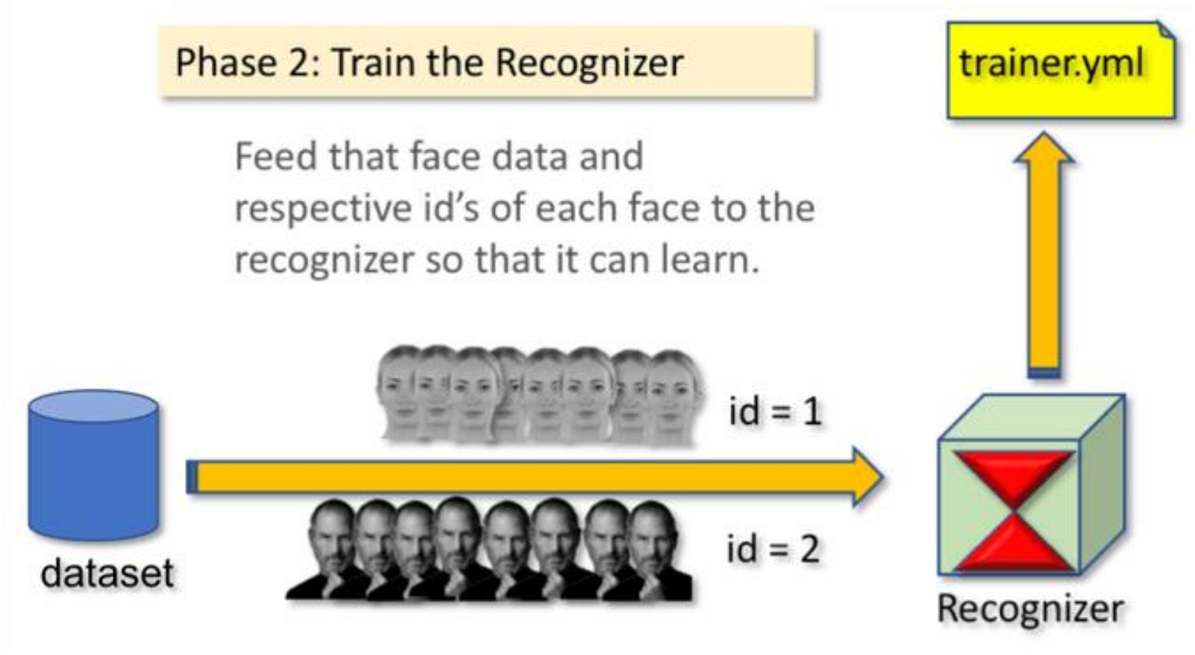
"User.1.4.jpg" (12.7 KiB) JPEG image
Free space: 16.7 GiB (Total: 29.2 GiB)

On my code, I am capturing 30 samples from each id. You can change it on the last "elif". The number of samples is used to break the loop where the face samples are captured.

Run the Python script and capture a few Ids. You must run the script each time that you want to aggregate a new user (or to change the photos for one that already exists).

6. Trainer

On this second phase, we must take all user data from our dataset and "trainer" the OpenCV Recognizer. This is done directly by a specific OpenCV function. The result will be a .yml file that will be saved on a "trainer/" directory.


Phase 2: Train the Recognizer
trainer.yml
Feed that face data and respective id's of each face to the recognizer so that it can learn.
id = 1
id = 2
dataset
Recognizer

So, let's start creating a subdirectory where we will store the trained data:

```
mkdir trainer
```

Download from my GitHub the second python script: [02_face_training.py]

```
import cv2
import numpy as np
from PIL import Image
import os# Path for face image database
path = 'dataset'
recognizer = cv2.face.LBPHFaceRecognizer_create()
detector =
cv2.CascadeClassifier("haarcascade_frontalface_default.xml");#
function to get the images and label data
def getImagesAndLabels(path):
    imagePaths = [os.path.join(path,f) for f in
os.listdir(path)]
    faceSamples=[]
    ids = []
    for imagePath in imagePaths:
        PIL_img = Image.open(imagePath).convert('L') # grayscale
        img_numpy = np.array(PIL_img,'uint8')
        id = int(os.path.split(imagePath)[-1].split(".")[1])
        faces = detector.detectMultiScale(img_numpy)
        for (x,y,w,h) in faces:
            faceSamples.append(img_numpy[y:y+h,x:x+w])
            ids.append(id)
    return faceSamples,idsprint ("\n [INFO] Training faces. It
will take a few seconds. Wait ...")faces,ids =
getImagesAndLabels(path)
recognizer.train(faces, np.array(ids))# Save the model into
trainer/trainer.yml
recognizer.write('trainer/trainer.yml') # Print the numer of
faces trained and end program
print("\n [INFO] {0} faces trained. Exiting
Program".format(len(np.unique(ids))))
```

```
# recognizer.save() worked on Mac, but not on Pi
```

Confirm if you have the PIL library installed on your Rpi. If not, run the below command in Terminal:

```
pip install pillow
```

We will use as a recognizer, the LBPH (LOCAL BINARY PATTERNS HISTOGRAMS) Face Recognizer, included on OpenCV package. We do this in the following line:

```
recognizer = cv2.face.LBPHFaceRecognizer_create()
```

The function “getImagesAndLabels (path)”, will take all photos on directory: “dataset/”, returning 2 arrays: “Ids” and “faces”. With those arrays as input, we will “train our recognizer”:

```
recognizer.train(faces, ids)
```

As a result, a file named “trainer.yml” will be saved in the trainer directory that was previously created by us.

That’s it! I included the last print statement where I displayed for confirmation, the number of User’s faces we have trained.

Every time that you perform Phase 1, Phase 2 must also be run.

7. Recognizer

Now, we reached the final phase of our project. Here, we will capture a fresh face on our camera and if this person had his face captured and trained before, our recognizer will make a "prediction" returning its id and an index, shown how confident the recognizer is with this match.

Let's download the 3rd phase python script from my GitHub: 03_face_recognition.py.

```
import cv2
import numpy as np
import os recognizer = cv2.face.LBPHFaceRecognizer_create()
recognizer.read('trainer/trainer.yml')
cascadePath = "haarcascade_frontalface_default.xml"
faceCascade = cv2.CascadeClassifier(cascadePath);
font = cv2.FONT_HERSHEY_SIMPLEX#iniciate id counter
id = 0# names related to ids: example ==> Marcelo: id=1,  etc
names = ['None', 'Marcelo', 'Paula', 'Ilza', 'Z', 'W'] #
Initialize and start realtime video capture
cam = cv2.VideoCapture(0)
cam.set(3, 640) # set video widht
cam.set(4, 480) # set video height# Define min window size to be
recognized as a face
minW = 0.1*cam.get(3)
minH = 0.1*cam.get(4)while True:
    ret, img =cam.read()
    img = cv2.flip(img, -1) # Flip vertically
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

    faces = faceCascade.detectMultiScale(
        gray,
        scaleFactor = 1.2,
        minNeighbors = 5,
        minSize = (int(minW), int(minH)),
       )
    for(x,y,w,h) in faces:
        cv2.rectangle(img, (x,y), (x+w,y+h), (0,255,0), 2)
        id, confidence = recognizer.predict(gray[y:y+h,x:x+w])
```

```
                # If confidence is less them 100 ==> "0" :
perfect match
        if (confidence < 100):
            id = names[id]
            confidence = "  {0}%".format(round(100 -
confidence))
        else:
            id = "unknown"
            confidence = "  {0}%".format(round(100 -
confidence))

        cv2.putText(
                    img,
                    str(id),
                    (x+5,y-5),
                    font,
                    1,
                    (255,255,255),
                    2
                   )
        cv2.putText(
                    img,
                    str(confidence),
                    (x+5,y+h-5),
                    font,
                    1,
                    (255,255,0),
                    1
                   )

    cv2.imshow('camera',img)
    k = cv2.waitKey(10) & 0xff # Press 'ESC' for exiting video
    if k == 27:
        break# Do a bit of cleanup
print("\n [INFO] Exiting Program and cleanup stuff")
cam.release()
cv2.destroyAllWindows()
```

We are including here a new array, so we will display “names”, instead of numbered ids:

```
names = ['None', 'Marcelo', 'Paula', 'Ilza', 'Z', 'W']
```

So, for example: Marcelo will the user with id = 1; Paula: id=2, etc.

Next, we will detect a face, same we did before with the haasCascade classifier. Having a detected face we can call the most important function in the above code:

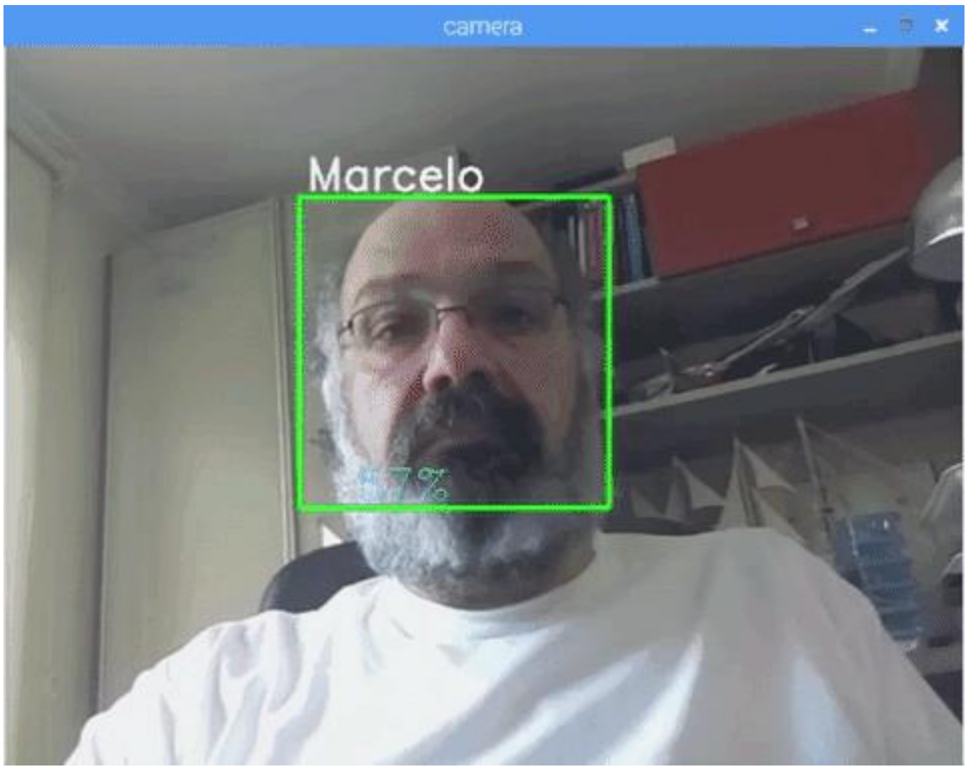
```
id, confidence = recognizer.predict(gray portion of the face)
```

The recognizer.predict (), will take as a parameter a captured portion of the face to be analyzed and will return its probable owner, indicating its id and how much confidence the recognizer is in relation with this match.

Note that the confidence index will return “zero” if it will be cosidered a perfect match

And at last, if the recognizer could predict a face, we put a text over the image with the probable id and how much is the “probability” in % that the match is correct (“probability” = 100 — confidence index). If not, an “unknow” label is put on the face.

Below a gif with the result:



On the picture, I show some tests done with this project, where I also have used photos to verify if the recognizer works.

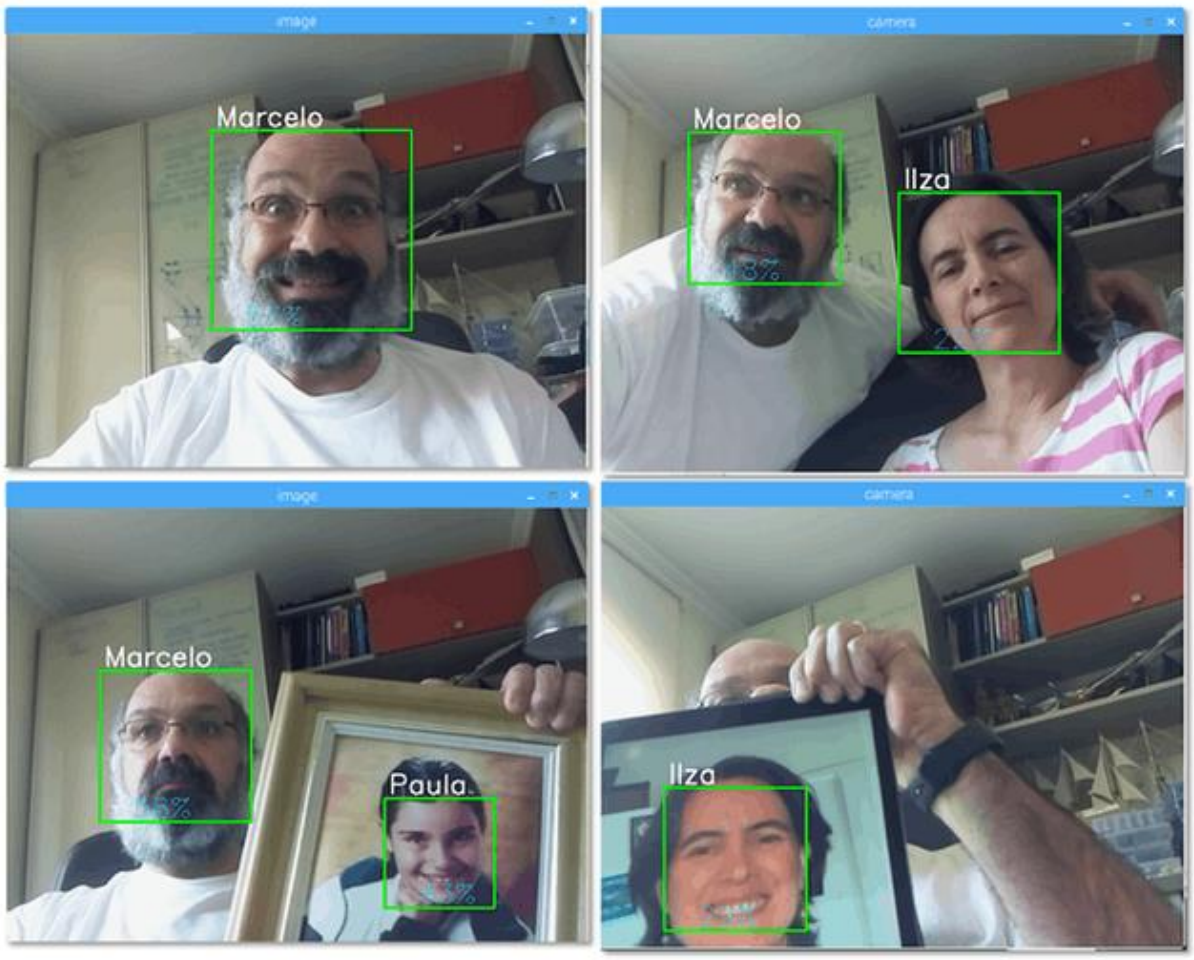


image
Marcelo
camera
Marcelo
Ilza
image
Marcelo
Paula
camera
Ilza