

Facial Recognition Lock Using Raspberry Pi

INDUSTRIAL TRAINING PROJECT REPORT

Submitted in partial fulfilment of the requirements for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

by

Naman Tyagi

Enrollment No: 45115603117



**DEPARTMENT OF INFORMATION TECHNOLOGY
DR.AKHILESH DASS GUPTA INSTITUTE OF TECH & MGMT
(AFFILIATED TO GURU GOBIND SINGH INDRAPRASTHA UNIVERSITY, DELHI)
NEW DELHI – 110053**

CERTIFICATE OF TRAINING



ABSTRACT

Upon completing my training in Machine Learning and Data Science, I was left curious enough to know the applications of such technologies in the real world.

All of us use locks to keep safe something in our lives that is private enough but we have all been careless enough to either lose the key or forget the passcode to the lock. To solve this problem, I turned to Machine Learning to make our face the key for our locks as it is the only feature unique to us and we carry it all the time with us.

To achieve this goal, I had to develop a facial recognition software using the OpenCV library available to us in Python on a Raspberry Pi. The reason why I chose a Raspberry Pi was because it can be connected to various other electronics using its GPIO (General Purpose Input Output) pins and takes power via micro-USB which makes it really portable and power efficient.

The basic idea is to connect an electrical solenoid lock to the RPi and somehow manage to send a signal to the lock to lock and unlock itself upon recognition of a face. The facial recognition software would run on the raspberry pi and take its inputs from the camera module connected to the raspberry pi.

This was my ladder of success towards my project that I chose. Each step, although difficult to solve alone but not impossible, was a step closer to the completion of my DIY (Do It Yourself) project to make all our lives simpler and convenient.

ACKNOWLEDGEMENT

First of all, I would like to thank UdeMy for such an amazing experience at your bedside. Be it any skill you desire, they have just the right courses for you to fulfill that desire. From that, I would like to acknowledge my trainers at UdeMy namely, Kirill Eremenko and Hadelin de Ponteves, for such a fantastic learning opportunity to develop my interest in Data Science and Machine Learning. Their course “Machine Learning A-Z for R and Python” is highly intuitive to learn and understand various topics in Data Science and Machine Learning.

Secondly, I would like to thank the IT department of my college for giving me the opportunity to assess myself based on my training to explore all the ideas in the world and choose any project of any difficulty to enhance my creativity and to put in hard work hours.

Also, I would like to thank my seniors who helped me through all the obstacles that I encountered during the development of this project.

Thank you all.

TABLE OF CONTENTS

Abstract	3
Acknowledgement	4
Table of contents	5-6
List of Abbreviations	7
Chapter 1 : Introduction	8-10
1.1. Raspberry Pi	8
1.2. Components of RPi	8
1.2.1. USB Ports	9
1.2.2. HDMI Port	9
1.2.3. Auxiliary Port	9
1.2.4. Ethernet Jack	
1.2.5. Memory	9
1.2.6. Processor	9
1.2.7. Storage	9
1.2.8. Power	9
1.2.9. GPIO Pins	9
1.2.10. CSI Camera Slot	9
1.3. Ladder of Success	10
1.4. Materials Required	10
Chapter 2 : Software	11-27
2.1. Raspbian Buster	11
2.2. Installing Dependencies	11-12
2.3. Downloading, Installing & Compiling OpenCV	12-14
2.4. Testing the camera	14-15
2.5. Face Detection	15-20
2.5.1. Haar Cascade Classifier	15-18

2.6. Creating Database for Training	21-22
2.7. Creating the Trainer File	23-25
2.8. Recognizer	25-27
Chapter 3 : Hardware	28-38
3.1. GPIO Pins	28-29
3.1.1. Voltages	28
3.1.2. Outputs	28
3.1.3. Inputs	29
3.2. Relay Module	29-31
3.2.1. Normally Open	30
3.2.2. Normally Closed	30-31
3.3. Solenoid Locks	31-32
3.4. Power Supply	32
3.5. Circuit Diagram	32-35
3.6. The Final Code	36-38
Chapter 4 : Difficulties, Bugs and Future Scope	39-40
4.1. Difficulties	39
4.2. Bugs	39
4.3. Future Scope	40
References	41-42

LIST OF ABBREVIATION

RPi	Raspberry Pi
I/O	Input and Output
USB	Universal Serial Bus
HDMI	High Definition Multimedia Interface
RAM	Random Access Memory
GPIO	General Purpose Input and Output
OS	Operating System
CSI	Camera Serial Interface

CHAPTER 1: INTRODUCTION

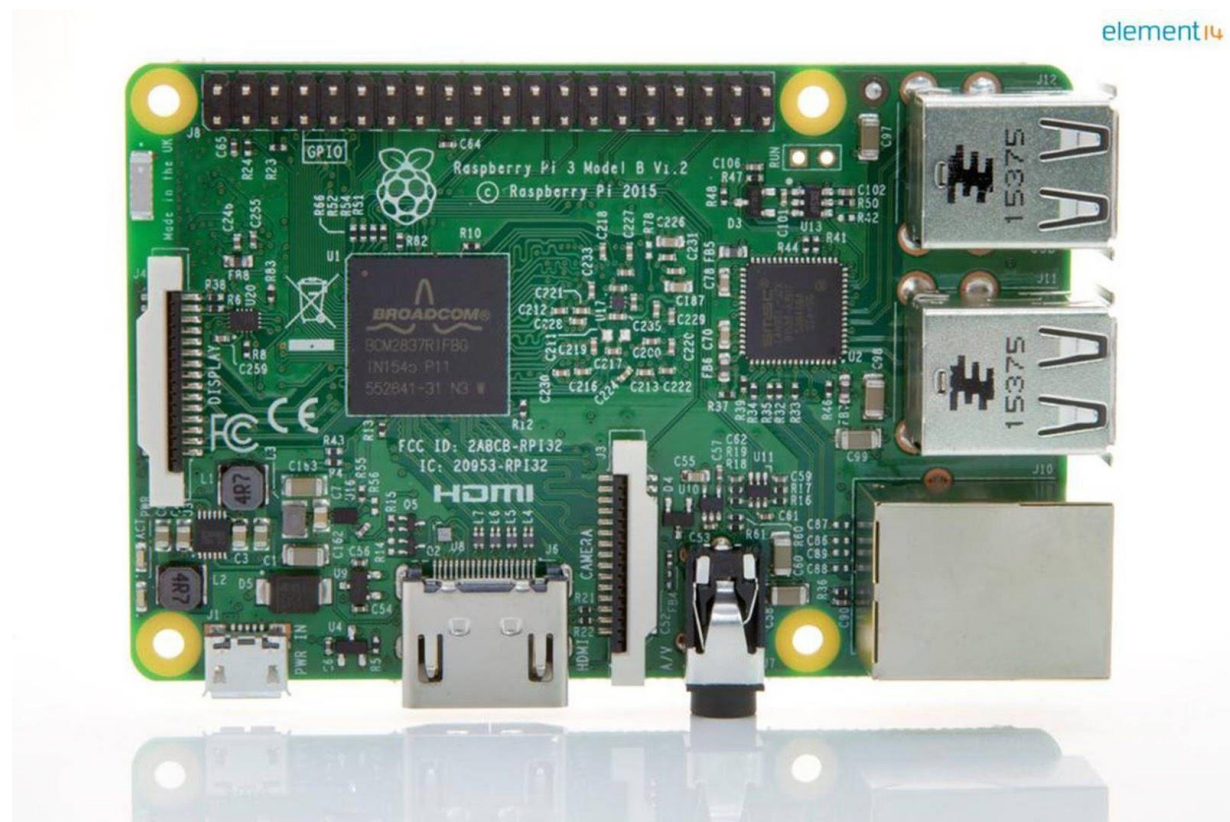
In this chapter I will mention the initial idea of the project. All the knowledge I had before advancing to the more complex stages. It will cover the RPi and its components, the plan of action and the materials required for development.

1.1 Raspberry Pi

The Raspberry Pi is a low cost, credit-card sized computer that plugs into a computer monitor or TV, and uses a standard keyboard and mouse. It is a capable little device that enables people of all ages to explore computing, and to learn how to program in languages like Scratch and Python. It's capable of doing everything you'd expect a desktop computer to do, from browsing the internet and playing high-definition video, to making spreadsheets, word-processing, and playing games.

1.2 Components of Raspberry Pi

The RPi has various components attached to it. There are various I/O to it, a processor, RAM, GPIO pins and so on. Lets talk about them briefly.



1.2.1 USB Ports

The RPi has 4 USB ports gen 2.0 ports.

1.2.2 HDMI port

The RPi has a full sized HDMI port to help hook it to a monitor and work on it just like any other computer.

1.2.3 Auxiliary port

The RPi has a 3.5 mm jack to connect it to various speakers, headphones, mics and so on.

1.2.4 Ethernet Jack

The RPi also comes with an RJ-45 port for wired internet connections.

1.2.5 Memory

The RPi has 1GB LPDDR2 SDRAM.

1.2.6 Processor

The RPi has a Broadcom BCM2837 Processor Quad core A53(ARM v8) 64-bit SoC

1.2.7 Storage

It has a microSD card slot for loading operating system and data storage.

1.2.8 Power

It has a micro USB connector for 5.1V / 2.5A dc.

1.2.9 GPIO Pins

A powerful feature of the RPi is the presence of 40 GPIO pins which gives it vast versatility.

1.2.10 CSI Camera Slot

The CSI camera slot provides an interface between the camera module and the host processor.

1.3 Ladder of Success

Step 4 : After the recognition of the individual on the database, the RPi should send a signal to the solenoid lock to unlock itself.

Step 3 : Create a database of images of the person to be trained on the ML algorithm so as to detect his face uniquely.

Step 2 : Build a software to detect all human faces and draw a box around them so as to detect them

Step 1 : Install the Raspbian OS on the RPi and connect the camera module with the CSI Camera Interface to see if it is working properly

1.4 Materials Required

- A Raspberry Pi
- A Raspberry Pi Camera Module
- A 16 GB SD Card
- An SD Card Reader
- A 12V solenoid door lock
- Jumper Wires
- External Power Supply (Output should be no more than 12V)
- Power Bank
- Relay Module
- Peripherals (Keyboard, Mouse, Monitor)
- USB DC 5V to 12V Step Up Converter
- DC power female jack connector

Chapter 2 : Software

In this chapter we will study about the various software aspects required for the development of this project. We will go through all of it step by step following our ladder of success.

2.1 Raspbian Buster

It is an OS present on the website of raspberry pi. It is free to download. You can either download a disc image file or a zip file. Regardless of what you download, they give you the links to the tools that properly burn/extract the file to the SD Card.

After burning/extracting the image file to the SD Card using the SD card reader, we plug the card into the RPi and wait for it to boot up. While it is booting up, we will get prompts to set the time and location and also about the updates.

Now our OS is installed properly on the RPi and we can commence with the rest of the steps.

2.2 Installing dependencies

This is the initial step to start installing OpenCV on the RPi.

- The first step is to manually update any existing packages using the command :
`sudo apt-get update && sudo apt-get upgrade`
- We then need to install some developer tools, including CMake, which helps us configure the OpenCV build process:
`sudo apt-get install build-essential cmake pkg-config`
- Next, we need to install some image I/O packages that allow us to load various image file formats from disk. Examples of such file formats include JPEG, PNG, TIFF, etc.:
`sudo apt-get install libjpeg-dev libtiff5-dev libjasper-dev libpng12-dev`
- Just as we need image I/O packages, we also need video I/O packages. These libraries allow us to read various video file formats from disk as well as work directly with video streams:
`sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev`
`sudo apt-get install libxvidcore-dev libx264-dev`

- The OpenCV library comes with a sub-module named highgui which is used to display images to our screen and build basic GUIs. In order to compile the highgui module, we need to install the GTK development library:
`sudo apt-get install libgtk2.0-dev libgtk-3-dev`
- Many operations inside of OpenCV (namely matrix operations) can be optimized further by installing a few extra dependencies:
`sudo apt-get install libatlas-base-dev gfortran`
- Lastly, let's install both the Python 2.7 and Python 3 header files so we can compile OpenCV with Python bindings.

`sudo apt-get install python2.7-dev python3-dev`

2.3 Downloading, Installing and compiling OpenCV

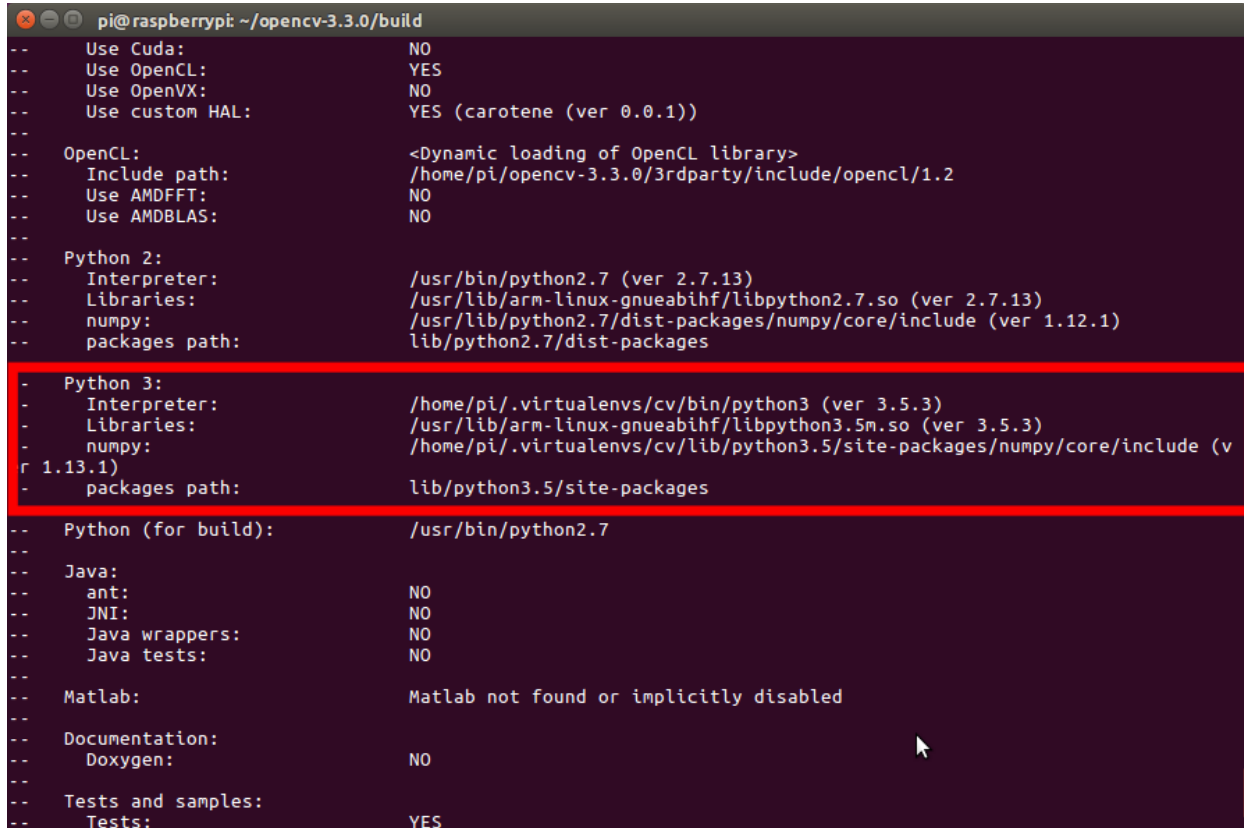
- Now that we have our dependencies installed, let's grab the 3.3.0 archive of OpenCV from the official OpenCV repository.
`Cd ~`
`wget -O opencv.zip https://github.com/Itseez/opencv/archive/3.3.0.zip`
`unzip opencv.zip`
- We'll want the *full install* of OpenCV 3 (to have access to features such as SIFT and SURF, for instance), so we also need to grab the opencv_contrib repository as well:
`wget -O opencv_contrib.zip`
https://github.com/Itseez/opencv_contrib/archive/3.3.0.zip
`unzip opencv_contrib.zip`
- Before we can start compiling OpenCV on our Raspberry Pi 3, we first need to install pip, a Python package manager:
`wget https://bootstrap.pypa.io/get-pip.py`
`sudo python get-pip.py`
`sudo python3 get-pip.py`
- We are now ready to compile and install OpenCV! Just type the following commands to set up the build using CMake:
`cd ~/opencv-3.3.0/`
`mkdir build`
`cd build`

```

cmake -D CMAKE_BUILD_TYPE=RELEASE \
-D CMAKE_INSTALL_PREFIX=/usr/local \
-D INSTALL_PYTHON_EXAMPLES=ON \
-D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib-
3.3.0/modules \
-D BUILD_EXAMPLES=ON ..

```

The output will look like this :



```

pi@raspberrypi: ~/opencv-3.3.0/build
-- Use Cuda: NO
-- Use OpenCL: YES
-- Use OpenVX: NO
-- Use custom HAL: YES (carotene (ver 0.0.1))
--
-- OpenCL: <Dynamic loading of OpenCL library>
-- Include path: /home/pi/opencv-3.3.0/3rdparty/include/opencvcl/1.2
-- Use AMDFFT: NO
-- Use AMDBLAS: NO
--
-- Python 2:
-- Interpreter: /usr/bin/python2.7 (ver 2.7.13)
-- Libraries: /usr/lib/arm-linux-gnueabi/libpython2.7.so (ver 2.7.13)
-- numpy: /usr/lib/python2.7/dist-packages/numpy/core/include (ver 1.12.1)
-- packages path: lib/python2.7/dist-packages
- Python 3:
- Interpreter: /home/pi/.virtualenvs/cv/bin/python3 (ver 3.5.3)
- Libraries: /usr/lib/arm-linux-gnueabi/libpython3.5m.so (ver 3.5.3)
- numpy: /home/pi/.virtualenvs/cv/lib/python3.5/site-packages/numpy/core/include (v
r 1.13.1)
- packages path: lib/python3.5/site-packages
-- Python (for build): /usr/bin/python2.7
--
-- Java:
-- ant: NO
-- JNI: NO
-- Java wrappers: NO
-- Java tests: NO
--
-- Matlab: Matlab not found or implicitly disabled
--
-- Documentation:
-- Doxygen: NO
--
-- Tests and samples:
-- Tests: YES

```

- Finally, we are now ready to compile OpenCV:
make -j4

And the output will look like this :

```
pi@raspberrypi: ~/opencv-3.3.0/build
Scanning dependencies of target example_tapi_clahe
[ 99%] Building CXX object samples/tapi/CMakeFiles/example_tapi_clahe.dir/clahe.cpp.o
[ 99%] Linking CXX executable ../../bin/tapi-example-clahe
[ 99%] Built target example_tapi_clahe
Scanning dependencies of target example_tapi_pyrlk_optical_flow
[ 99%] Building CXX object samples/tapi/CMakeFiles/example_tapi_pyrlk_optical_flow.dir/pyrlk_optical_flow.cpp.o
[ 99%] Linking CXX executable ../../bin/tapi-example-pyrlk_optical_flow
[ 99%] Built target example_tapi_pyrlk_optical_flow
Scanning dependencies of target example_tapi_bgfg_segm
[ 99%] Building CXX object samples/tapi/CMakeFiles/example_tapi_bgfg_segm.dir/bgfg_segm.cpp.o
[ 99%] Linking CXX executable ../../bin/tapi-example-bgfg_segm
[ 99%] Built target example_tapi_bgfg_segm
Scanning dependencies of target example_tapi_camshift
[ 99%] Building CXX object samples/tapi/CMakeFiles/example_tapi_camshift.dir/camshift.cpp.o
[ 99%] Linking CXX executable ../../bin/tapi-example-camshift
[ 99%] Built target example_tapi_camshift
Scanning dependencies of target example_tapi_tvli_optical_flow
[100%] Building CXX object samples/tapi/CMakeFiles/example_tapi_tvli_optical_flow.dir/tvli_optical_flow.cpp.o
[100%] Linking CXX executable ../../bin/tapi-example-tvli_optical_flow
[100%] Built target example_tapi_tvli_optical_flow
Scanning dependencies of target example_tapi_squares
[100%] Building CXX object samples/tapi/CMakeFiles/example_tapi_squares.dir/squares.cpp.o
[100%] Linking CXX executable ../../bin/tapi-example-squares
[100%] Built target example_tapi_squares
Scanning dependencies of target example_tapi_ufacedetect
[100%] Building CXX object samples/tapi/CMakeFiles/example_tapi_ufacedetect.dir/ufacedetect.cpp.o
[100%] Linking CXX executable ../../bin/tapi-example-ufacedetect
[100%] Built target example_tapi_ufacedetect
(cv) pi@raspberrypi:~/opencv-3.3.0/build $
```

All that's left to do is now install OpenCV :

`sudo make install`

`sudo ldconfig`

Finally, we have installed OpenCV version 3.3.0 on our Raspberry Pi.

2.4 Testing the camera

Now that we have our dependencies installed and OpenCV is working properly, we can test out the camera. The camera module should be able to capture video feed in gray scale as well as colored video.

```
import numpy as np
import cv2
cap = cv2.VideoCapture(0)
cap.set(3,640) # set Width
cap.set(4,480) # set Height
while(True):
    ret, frame = cap.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

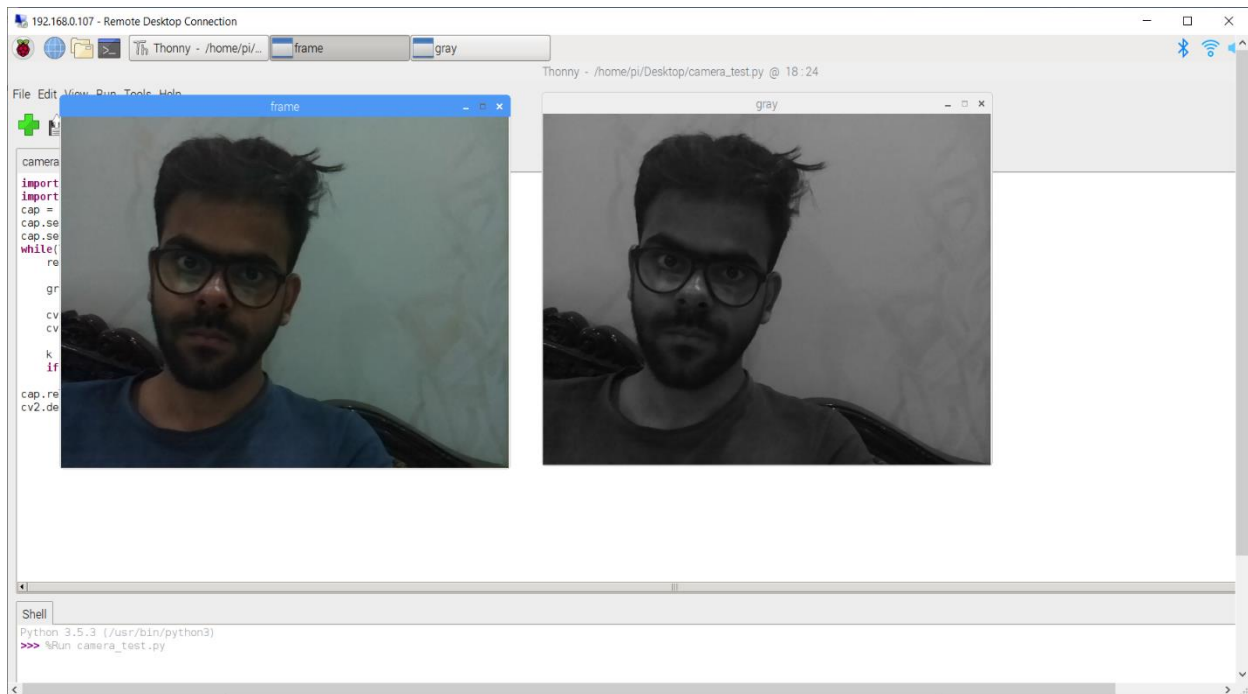
    cv2.imshow('frame', frame)
    cv2.imshow('gray', gray)
```

```

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

```

The above code captures 2 streams of video. One is in grayscale and one is colored.



2.5 Face Detection

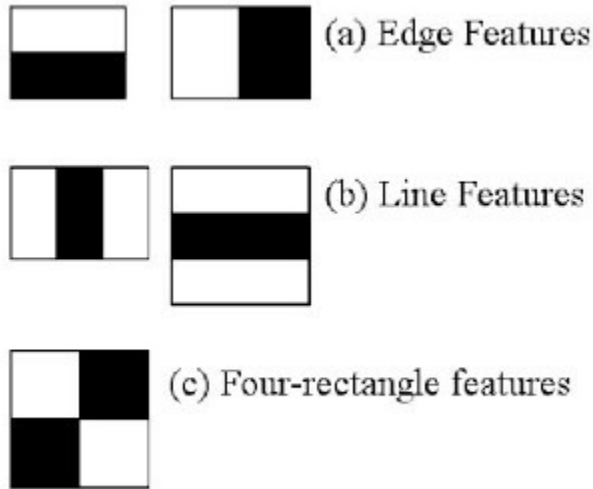
Now that our camera is working and capturing the video feed, we need an algorithm that detects faces and puts a box around the face.

For this, I used Haar Cascade Classifier.

2.5.1 Haar Cascade Classifier

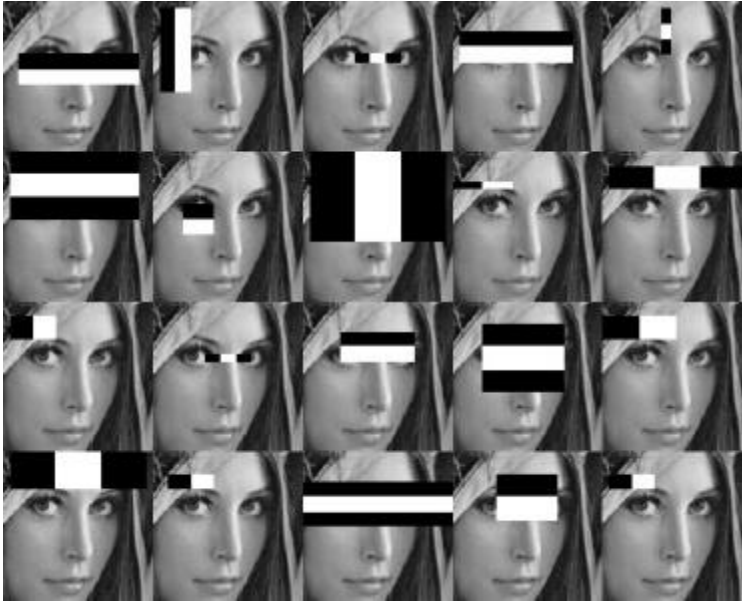
Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001. It is a machine learning based approach where a cascade function 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. For this, Haar features shown in the below image are used. They are just like our convolutional kernel. Each feature is a single value obtained by subtracting sum of pixels under the white rectangle from sum of pixels under the black rectangle.



Now, all possible sizes and locations of each kernel are used to calculate lots of features. (Just imagine how much computation it needs? Even a 24x24 window results over 160000 features). For each feature calculation, we need to find the sum of the pixels under white and black rectangles. To solve this, they introduced the integral image. However large your image, it reduces the calculations for a given pixel to an operation involving just four pixels. Nice, isn't it? It makes things super-fast.

But among all these features we calculated, most of them are irrelevant. For example, consider the image below. The top row shows two good features. The first feature selected seems to focus on the property that the region of the eyes is often darker than the region of the nose and cheeks. The second feature selected relies on the property that the eyes are darker than the bridge of the nose. But the same windows applied to cheeks or any other place is irrelevant. So how do we select the best features out of 160000+ features? It is achieved by **Adaboost**.



For this, we apply each and every feature on all the training images. For each feature, it finds the best threshold which will classify the faces to positive and negative. Obviously, there will be errors or misclassifications. We select the features with minimum error rate, which means they are the features that most accurately classify the face and non-face images. (The process is not as simple as this. Each image is given an equal weight in the beginning. After each classification, weights of misclassified images are increased. Then the same process is done. New error rates are calculated. Also new weights. The process is continued until the required accuracy or error rate is achieved or the required number of features are found).

The final classifier is a weighted sum of these weak classifiers. It is called weak because it alone can't classify the image, but together with others forms a strong classifier. The paper says even 200 features provide detection with 95% accuracy. Their final setup had around 6000 features. (Imagine a reduction from 160000+ features to 6000 features. That is a big gain).

So now you take an image. Take each 24x24 window. Apply 6000 features to it. Check if it is face or not. Wow.. Isn't it a little inefficient and time consuming?

Yes, it is. The authors have a good solution for that.

In an image, most of the image is non-face region. So it is a better idea to have a simple method to check if a window is not a face region. If it is not, discard it in a single shot, and don't process it again. Instead, focus on regions where there can be a face. This way, we spend more time checking possible face regions.

For this they introduced the concept of **Cascade of Classifiers**. Instead of applying all 6000 features on a window, the features are grouped into different stages of classifiers and applied one-by-one. (Normally the first few stages will contain very many fewer features). If a window fails the first stage, discard it.

We don't consider the remaining features on it. If it passes, apply the second stage of features and continue the process. The window which passes all stages

is a face region. How is that plan!

The authors' detector had 6000+ features with 38 stages with 1, 10, 25, 25 and 50 features in the first five stages. (The two features in the above image are actually obtained as the best two features from Adaboost). According to the authors, on average 10 features out of 6000+ are evaluated per sub-window.

So this is a simple intuitive explanation of how Viola-Jones face detection works.

I used the code below to identify faces in the video stream :

```
import numpy as np

import cv2

faceCascade=cv2.CascadeClassifier('home/pi/opencv3.3.0/data/haarcascades/Cascade/haarcascade_frontalface_default.xml')

cap = cv2.VideoCapture(0)

cap.set(3,640) # set Width

cap.set(4,480) # set Height

while 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
    break

cap.release()

cv2.destroyAllWindows()

```

I will explain each addition of code that is not there in the camera test.

- `faceCascade=cv2.CascadeClassifier('home/pi/opencv3.3.0/data/haarcascades/Cascade/haarcascade_frontalface_default.xml')`

This is the line that loads the cascade and just make sure that the path is the correct path in which the xml file is stored.

- `faces = faceCascade.detectMultiscale(
 gray,
 scaleFactor=1.2,
 minNeighbors=5,
 minsize=(20, 20)
)`

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. 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.

- *for (x,y,w,h) in faces:*

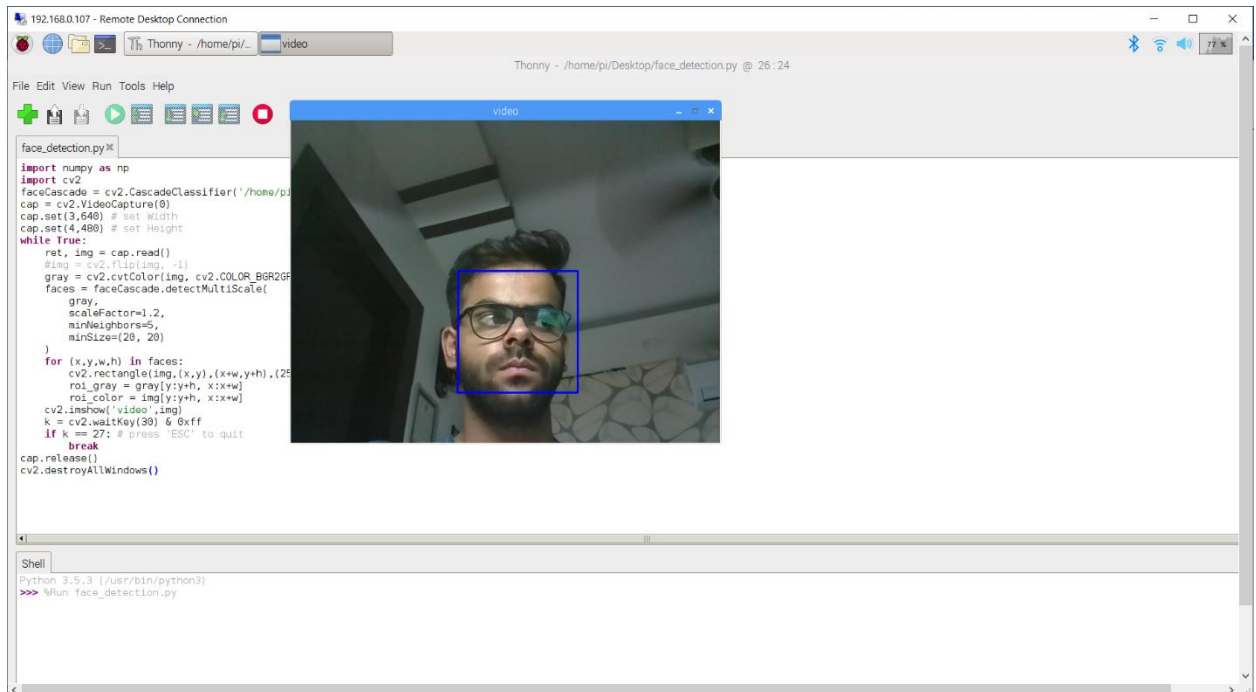
```
cv2.rectangle(img, (x,y),( x+w, y+h), (255,0,0), 2)
```

```
roi_gray = gray[y:y+h, x:x+w]
```

```
roi_color = img[y:y+h, x:x+w]
```

Next, we must "mark" the faces in the image, using, for example, a blue rectangle. This is done with this portion of the code. 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 to make a set as: (x,y,w,h).

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



We can also include classifier 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.

2.6 Creating Database for Training

In this step we will first create a folder called "Dataset" in which we will store photos of the person we want to train the recognizer on. Because Raspberry Pi model 3 had only a single GigaByte of RAM SO I was not able to train more than 80 images of a single person as my RPi could assign the memory of more images in the RAM.

I shot photos from my phone. A total of 80 photos made the database as big as 500 MB of data so I had to resize them to reduce their size. I used a free software online to resize my images to 70%

Now all the image names of all the images were not in a serial manner so I had to create some uniformity to help ease the training process. I again used code to rename all the images in my folder at once to a uniform structure. The code is as follows:

```
import os

path = os.chdir("C:\\Users\\Naman Tyagi\\Desktop\\RaspPi Project\\User1 Database")

i=27

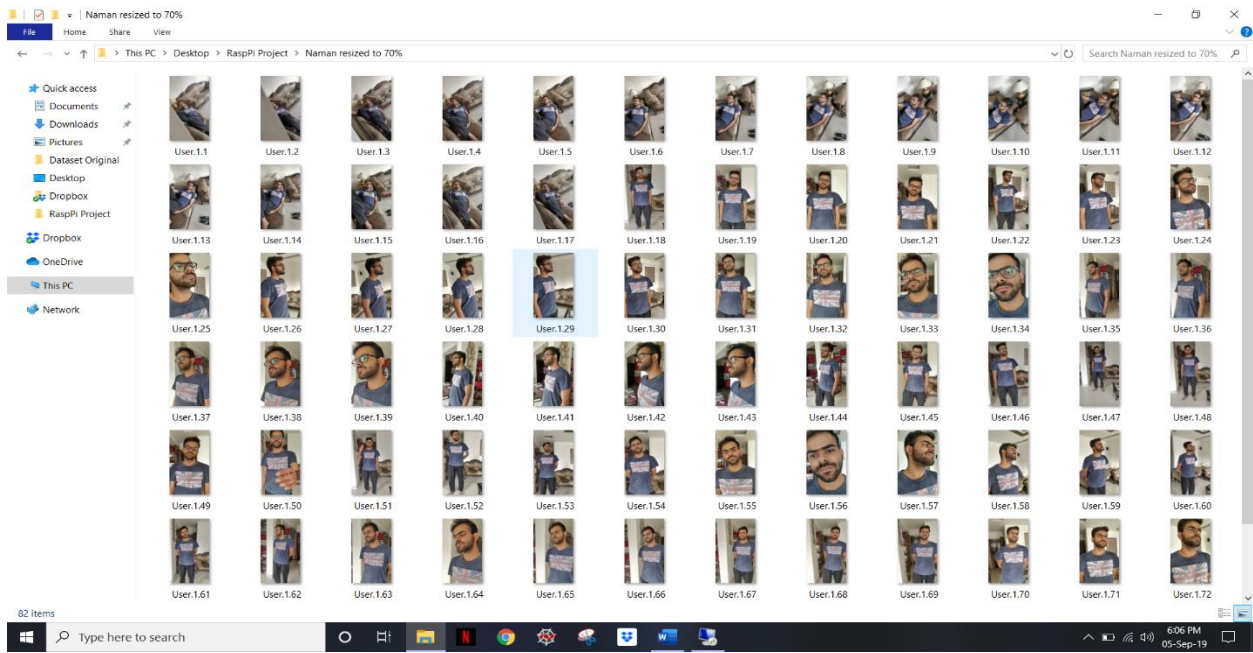
for file in os.listdir(path):

    new_file_name="User.1.{}.jpg".format(i)

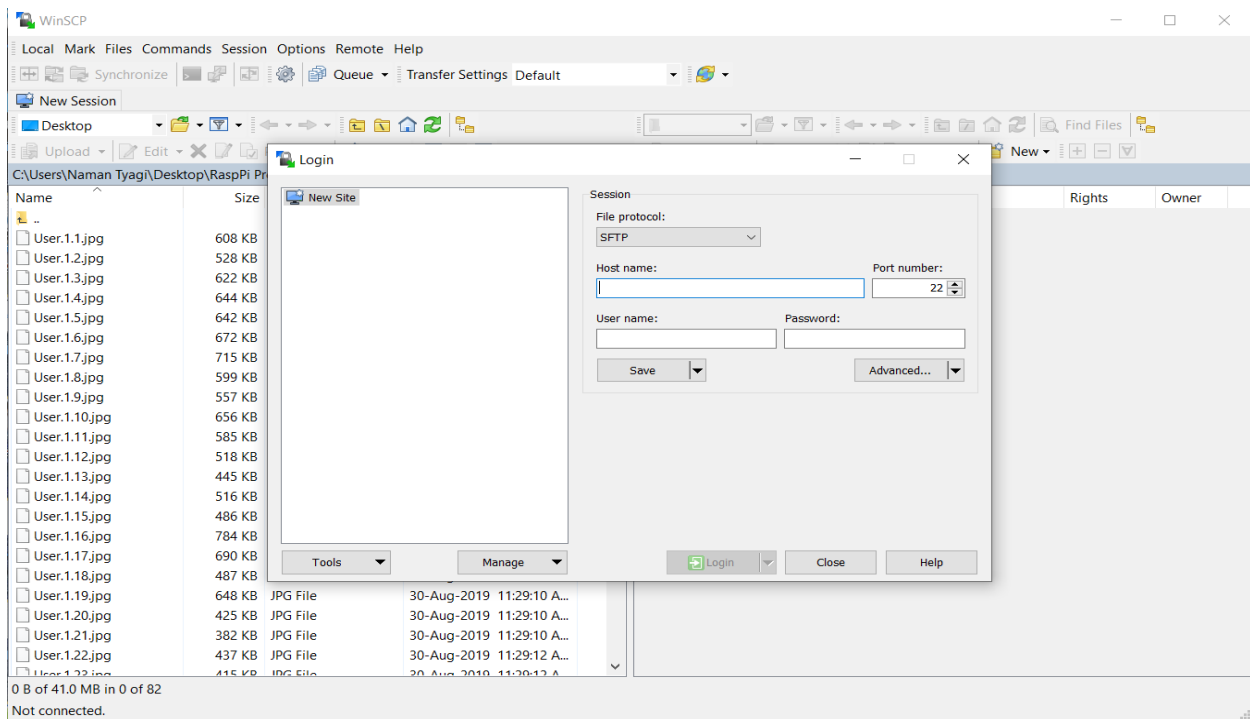
    os.rename(file, new_file_name)

    i+=1
```

The screenshot is as follows :



After this I had to transfer the images to my "Dataset" folder on my Raspberry Pi. To transfer those photos I used a software called "WinSCP" to wirelessly transfer them to the desired directory on my Raspberry Pi.



2.7 Creating the trainer file

In this section we will create a trainer file with a .xml extension in the same directory as that of the dataset. It extracts the features from the dataset and is used in the recognition process.

```
import cv2

import numpy as np

from PIL import Image

import os

# Path for face image database

path = 'home/pi/dataset'

recognizer = cv2.face.LBPHFaceRecognizer_create()

detector=cv2.CascadeClassifier('home/pi/opencv3.3.0/data/haarcascades/Cascade/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') # convert it to grayscale

        img_numpy = np.array(PIL_img,'uint8')

        id = int(os.path.splitext(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,ids

print ("\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('home/pi/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))))

```

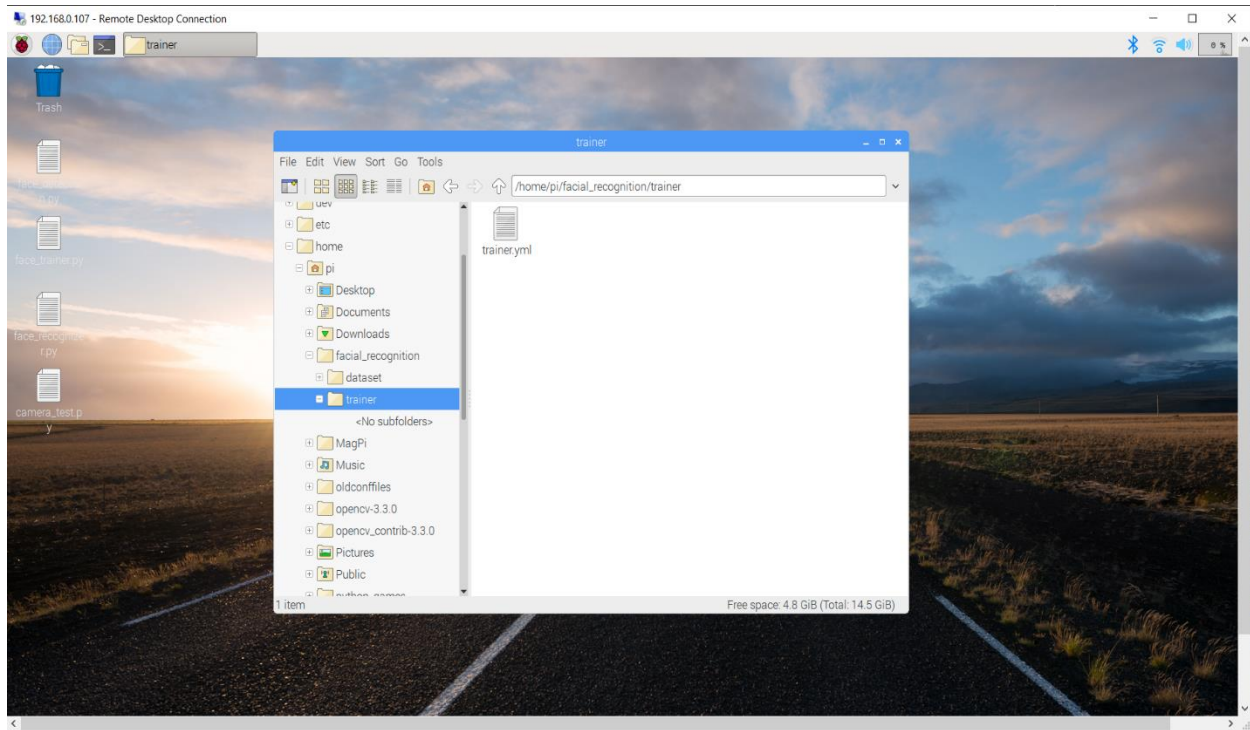
- We will use as a recognizer, the LBPH (LOCAL BINARY PATTERNS HISTOGRAMS) Face Recognizer, included on OpenCV package. We do this in the following line:

```
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, np.array(ids))
```

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



2.8 Recognizer

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.

The script for recognizing faces is as follows :

```
import cv2
import numpy as np
import os
recognizer = cv2.face.LBPHFaceRecognizer_create()
recognizer.read(home/pi/trainer/trainer.yml)
cascadePath = "home/pi/opencv3.3.0/data/haarcascades/Cascade/haarcascade_frontalface
default.xml"
faceCascade = cv2.CascadeClassifier(cascadePath);
font = cv2.FONT_HERSHEY_SIMPLEX
#initiate id counter
id = 0
# names related to ids: example ==> Naman: id=1, etc
names = ['None', 'Naman']
# Initialize and start realtime video capture
```

```

cam = cv2.VideoCapture(0)
cam.set(3, 640) # set video width
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()
    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])
        # Check if confidence is less than 100 ==> "0" is 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()

```

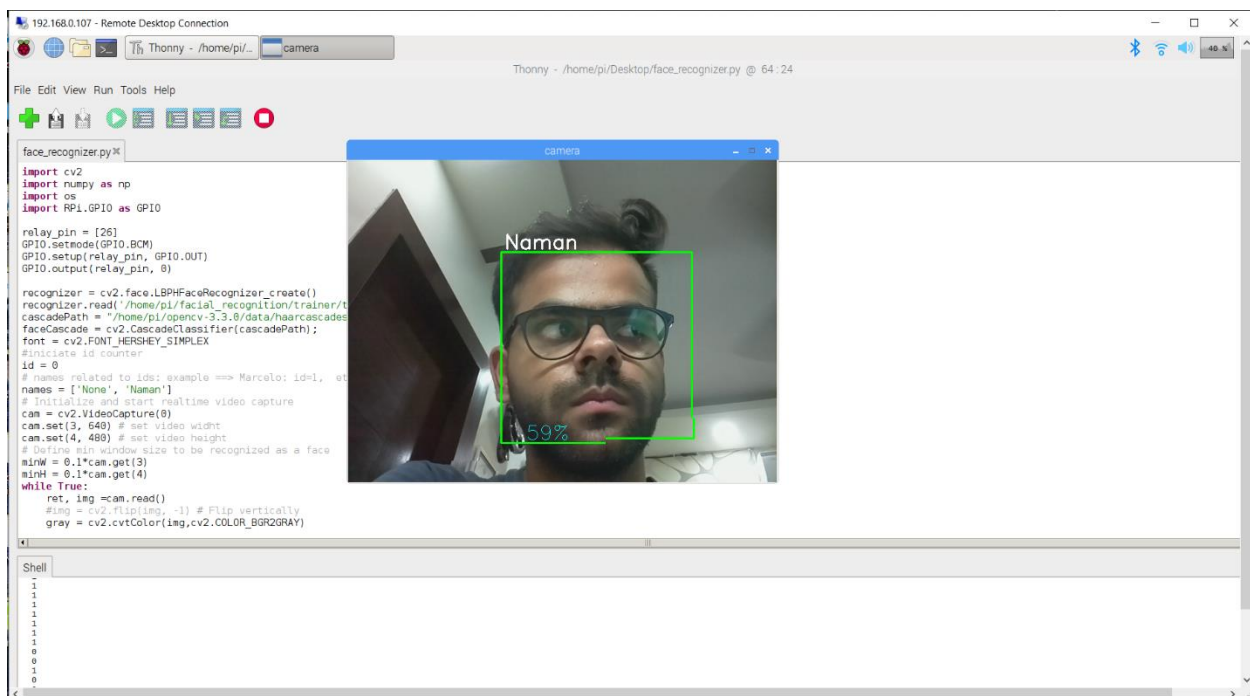
- We are including here a new array, so we will display "names", instead of numbered ids:
Names = ['None', 'Naman']
So, for example, Naman will be the user with id = 1
- Next, we will detect a face, same we did before with the haasCascade classifier. Having a detected face we can call the most important function in the above code:

id, confidence = recognizer.predict(gray[y:y+h,x:x+w])

- 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 considered 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 "unknown" label is put on the face.



Chapter 3 : Hardware

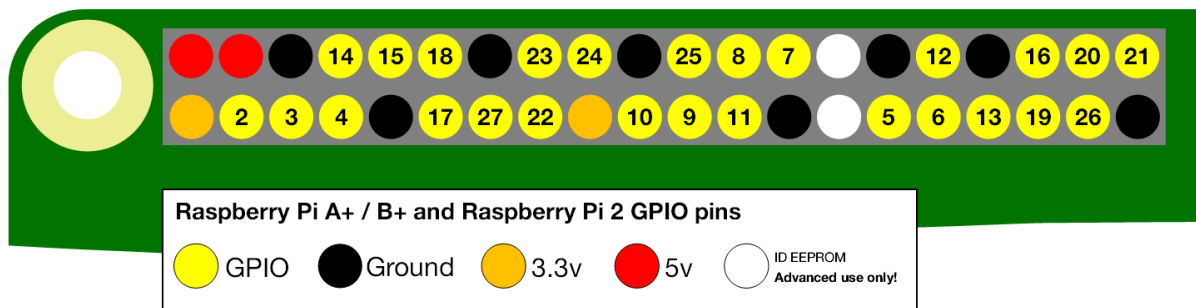
In this chapter we will work on the hardware equipment i.e connect the RPi to the solenoid lock. I will cover the topics of the GPIO pins, Relay Module, Power Supply in this chapter.

3.1 GPIO Pins

A powerful feature of the Raspberry Pi is the row of GPIO (general-purpose input/output) pins along the top edge of the board. A 40-pin GPIO header is found on all current Raspberry Pi boards (unpopulated on Pi Zero and Pi Zero W). Prior to the Pi 1 Model B+ (2014), boards comprised a shorter 26-pin header.



Any of the GPIO pins can be designated (in software) as an input or output pin and used for a wide range of purposes.



3.1.1 Voltages

Two 5V pins and two 3V3 pins are present on the board, as well as a number of ground pins (0V), which are unconfigurable. The remaining pins are all general purpose 3V3 pins, meaning outputs are set to 3V3 and inputs are 3V3-tolerant.

3.1.2 Outputs

A GPIO pin designated as an output pin can be set to high (3V3) or low (0V).

3.1.3 Inputs

A GPIO pin designated as an input pin can be read as high (3V3) or low (0V). This is made easier with the use of internal pull-up or pull-down resistors. Pins GPIO2 and GPIO3 have fixed pull-up resistors, but for other pins this can be configured in software.

3.2 Relay Module

The relay module has three high voltage terminals (NC, C, and NO) which connect to the device you want to control. The other side has three low voltage pins (Ground, Vcc, and Signal) which connect to the RPi.

5V Relay Terminals and Pins



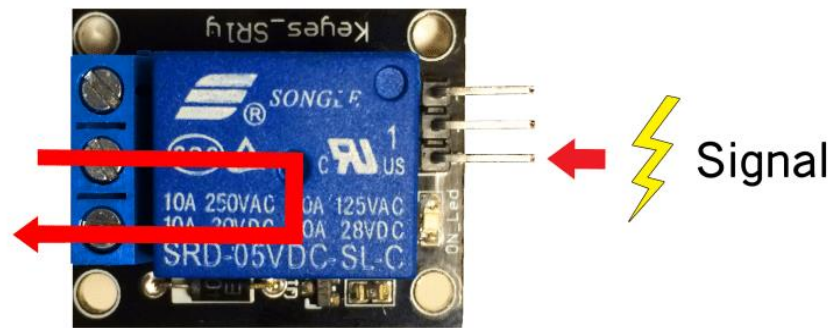
- NC: Normally closed 120-240V terminal
- NO: Normally open 120-240V terminal
- C: Common terminal
- Ground: Connects to the ground pin on the RPi
- 5V Vcc: Connects the RPi's 5V pin
- Signal: Carries the trigger signal from the RPi that activates the relay

Inside the relay is a 120-240V switch that's connected to an electromagnet. When the relay receives a HIGH signal at the signal pin, the electromagnet becomes charged and moves the contacts of the switch open or closed.

3.2.1 Normally Open

In the normally open configuration, when the relay receives a HIGH signal the 120-240V switch closes and allows current to flow from the C terminal to the NO terminal. A LOW signal deactivates the relay and stops the current. So if you want the HIGH signal to turn ON the relay, use the normally open terminal:

Normally
Open



3.2.2 Normally Closed

In the normally closed configuration, a HIGH signal opens the switch and interrupts the 120-240V current. A LOW signal closes the switch and allows current to flow from the C terminal to the NC terminal. Therefore, if you want the HIGH signal to turn OFF the 120-240V current, use the

normally closed terminal:



3.3 Solenoid Lock



The principle behind an electromagnetic lock is the use of electromagnetism to lock a door when energized. The holding force should be collinear with the load, and the lock and armature plate should be face-to-face to achieve optimal operation.



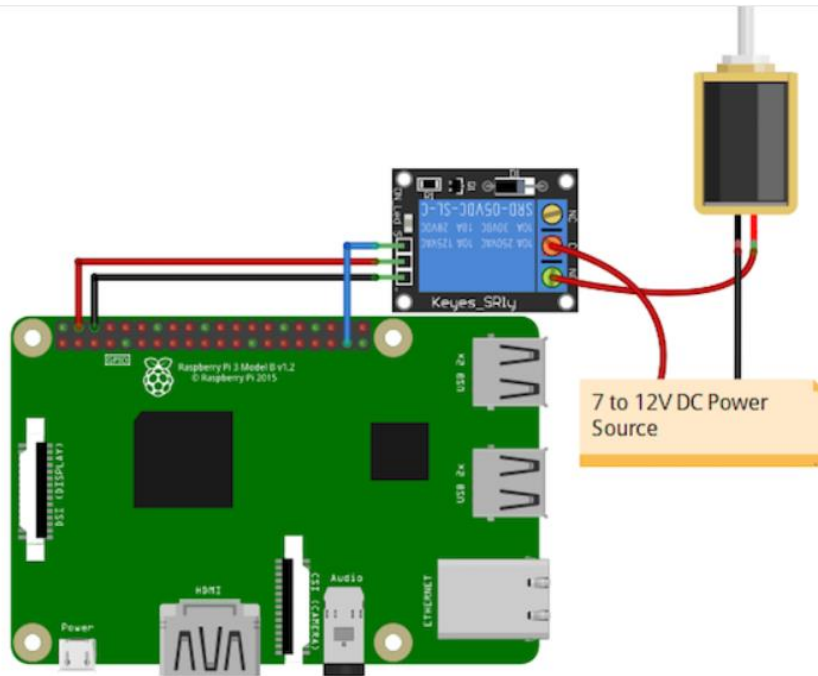
3.4 Power Supply

We can use a power supply that plugs into an AC supply and steps down the voltage from the normal 220V to 12V for safe working of the solenoid lock or we can use a power bank for portability as it does not require a connection to the AC supply but in case of a power bank we need to step up the voltage from the normal 5V to 12V in order for the solenoid lock to work.

3.5 Circuit Diagram

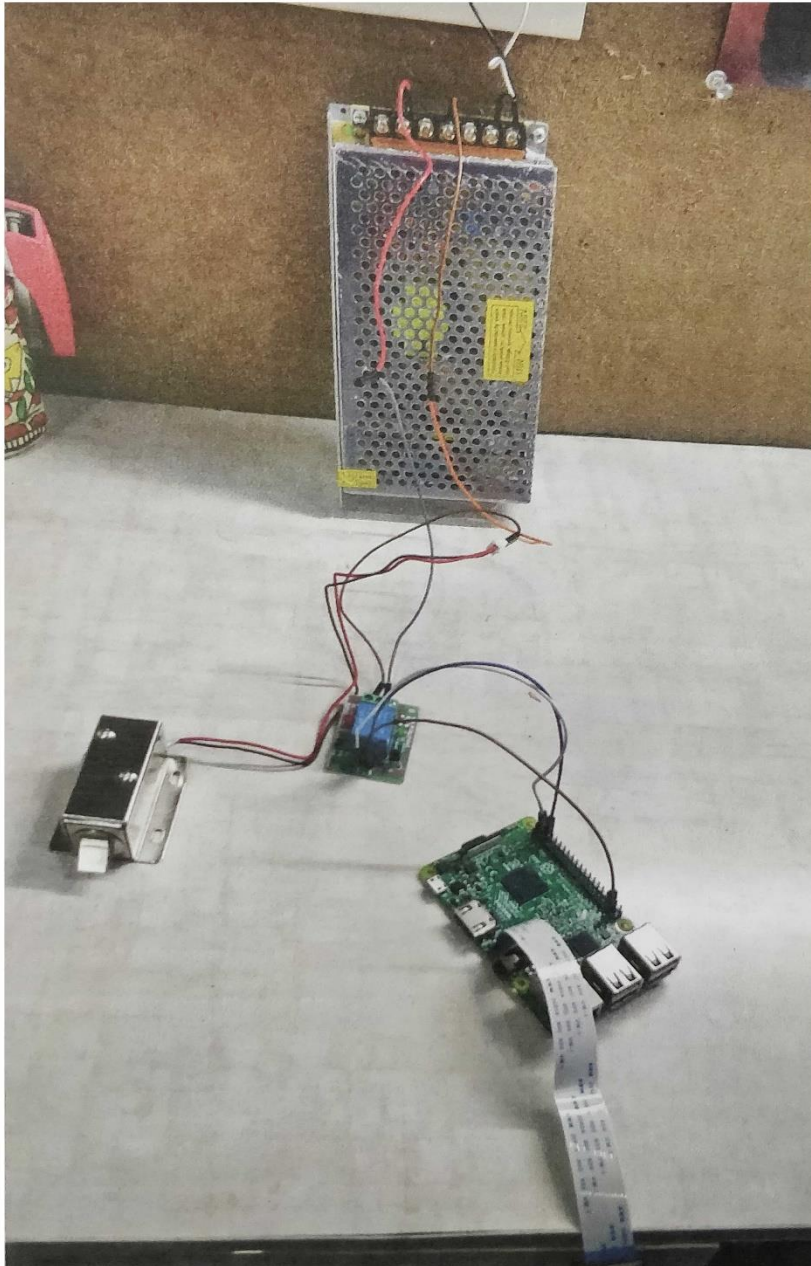
Connect the VCC and GND of the relay module to 5V and GND of Raspberry Pi. Then connect the signal pin of the relay module to the GPIO 26 of Raspberry Pi.

On the other side of the relay module, connect the negative form DC power source to the negative of the solenoid door lock. Connect the positive from the DC power source to the common of the relay module and then connect normally open from the relay module to positive of the solenoid door lock.



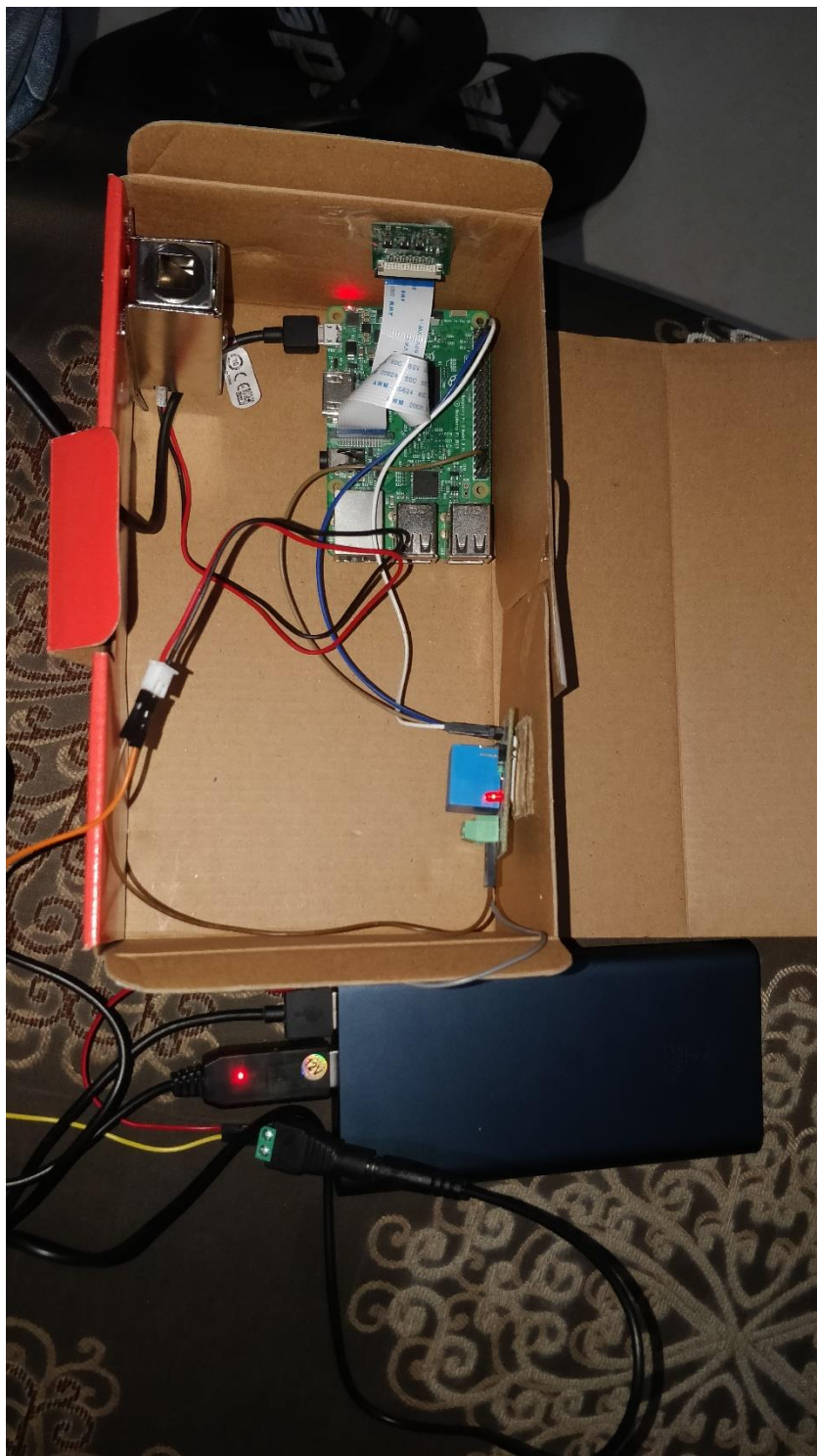
I have designed my circuit in both the ways. I have used a power supply as well as a power bank to make the entire system portable.

The following image is that of the circuit using an external power supply. It can be seen in the picture that 2 wire are going upwards and plugging into the switchboard.



PhotoScan by Google Photos

In the next picture I have replaced the power supply with a power bank. To connect the power bank to the lock I had to use a USB cable which bumps the voltage from the power bank (5V) to the required 12V. One end of that cable is USB-A and the other end is DC wire, so to separate the DC wire into positive and negative terminals I had to use a female connector.



3.6 The final code

After all this effort, I have made some modifications to the code of the face recognizer. I have used the time module to keep the lock unlocked for 5 seconds everytime it sees the face on the database so that the user can open the drawer, cabinet that he has locked. To close the drawer, cabinet again, the user should show his face again and 5 seconds later the lock will lock itself giving the user enough time to close the cabinet, drawer.

The code is as follows :

```
import cv2
import numpy as np
import os
import RPi.GPIO as GPIO
import time

relay_pin = [26]
GPIO.setmode(GPIO.BCM)
GPIO.setup(relay_pin, GPIO.OUT)
GPIO.output(relay_pin, 1)

recognizer = cv2.face.LBPHFaceRecognizer_create()
recognizer.read('/home/pi/facial_recognition/trainer/trainer.yml')
cascadePath = "/home/pi/opencv-3.3.0/data/haarcascades/haarcascade_frontalface_default.xml"
faceCascade = cv2.CascadeClassifier(cascadePath);
font = cv2.FONT_HERSHEY_SIMPLEX
#initiate id counter
id = 0
names = ['None', 'Naman']
# Initialize and start realtime video capture
cam = cv2.VideoCapture(0)
cam.set(3, 640) # set video width
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()
    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])

        #Check if confidence is less them 100 ==> "0" is perfect match
        if (confidence < 50):

            id = names[id]
            confidence = " {0}%".format(round(100 - confidence))
            GPIO.output(relay_pin, 0)
            time.sleep(5)
            GPIO.output(relay_pin, 1)
            break

        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()

```

The basic idea behind the modifications in the above script is controlling the lock and pausing the code for 5 seconds.

```

Import time
Import RPi.GPIO as GPIO

Relay_pin=[26]
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(relay_pin, GPIO.OUT)
GPIO.output(relay_pin, 1)
Time.sleep(5)
GPIO.output(relay_pin, 0)

```

```
Time.sleep(5)  
GPIO.output(relay_pin, 1)
```

The above code locked the lock and the waited 5 seconds to unlock and then again waited 5 seconds to lock again.

Chapter 4 : Difficulties, Bugs and Future Scope

4.1 Difficulties

- In the first shot, for some reason my OpenCV compilation was stuck at 97% and not moving forward. After hours of research, I had to erase the OS and download the OS to which OpenCV-3.3.0 was compatible. It is a very tedious and very long process. It really checks one's patience.
- In the second shot also, I used virtual environment. Virtual environment is nothing but an isolated environment on your RPi in which you can run different functions of OpenCV without affecting other environments. This is preferred if you want to use a single RPi for multiple functionalities using different functions of OpenCV without affecting one another. But for some reason the GPIO module was not installing in the virtual environment and I could not proceed with the circuit building. It was only much later that I realized this functionality of the virtual environment. It is useless to me as I am using my RPi for one purpose only. So I again erased the SD Card and installed OpenCV without virtual environment.
- I did my training in Data Science and Machine Learning so I had no prior knowledge of the various components of the circuits. But thanks to my hard work and determination, I learnt the entire functioning of all the components of the circuit. It was time taking but not impossible. After all, knowledge can never be waste.
- I tried improving the accuracy of the recognition by parameter tuning, increasing the number of photos in the database, compressing them to various sizes to get better results but since I am limited to only a single gigabyte of RAM, I could not train more than 80 images at once.

4.2 Bugs

- Due to the weak processing power, everytime a face comes in the video stream, the frames fall drastically as the algorithm is running on each frame. Also, when the video stream pauses itself for 5 seconds when it comes across the `time.sleep(5)` line, if the user's face is on the frame that has paused then after 5 seconds the processor again recognizes that frame and unlocks the lock, this keeps happening until you exit the program and start again.
- It can be unlocked by the user's photograph as it uses optical face recognition. To make it more secure, a dot projector can be used to form a 3-D matrix of the user's face.

4.3 Future Scope

The future is using AI to make our lives much more simple and convenient.

This was a small effort to adhere to our day to day problems. This model can be perfected by making the following amends :

- A motion detector can be attached to it which turns on the whole system whenever there is movement around it. When no one is around the system, it is powered off to save energy.
- We can also connect a hard drive to record footage whenever the motion detector turns on the system thereby not recording the unnecessary footage and saving on electricity, space on the hard drive and thus making it less time consuming for the user to go through the footage.
- The camera attached to it can be more advanced having a wide aperture to let more light in thus making it easy for the software to identify faces. The lens used preferably should be a wide angle lens so that it covers a much larger area in the frame.
- We can train all the faces in the user's family so that it is able to distinguish between the user's family and unknown strangers. Furthermore, we can design a code that learns about the "usual" activities of the user's family and whenever there is some "unusual" activity by unknown strangers, it can alert the user inside the house by sending some sort of message on the user's smartphone.
- We can use a faster processor to eliminate the bug encountered in my project where after the line time.sleep(5) the frame freezes and software keeps on detecting the user's face on the frame.
- We can use a better solenoid lock that doesn't heat up as much when running as the lock that I am using in my project.

REFERENCES

To learn about the haar cascade classifier, I went to OpenCV website. The link is as follows:

https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html

To find help in recording, displaying the video captured by the PiCamera module, I found this website :

<https://www.learnopencv.com/read-write-and-display-a-video-using-opencv-cpp-python/>

To learn about the raspberry pi, its working, its components, raspberry pi's own website is extremely useful. The link is as follows:

<https://www.raspberrypi.org/>


To learn about the connections and working of the various electrical components, I scoured YouTube, the web and so on. Some references are as follows :

<http://www.circuitbasics.com/setting-up-a-5v-relay-on-the-arduino/>

<https://www.machinedesign.com/fasteners/how-tailor-electronic-locks-your-applications>

Below are screenshots of all the materials I used to make this product :

Delivered 23-Sep-2019
Package was handed directly to the customer.
Signed by: Naman Tyagi.



TES DC Power Female Jack Connector Plugs 5 Pcs
Sold by: KVR Online
₹199.00
[Buy it again](#)

[Track package](#)
[Return or replace items](#)
[Share gift receipt](#)
[Leave seller feedback](#)
[Leave delivery feedback](#)


ORDER PLACED
19 September 2019

TOTAL
₹1,148.00

SHIP TO
Naman Tyagi

ORDER # 406-2983256-5709955
[Order Details](#) | [Invoice](#)


Delivered 24-Sep-2019
Package was handed directly to the customer.
Signed by: Naman Tyagi.



Sunrobotics USB DC 5V TO DC 12V 0.8A MALE BARREL STEP UP CONVERTER
Sold by: SunRobotics®
Return eligible through 04-Oct-2019
₹249.00
[Buy it again](#)

[Track package](#)
[Return or replace items](#)
[Share gift receipt](#)
[Leave seller feedback](#)
[Leave delivery feedback](#)
[Write a product review](#)

Delivered 22-Sep-2019




Mi 10000mAh Li-Polymer Power Bank 2i (Black) with 18W Fast Charging
Sold by: Appario Retail Private Ltd
Replacement may be possible through 05-Oct-2019
₹899.00
[Buy it again](#)

[Track package](#)
[Return or replace items](#)
[Share gift receipt](#)
[Leave seller feedback](#)

41

Delivered 21-Aug-2019
Signed by: Naman Tyagi.




ElectroBot Breadboard Jumper Wires Ribbon Cables Kit, Multicolored (120 Pieces)
Sold by: Cloudtail India
Return window closed on 31-Aug-2019
₹160.00

Buy it again

[Track package](#)
[Leave seller feedback](#)
[Write a product review](#)

Delivered 21-Aug-2019
Signed by: Naman Tyagi.




DC12V Lock Tongue Luggage Electric Solenoid Assembly for Auto Door Sauna Cabinet Drawer
Sold by: Bigbig Mall
Return window closed on 31-Aug-2019
₹496.00

Buy it again

[Track package](#)
[Leave seller feedback](#)
[Write a product review](#)

ORDER PLACED 20 August 2019	TOTAL ₹483.00	SHIP TO Naman Tyagi	ORDER # 405-6760082-8548350 Order Details Invoice
--------------------------------	------------------	------------------------	--

Delivered 22-Aug-2019




ZVision 12V 10Amp 120W DC Power Supply Driver for CCTV and LED Strip light Lamp 12 Volt 10A
Sold by: TRP TRADERS | Product question? Ask Seller
₹483.00

Buy it again

[Track package](#)
[Return or replace items](#)
[Share gift receipt](#)
[Leave seller feedback](#)

ORDER PLACED 20 August 2019	TOTAL ₹160.00	SHIP TO Naman Tyagi	ORDER # 405-0083668-2916340 Order Details Invoice
--------------------------------	------------------	------------------------	--

Delivered 22-Aug-2019




ElectroBot EB-1CH-REB-1 5V Single Channel Relay Module for Arduino, AVR, PIC, ARM7, 8051, Raspberry Pi with Indicator Light (1)
Sold by: ElectroBot | Product question? Ask Seller
Return window closed on 01-Sep-2019
₹160.00

Buy it again

[Track package](#)
[Ask Product Question](#)
[Leave seller feedback](#)
[Write a product review](#)

ORDER PLACED 11 August 2019	TOTAL ₹250.00	SHIP TO Naman Tyagi	ORDER # 405-9744210-6295566 Order Details Invoice
--------------------------------	------------------	------------------------	--

Delivered 12-Aug-2019
Signed by: Naman Tyagi.



Zebtronics Zeb KM2100 Multimedia USB Keyboard
Sold by: Appario Retail Private Ltd
Return window closed on 22-Aug-2019
₹250.00

Buy it again

[Leave seller feedback](#)
[Write a product review](#)

ORDER PLACED 10 August 2019	TOTAL ₹799.00	SHIP TO Naman Tyagi	ORDER # 405-4368636-4530738 Order Details Invoice
--------------------------------	------------------	------------------------	--

Delivered 06-Aug-2019
Signed by: Naman Tyagi.



Raspberry Pi 3-MODB-1GB Motherboard (Black)
Sold by: ElectroBot | Product question? Ask Seller
Return window closed on 16-Aug-2019
₹2,579.00

Buy it again

[Ask Product Question](#)
[Leave seller feedback](#)
[Write a product review](#)

ORDER PLACED 5 August 2019	TOTAL ₹390.00	SHIP TO Naman Tyagi	ORDER # 405-1281661-6156331 Order Details Invoice
-------------------------------	------------------	------------------------	--

Delivered 06-Aug-2019
Signed by: Naman Tyagi.



Raspberry Pi SMP Camera Board Module
Sold by: Allianz Technologies
₹190.00

Buy it again

[Return or replace items](#)
[Share gift receipt](#)
[Leave seller feedback](#)