DOOR ACCESS CONTROL SYSTEM WITH FACE RECOGNITION

GROUP MEMBERS

IIT2019037 - ABHISHEK BHAWARE
IIT2019034 - RAJAT MEHRA
IIT2019030 - KAUSHAL KUMAR
IIT2019061 - RAJPAL SINGH SHEKHAWAT

PROBLEM STATEMENT

Door access control system using face recognition

Why we’ve chosen it

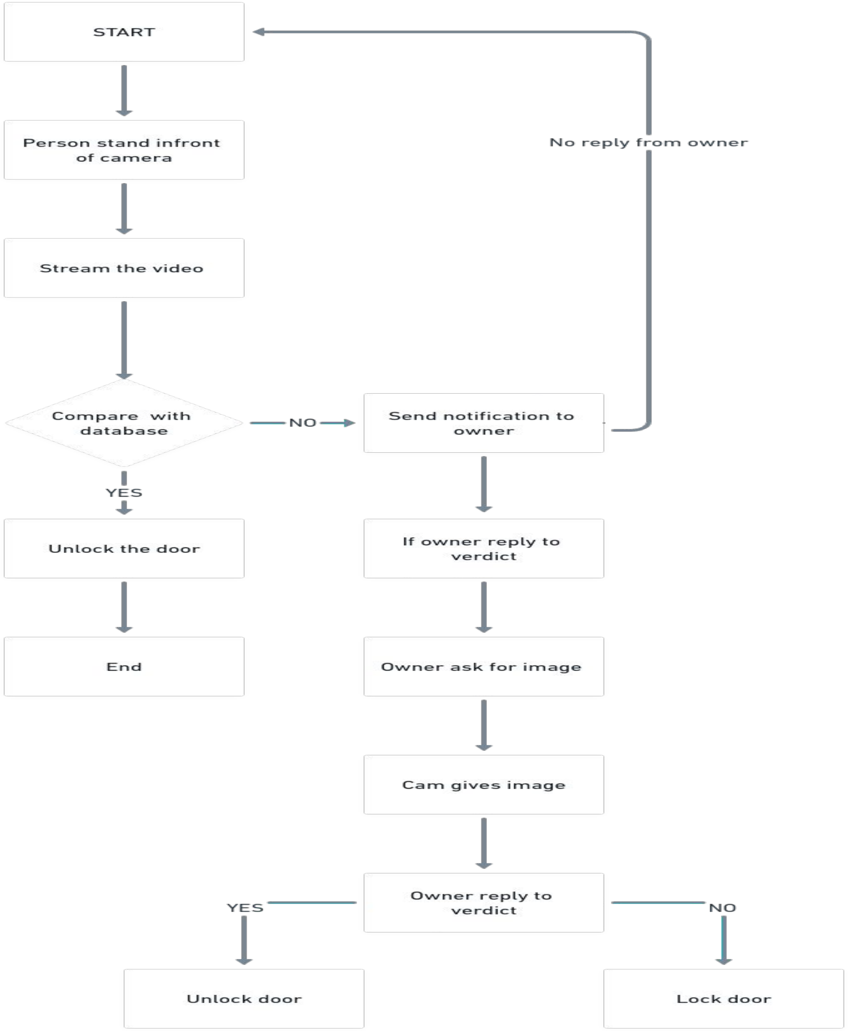
In today's world, home security is of utmost priority. IOT (Internet of Things) being an emerging technology can be used along with facial recognition to make our task of providing smart home security easier, simpler and foolproof.

The Face Detection System (FDRS) is a technology that recognizes body features by using mathematical factors inherent in human appearance. This technology is easy to use and secure. The Internet of Things (IoT) is a popular technology that allows you to track and control harmful devices in your house. Identifying a person to enter and exit the house is an important aspect of a home security system.

APPLICATION

The aim of this project is to assist users for improvement of the door security of sensitive locations by using face detection and recognition. The proposed system mainly consists of subsystems namely image capture, face detection and recognition, notification and automatic door access management. Face Recognition supported openCV is brought up because it uses Eigen faces(it utilizes linear algebra and Principal Component Analysis (PCA) to perform face recognition). and reduces the scale of face images without losing vital features, facial images for many persons can be stored in the database. The door lock can also be accessed remotely from any part of the room. The captured image from esp32 camera will be sent to the authorized person.

USE CASE DIAGRAM



SOLUTION

- HARDWARE requirements
  - ESP32 CAM WiFi Module Bluetooth with OV2640 Camera Module 2MP For Face Recognition
  - FT232RL USB to TTL 3.3V 5.5V Serial Adapter Module
  - Bread Board
  - Arduino Uno
  - Jumper wires
  - Serial port USB cable 5V mini
  - Electronic door lock 12V
  - PCB Mounted Passive Buzzer Module
  - 2 Channel 5V Relay Module with Optocoupler
  - Battery
- Software requirements
  - OpenCV python package
  - Arduino software
  - Web Browser
  - Raspberry Pi/ PC as server
- Technology/language requirements
  - C++/C
  - Python

Deliverable

User can expect:

- Fully functional Facial recognition device.
- User also get a list of people with time who are coming in front of camera.
- The device setup also consist of a strong solenoid lock.
- For security purposes, video streaming can only be seen if both the devices are on same network.

Timeline

Module 1 - Connect esp32 cam with PC and stream the data in real time from esp32 to processing unit, and for security purpose we are transferring the data directly by wifi. The only requirement is that both sender and receiver has to be connected to that same wifi service.

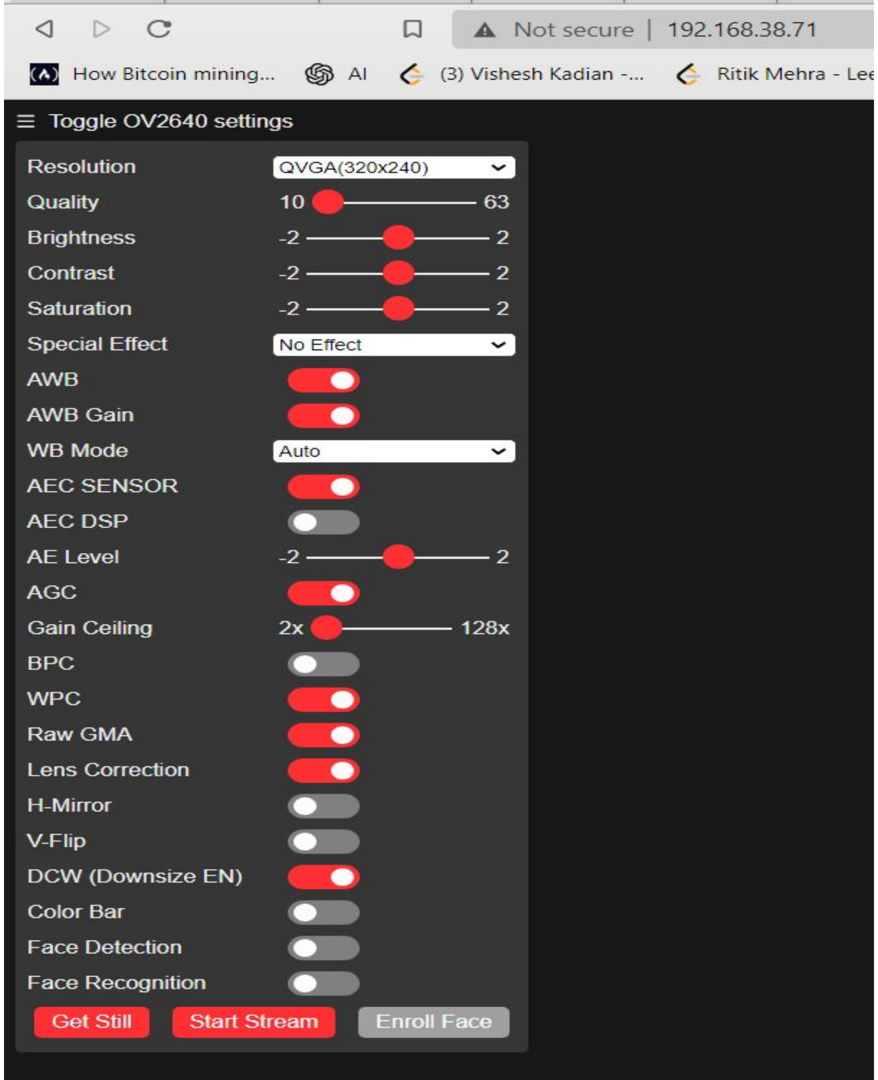
Module 2 - Processing unit is able to process the real time stream, applies Face Recognition model and check that person in our database. It then sends the verdict to the solenoid lock. If it is “YES”, we open the lock, else we take some photos of that person, and send them to the user. If user allows that person, lock opens and that person is registered in the database else the lock remains closed. Also if user ignores, then that is interpreted as “NO”.

Module 3 - With the given verdict processing unit is able to open the lock and also maintaining a excel sheet by which client can get info about who is entering at what time.

Implementation of Module 1

- Screenshot
- Readme file Explaining in details how to run your code
- Code
- Sample input/output

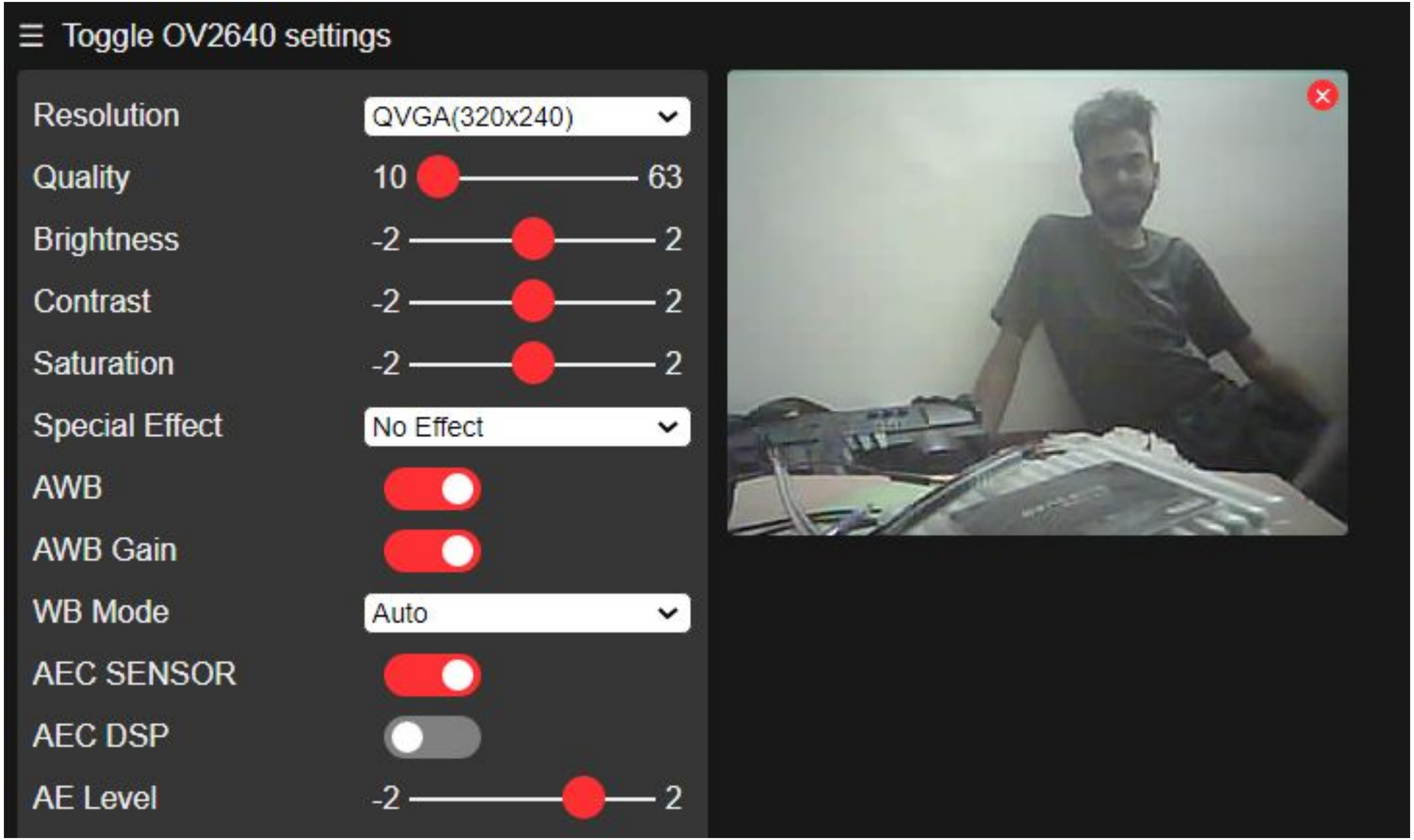
Screenshot of ESP32 Website where video is streamed



≡ Toggle OV2640 settings

Resolution	QVGA(320x240)
Quality	10 ● 63
Brightness	-2 ● 2
Contrast	-2 ● 2
Saturation	-2 ● 2
Special Effect	No Effect
AWB	☑
AWB Gain	☑
WB Mode	Auto
AEC SENSOR	☑
AEC DSP	☐
AE Level	-2 ● 2
AGC	☑
Gain Ceiling	2x ● 128x
BPC	☐
WPC	☑
Raw GMA	☑
Lens Correction	☑
H-Mirror	☐
V-Flip	☐
DCW (Downsize EN)	☑
Color Bar	☐
Face Detection	☐
Face Recognition	☐

Video streamed on site



☰ Toggle OV2640 settings

Resolution: QVGA(320x240)

Quality: 10 — 63

Brightness: -2 — 2

Contrast: -2 — 2

Saturation: -2 — 2

Special Effect: No Effect

AWB ☑

AWB Gain ☑

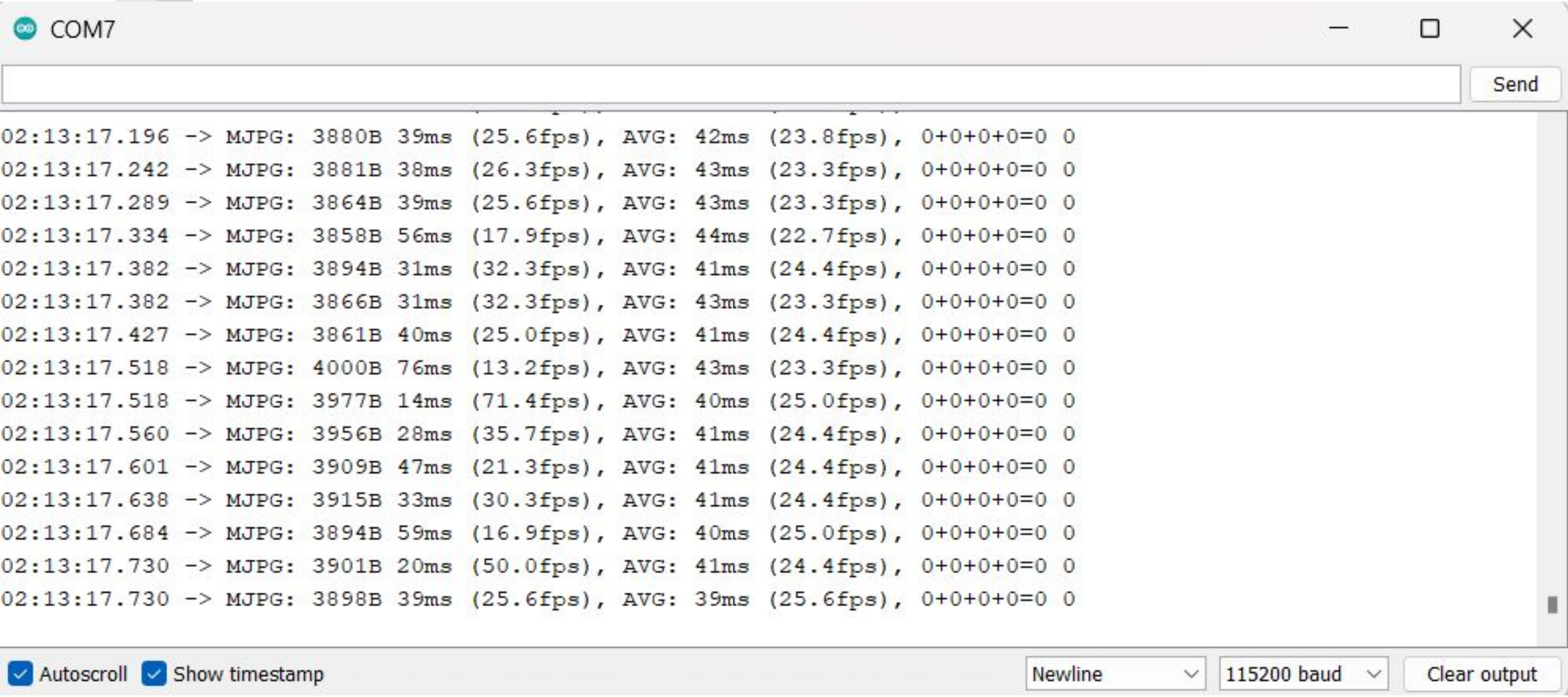
WB Mode: Auto

AEC SENSOR ☑

AEC DSP ☐

AE Level: -2 — 2

Continuous Image Transfer log



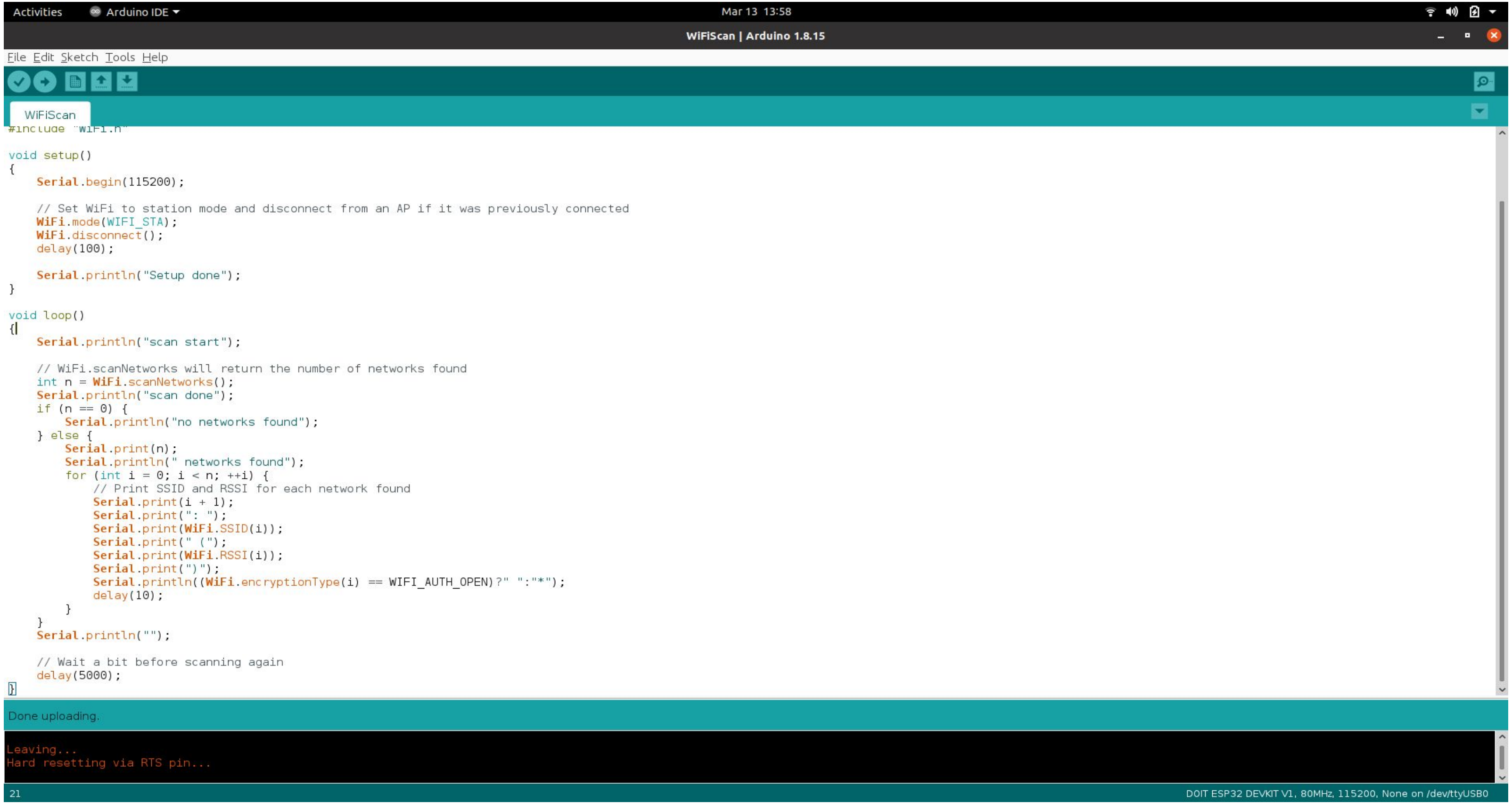
```
02:13:17.196 -> MJPG: 3880B 39ms (25.6fps), AVG: 42ms (23.8fps), 0+0+0+0=0 0
02:13:17.242 -> MJPG: 3881B 38ms (26.3fps), AVG: 43ms (23.3fps), 0+0+0+0=0 0
02:13:17.289 -> MJPG: 3864B 39ms (25.6fps), AVG: 43ms (23.3fps), 0+0+0+0=0 0
02:13:17.334 -> MJPG: 3858B 56ms (17.9fps), AVG: 44ms (22.7fps), 0+0+0+0=0 0
02:13:17.382 -> MJPG: 3894B 31ms (32.3fps), AVG: 41ms (24.4fps), 0+0+0+0=0 0
02:13:17.382 -> MJPG: 3866B 31ms (32.3fps), AVG: 43ms (23.3fps), 0+0+0+0=0 0
02:13:17.427 -> MJPG: 3861B 40ms (25.0fps), AVG: 41ms (24.4fps), 0+0+0+0=0 0
02:13:17.518 -> MJPG: 4000B 76ms (13.2fps), AVG: 43ms (23.3fps), 0+0+0+0=0 0
02:13:17.518 -> MJPG: 3977B 14ms (71.4fps), AVG: 40ms (25.0fps), 0+0+0+0=0 0
02:13:17.560 -> MJPG: 3956B 28ms (35.7fps), AVG: 41ms (24.4fps), 0+0+0+0=0 0
02:13:17.601 -> MJPG: 3909B 47ms (21.3fps), AVG: 41ms (24.4fps), 0+0+0+0=0 0
02:13:17.638 -> MJPG: 3915B 33ms (30.3fps), AVG: 41ms (24.4fps), 0+0+0+0=0 0
02:13:17.684 -> MJPG: 3894B 59ms (16.9fps), AVG: 40ms (25.0fps), 0+0+0+0=0 0
02:13:17.730 -> MJPG: 3901B 20ms (50.0fps), AVG: 41ms (24.4fps), 0+0+0+0=0 0
02:13:17.730 -> MJPG: 3898B 39ms (25.6fps), AVG: 39ms (25.6fps), 0+0+0+0=0 0
```

☑ Autoscroll ☑ Show timestamp

Newline

115200 baud

Code Screenshots



```
#include "WiFi.h"

void setup()
{
    Serial.begin(115200);

    // Set WiFi to station mode and disconnect from an AP if it was previously connected
    WiFi.mode(WIFI_STA);
    WiFi.disconnect();
    delay(100);

    Serial.println("Setup done");
}

void loop()
{
    Serial.println("scan start");

    // WiFi.scanNetworks will return the number of networks found
    int n = WiFi.scanNetworks();
    Serial.println("scan done");
    if (n == 0) {
        Serial.println("no networks found");
    } else {
        Serial.print(n);
        Serial.println(" networks found");
        for (int i = 0; i < n; ++i) {
            // Print SSID and RSSI for each network found
            Serial.print(i + 1);
            Serial.print(": ");
            Serial.print(WiFi.SSID(i));
            Serial.print(" (");
            Serial.print(WiFi.RSSI(i));
            Serial.print(")");
            Serial.println((WiFi.encryptionType(i) == WIFI_AUTH_OPEN)?" ":"*");
            delay(10);
        }
    }
    Serial.println("");

    // Wait a bit before scanning again
    delay(5000);
}
```

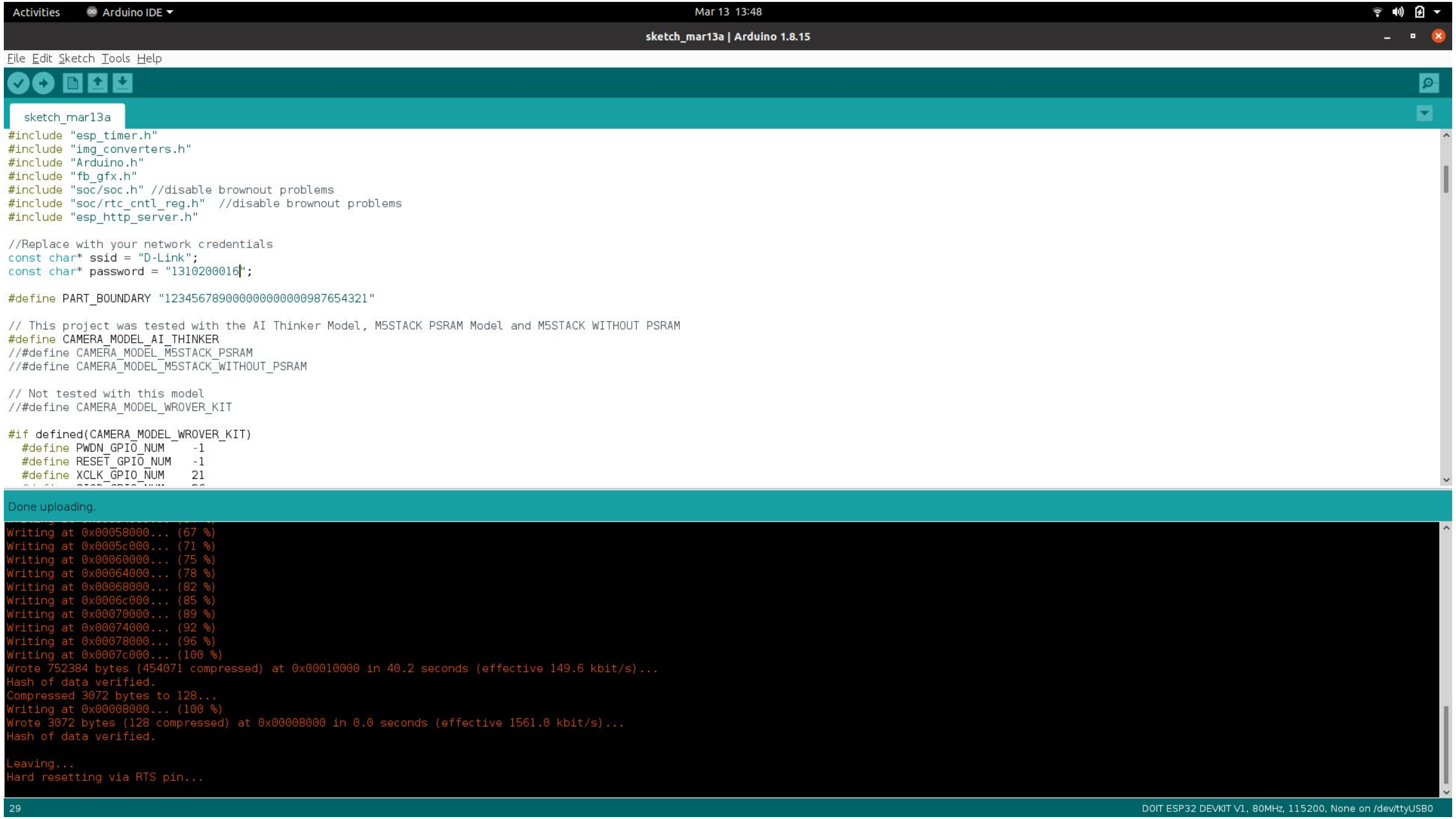
Done uploading.

```
Leaving...
Hard resetting via RTS pin...
```

21

DOIT ESP32 DEVKIT V1, 80MHz, 115200, None on /dev/ttyUSB0

Code Screenshots



```
#include "esp_timer.h"
#include "img_converters.h"
#include "Arduino.h"
#include "fb_gfx.h"
#include "soc/soc.h" //disable brownout problems
#include "soc/rtc_cntl_reg.h"  //disable brownout problems
#include "esp_http_server.h"

//Replace with your network credentials
const char* ssid = "D-Link";
const char* password = "1310200016";

#define PART_BOUNDARY "123456789000000000000987654321"

// This project was tested with the AI Thinker Model, M5STACK PSRAM Model and M5STACK WITHOUT PSRAM
#define CAMERA_MODEL_AI_THINKER
//#define CAMERA_MODEL_M5STACK_PSRAM
//#define CAMERA_MODEL_M5STACK_WITHOUT_PSRAM

// Not tested with this model
//#define CAMERA_MODEL_WROVER_KIT

#if defined(CAMERA_MODEL_WROVER_KIT)
  #define PWDN_GPIO_NUM    -1
  #define RESET_GPIO_NUM   -1
  #define XCLK_GPIO_NUM    21
```

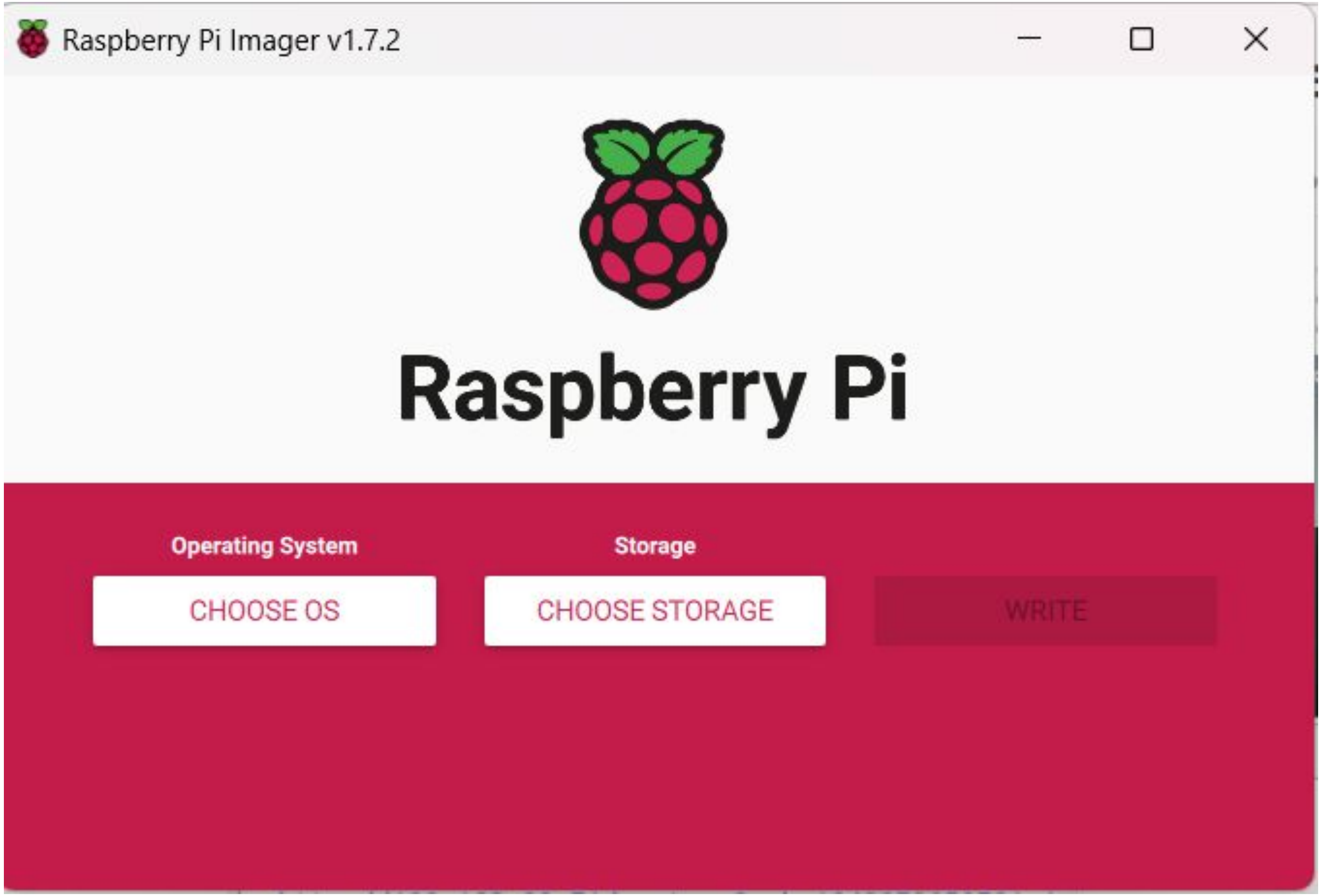
Done uploading.

```
Writing at 0x00058000... (67 %)
Writing at 0x0005c000... (71 %)
Writing at 0x00060000... (75 %)
Writing at 0x00064000... (78 %)
Writing at 0x00068000... (82 %)
Writing at 0x0006c000... (85 %)
Writing at 0x00070000... (89 %)
Writing at 0x00074000... (92 %)
Writing at 0x00078000... (96 %)
Writing at 0x0007c000... (100 %)
Wrote 752384 bytes (454071 compressed) at 0x00010000 in 40.2 seconds (effective 149.6 kbit/s)...
Hash of data verified.
Compressed 3072 bytes to 128...
Writing at 0x00008000... (100 %)
Wrote 3072 bytes (128 compressed) at 0x00008000 in 0.0 seconds (effective 1561.0 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
```

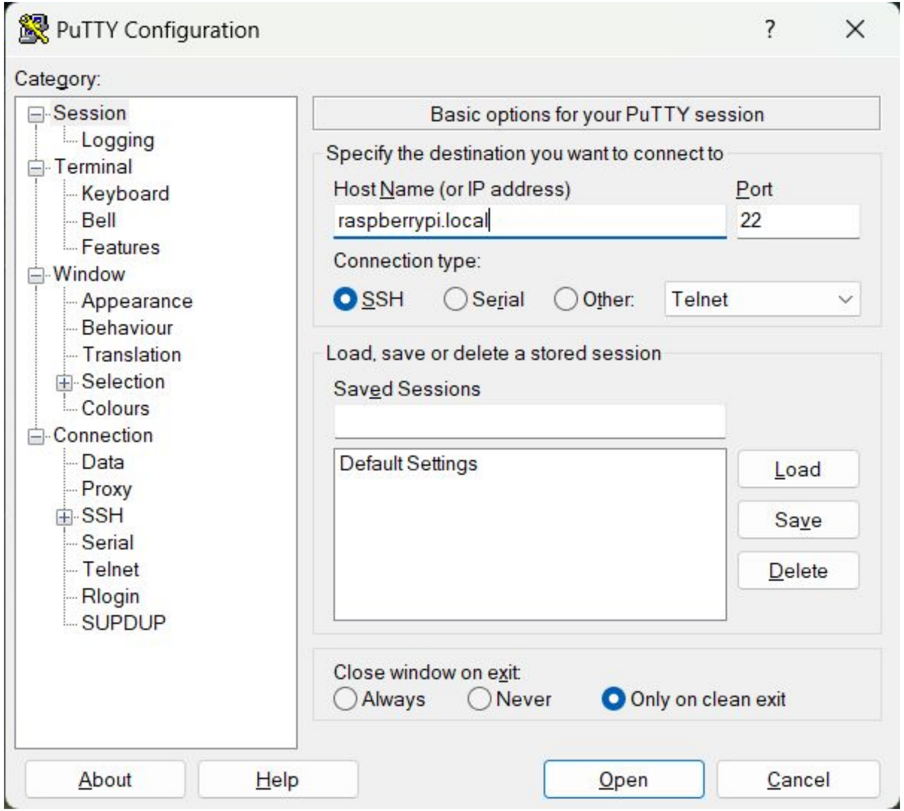
29

DOIT ESP32 DEVKIT V1, 80MHz, 115200, None on /dev/ttyUSB0

Using Raspberry Pi imager to write on disk



Setting PuTTY for configuration



PuTTY Configuration

Category:

Basic options for your PuTTY session

Specify the destination you want to connect to

Host Name (or IP address): raspberrypi.local

Port: 22

Connection type:

● SSH ○ Serial ○ Other: Telnet

Load, save or delete a stored session

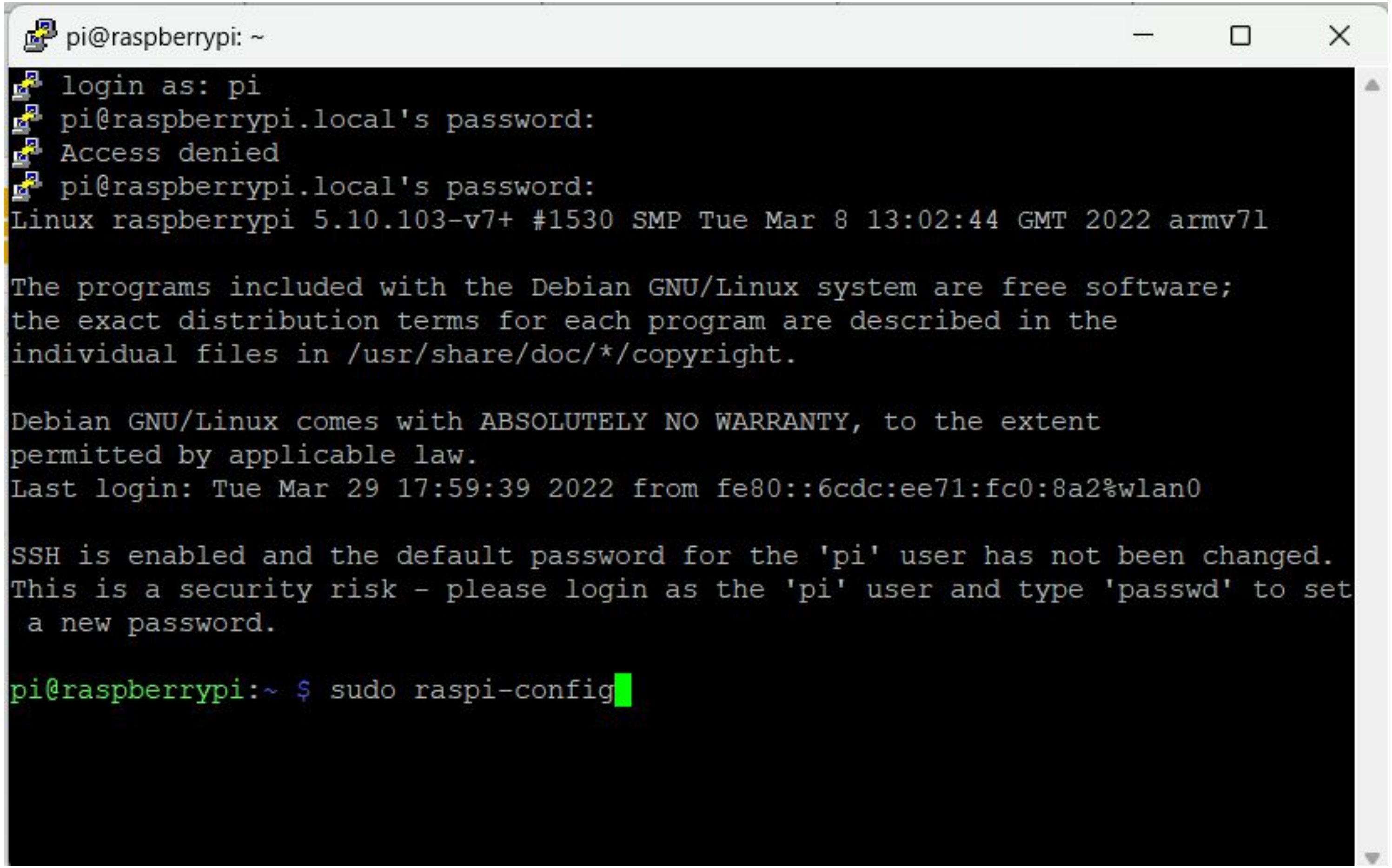
Saved Sessions

Default Settings

Close window on exit:

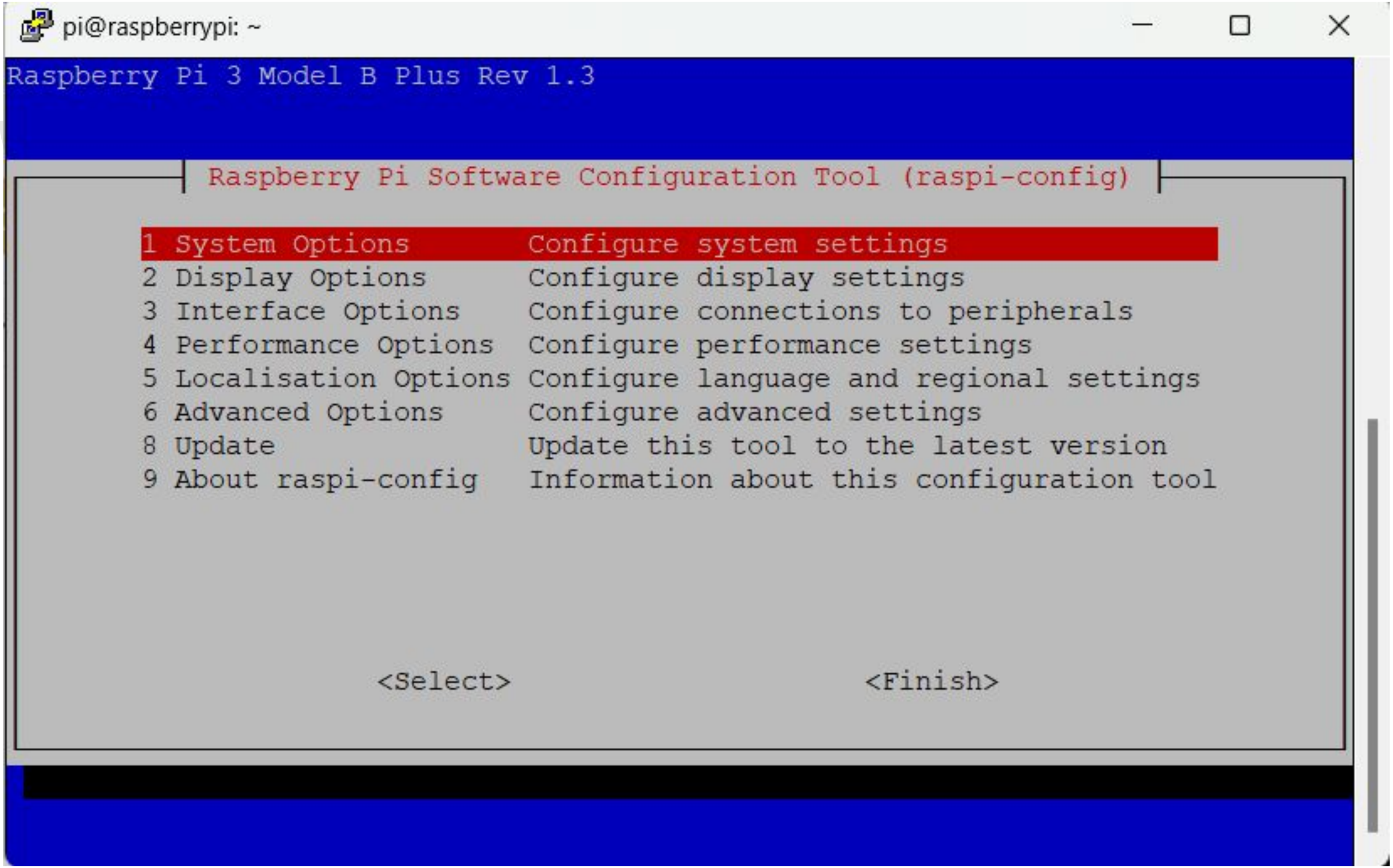
○ Always ○ Never ● Only on clean exit

Configuring raspberry pi



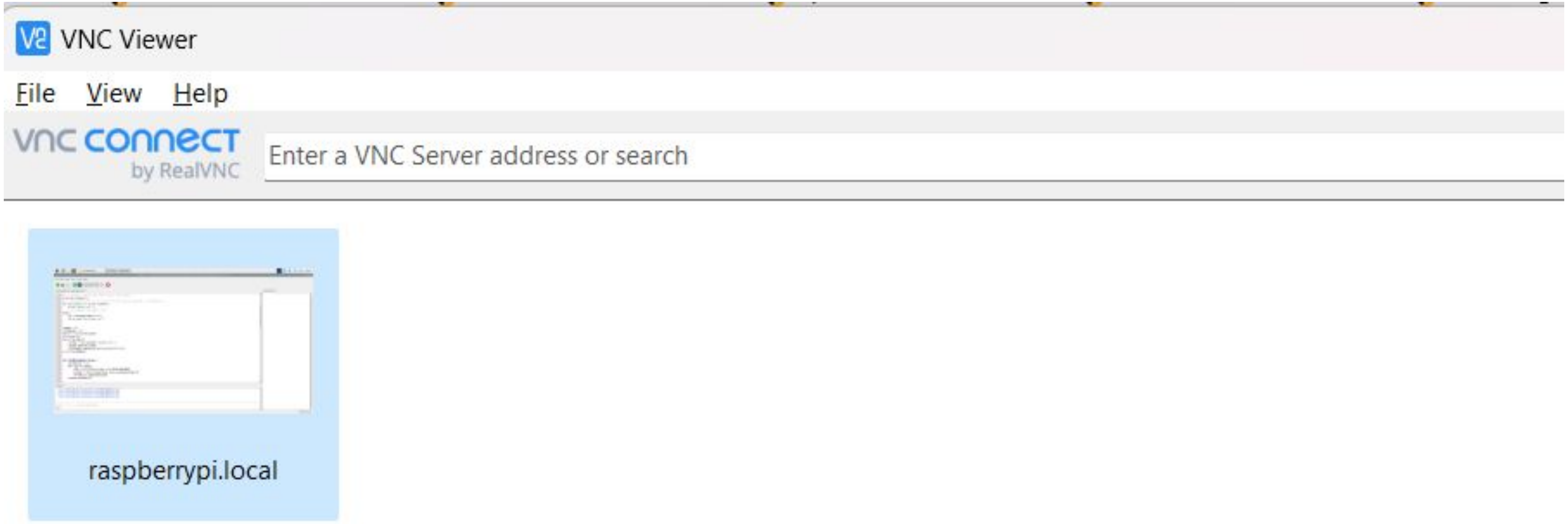
pi@raspberrypi: ~

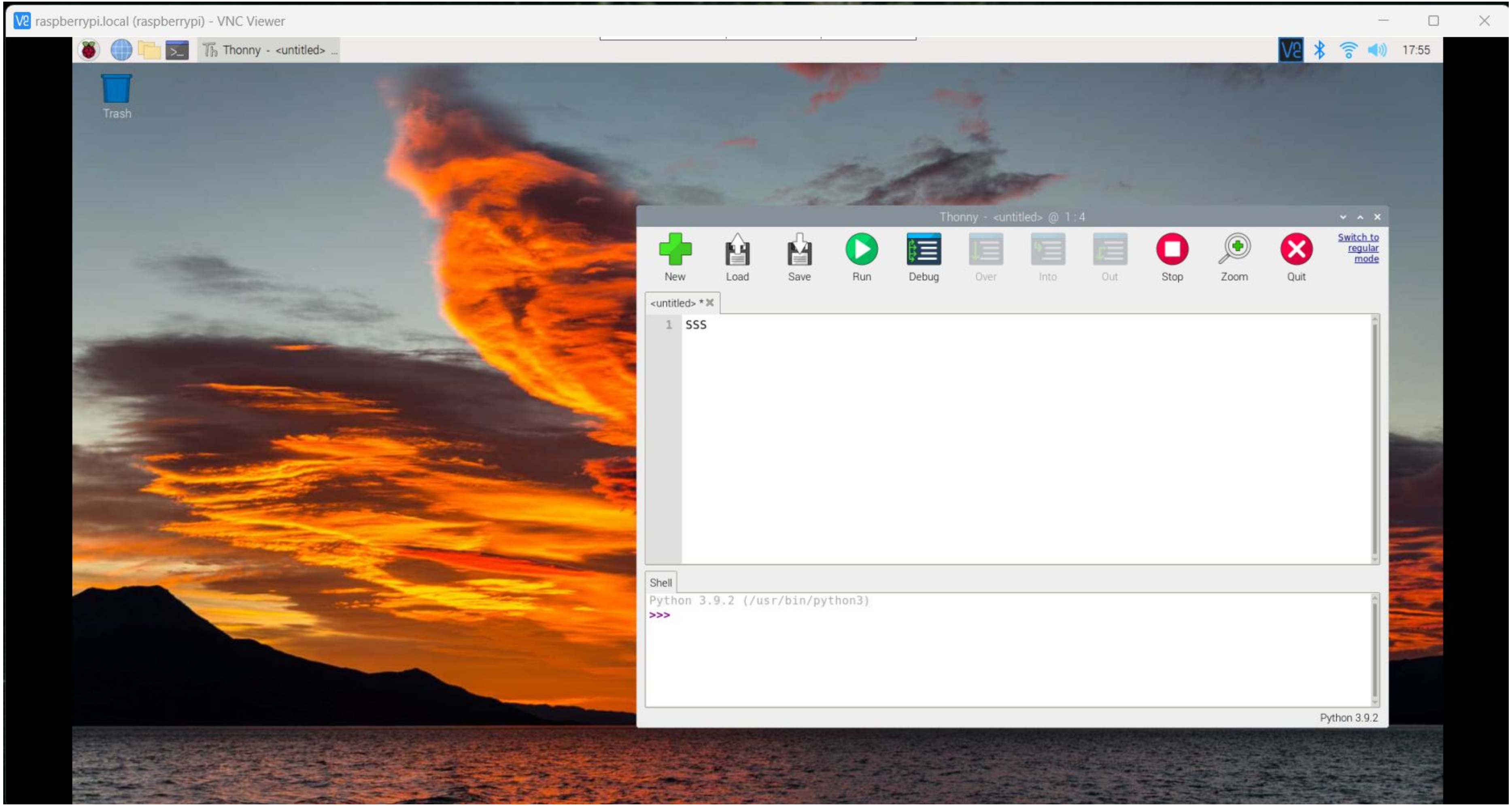
```
login as: pi
pi@raspberrypi.local's password:
Access denied
pi@raspberrypi.local's password:
Linux raspberrypi 5.10.103-v7+ #1530 SMP Tue Mar 8 13:02:44 GMT 2022 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Mar 29 17:59:39 2022 from fe80::6cdc:ee71:fc0:8a2%wlan0

SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set
 a new password.

pi@raspberrypi:~ $ sudo raspi-config
```



```
Raspberry Pi 3 Model B Plus Rev 1.3

            Raspberry Pi Software Configuration Tool (raspi-config)

       1 System Options       Configure system settings
       2 Display Options      Configure display settings
       3 Interface Options    Configure connections to peripherals
       4 Performance Options  Configure performance settings
       5 Localisation Options Configure language and regional settings
       6 Advanced Options     Configure advanced settings
       8 Update               Update this tool to the latest version
       9 About raspi-config   Information about this configuration tool

              <Select>                      <Finish>
```

Using VNC viewer to display Raspi’s screen



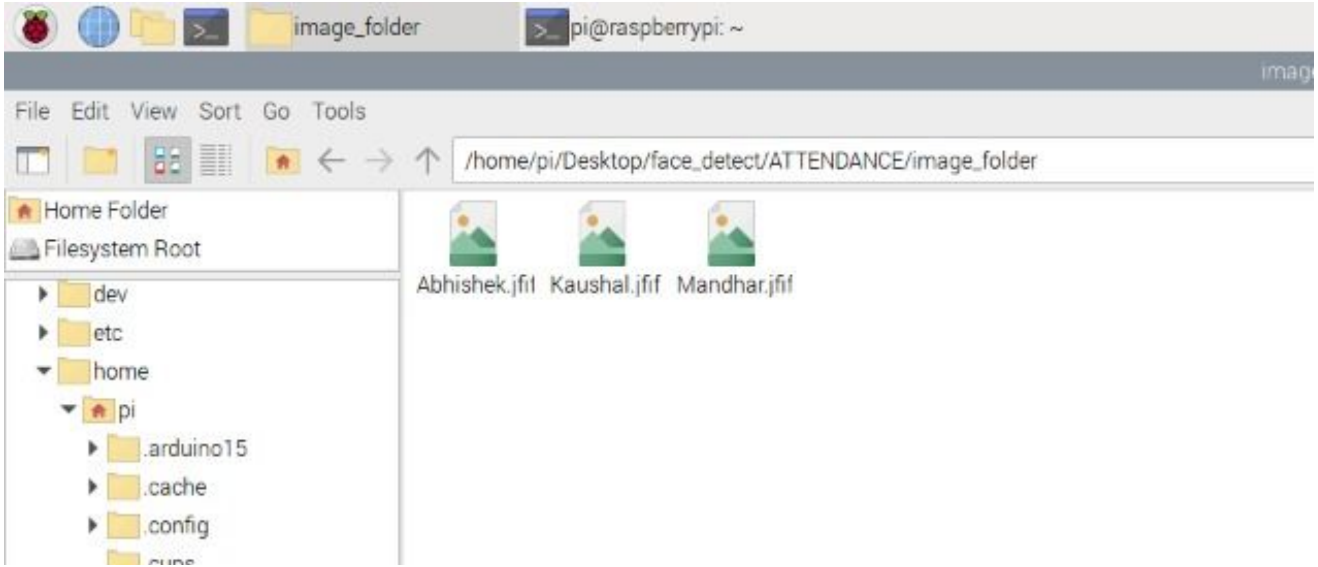


```
sss
```

```
Python 3.9.2 (/usr/bin/python3)
>>>
```

Implementation of Module 2

In this module, we implemented the machine learning part in Raspi and ran a successful facial recognition. We have a folder named images_folder that contains images of all the person allowed to enter. We feed those images to our machine learning algorithm to train it so that it can recognize them whenever they come in front of camera. The screenshots of the same are attached here.

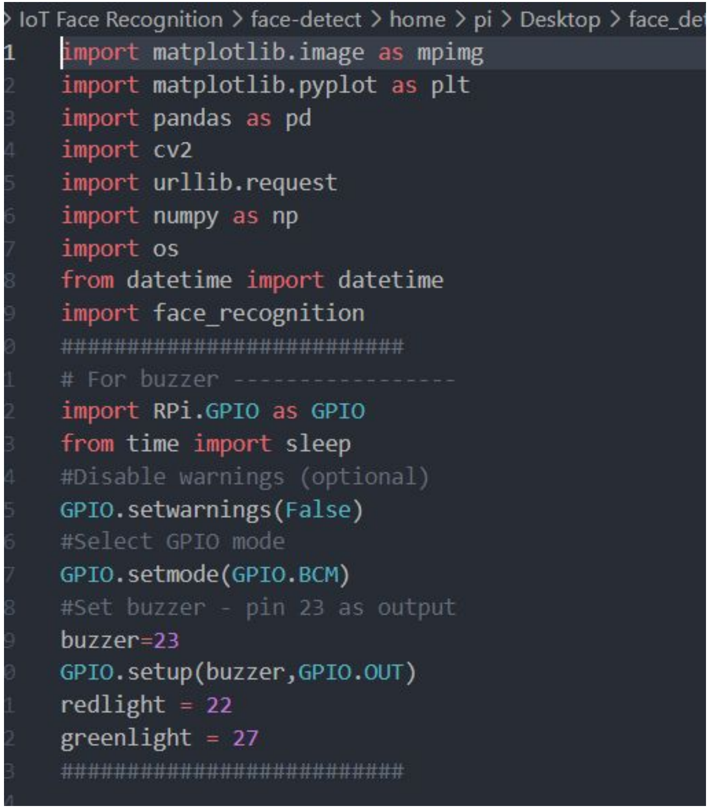


/home/pi/Desktop/face_detect/ATTENDANCE/image_folder

Abhishek.jfif Kaushal.jfif Mandhar.jfif

Implementation of Module 2

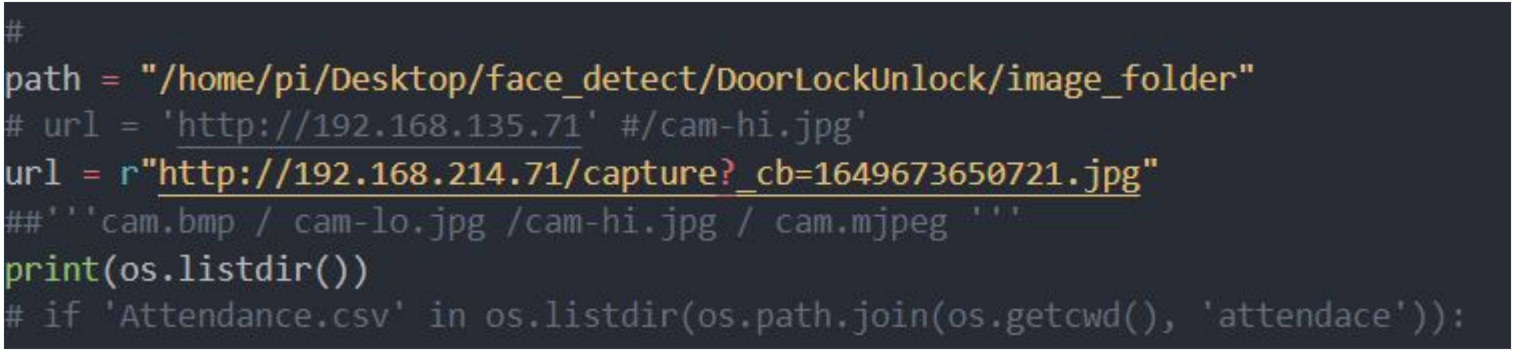
All the important packages are imported including matplotlib.image which is used to handle images and datetime, face_recognition to recognize faces


IoT Face Recognition > face-detect > home > pi > Desktop > face_det

```
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import pandas as pd
import cv2
import urllib.request
import numpy as np
import os
from datetime import datetime
import face_recognition
#########################
# For buzzer -----------------
import RPi.GPIO as GPIO
from time import sleep
#Disable warnings (optional)
GPIO.setwarnings(False)
#Select GPIO mode
GPIO.setmode(GPIO.BCM)
#Set buzzer - pin 23 as output
buzzer=23
GPIO.setup(buzzer,GPIO.OUT)
redlight = 22
greenlight = 27
#########################
```

Implementation of Module 2

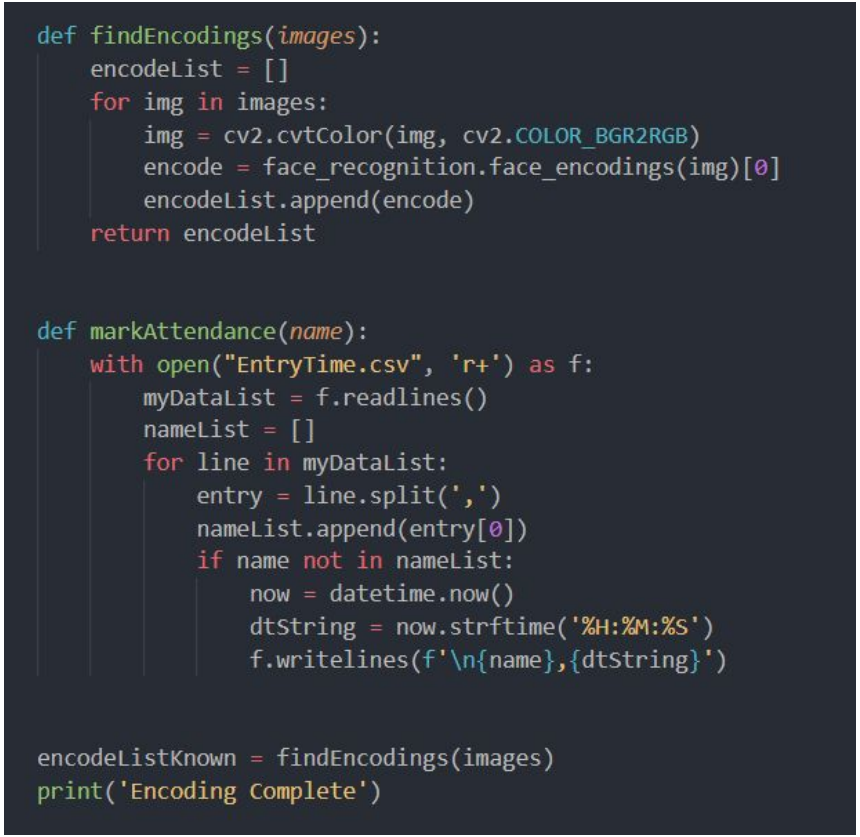
This is where we set the path of the training data set containing images of all the authorized people and the url to the live streaming of esp32 cam. Both are stored in their respective variables.



```
#
path = "/home/pi/Desktop/face_detect/DoorLockUnlock/image_folder"
# url = 'http://192.168.135.71' #/cam-hi.jpg'
url = r"http://192.168.214.71/capture?_cb=1649673650721.jpg"
##'''cam.bmp / cam-lo.jpg /cam-hi.jpg / cam.mjpeg '''
print(os.listdir())
# if 'Attendance.csv' in os.listdir(os.path.join(os.getcwd(), 'attendace')):
```

Implementation of Module 2

This section right here shows two functions, findEncodings which is used to find encodings in a n image and another is markAttendance, which is used to mark the entry time of any person who enters through the door.

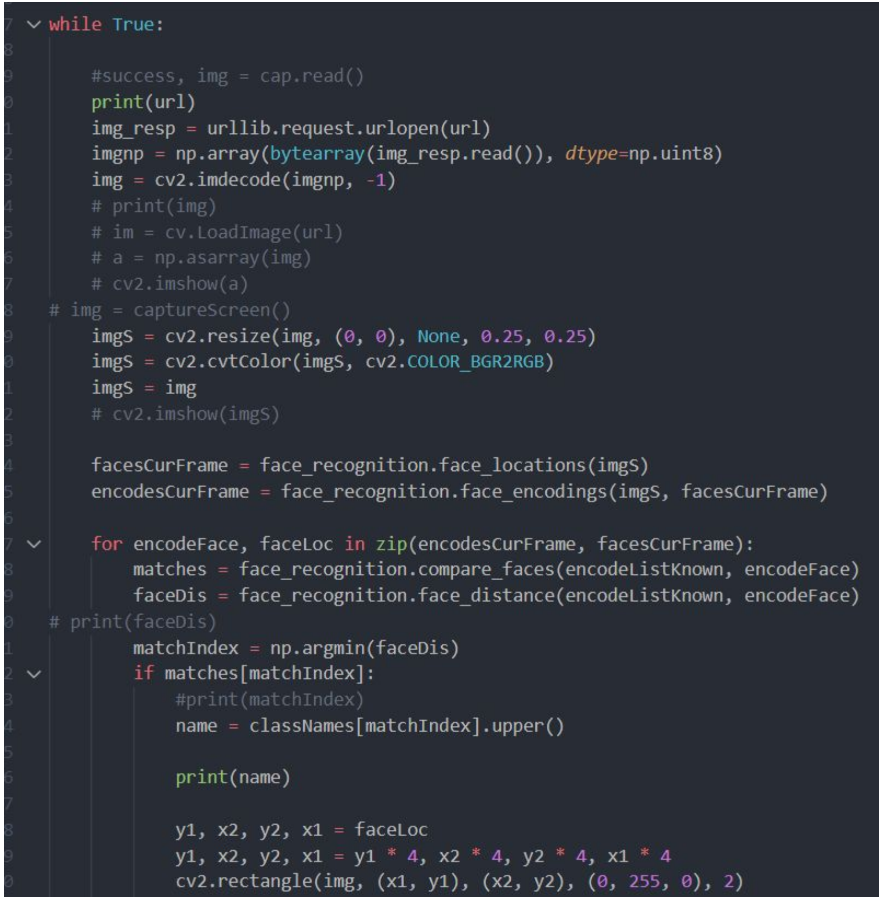


```
def findEncodings(images):
    encodeList = []
    for img in images:
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        encode = face_recognition.face_encodings(img)[0]
        encodeList.append(encode)
    return encodeList


def markAttendance(name):
    with open("EntryTime.csv", 'r+') as f:
        myDataList = f.readlines()
        nameList = []
        for line in myDataList:
            entry = line.split(',')
            nameList.append(entry[0])
            if name not in nameList:
                now = datetime.now()
                dtString = now.strftime('%H:%M:%S')
                f.writelines(f'\n{name},{dtString}')


encodeListKnown = findEncodings(images)
print('Encoding Complete')
```

Implementation of Module 2

Now from this while loop, our main logic starts. This while loop make sure that the esp32 cam runs constantly and in each run, we capture the livestream, check if there is someone in front of the camera, and if there is a person, there are two scenarios created:

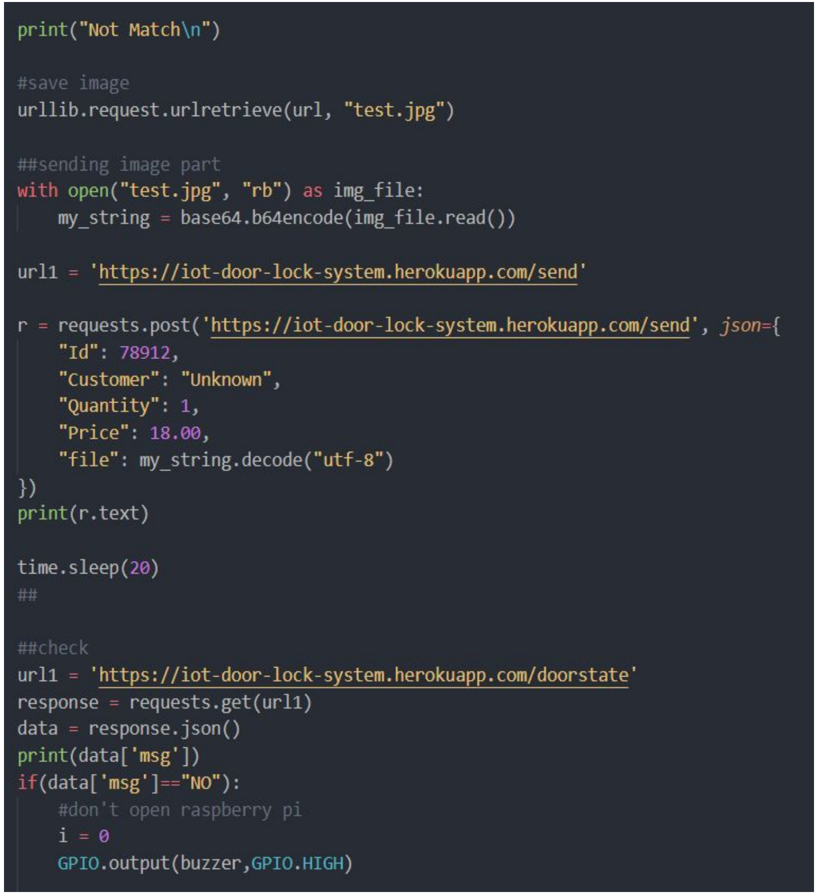
1. **Person is known** : In this case, the door simply unlocks and allows the person to enter
2. **Person is Unknown** : In this scenario, a photo is take of the intruder, and sent to the owner. Now the owner has to allow or disallow the intruder.



```
while True:

    #success, img = cap.read()
    print(url)
    img_resp = urllib.request.urlopen(url)
    imgnp = np.array(bytearray(img_resp.read()), dtype=np.uint8)
    img = cv2.imdecode(imgnp, -1)
    # print(img)
    # im = cv.LoadImage(url)
    # a = np.asarray(img)
    # cv2.imshow(a)
# img = captureScreen()
    imgS = cv2.resize(img, (0, 0), None, 0.25, 0.25)
    imgS = cv2.cvtColor(imgS, cv2.COLOR_BGR2RGB)
    imgS = img
    # cv2.imshow(imgS)

    facesCurFrame = face_recognition.face_locations(imgS)
    encodesCurFrame = face_recognition.face_encodings(imgS, facesCurFrame)

    for encodeFace, faceLoc in zip(encodesCurFrame, facesCurFrame):
        matches = face_recognition.compare_faces(encodeListKnown, encodeFace)
        faceDis = face_recognition.face_distance(encodeListKnown, encodeFace)
# print(faceDis)
        matchIndex = np.argmin(faceDis)
        if matches[matchIndex]:
            #print(matchIndex)
            name = classNames[matchIndex].upper()

            print(name)

            y1, x2, y2, x1 = faceLoc
            y1, x2, y2, x1 = y1 * 4, x2 * 4, y2 * 4, x1 * 4
            cv2.rectangle(img, (x1, y1), (x2, y2), (0, 255, 0), 2)
```

Implementation of Module 3

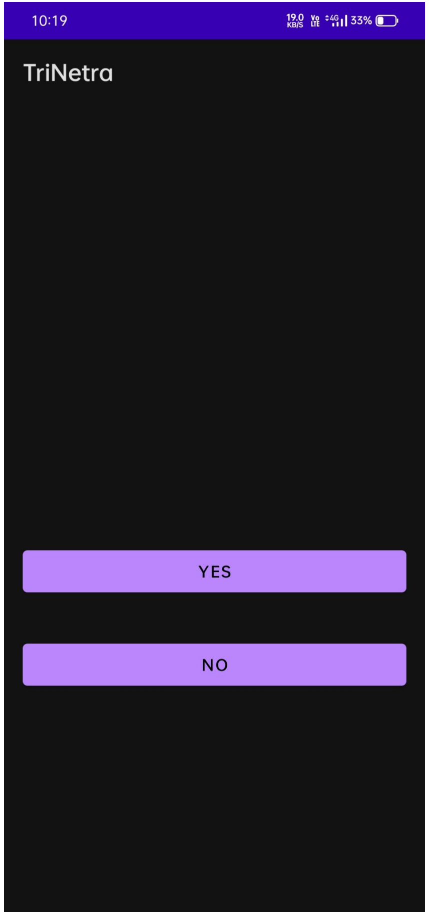
This part is handled here. When the person is not recognized, we take a snap from the livestream, save it as *test.jpg*, and send it our app which is hosted at herokuaap.
Our app has a basic functionality. The image of the intruder is shown and there are two options, YES, or NO. If the owner hits YES, the intruder is given the permission to enter, or else the intruder is disallowed. If the owner does not respond, the intruder is disallowed automatically after 20 seconds.

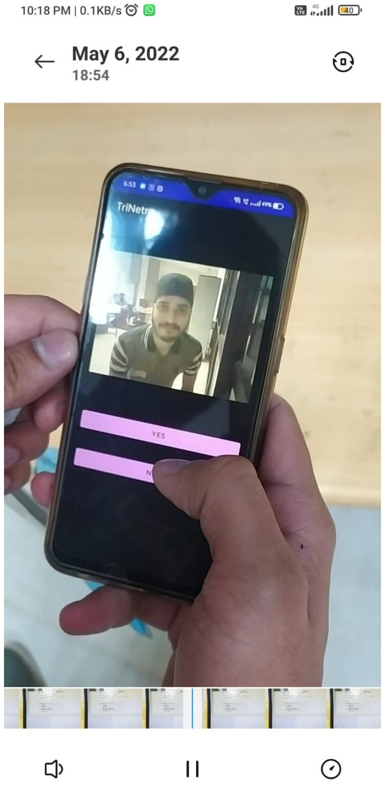


```
print("Not Match\n")

#save image
urllib.request.urlretrieve(url, "test.jpg")

##sending image part
with open("test.jpg", "rb") as img_file:
    my_string = base64.b64encode(img_file.read())

url1 = 'https://iot-door-lock-system.herokuapp.com/send'

r = requests.post('https://iot-door-lock-system.herokuapp.com/send', json={
    "Id": 78912,
    "Customer": "Unknown",
    "Quantity": 1,
    "Price": 18.00,
    "file": my_string.decode("utf-8")
})
print(r.text)

time.sleep(20)
##

##check
url1 = 'https://iot-door-lock-system.herokuapp.com/doorstate'
response = requests.get(url1)
data = response.json()
print(data['msg'])
if(data['msg']=="NO"):
    #don't open raspberry pi
    i = 0
    GPIO.output(buzzer,GPIO.HIGH)
```

Implementation of Module 3

This is the UI of our app. It displays a photo of the intruder and gives two options to the user, Yes(to allow) and No(to disallow).

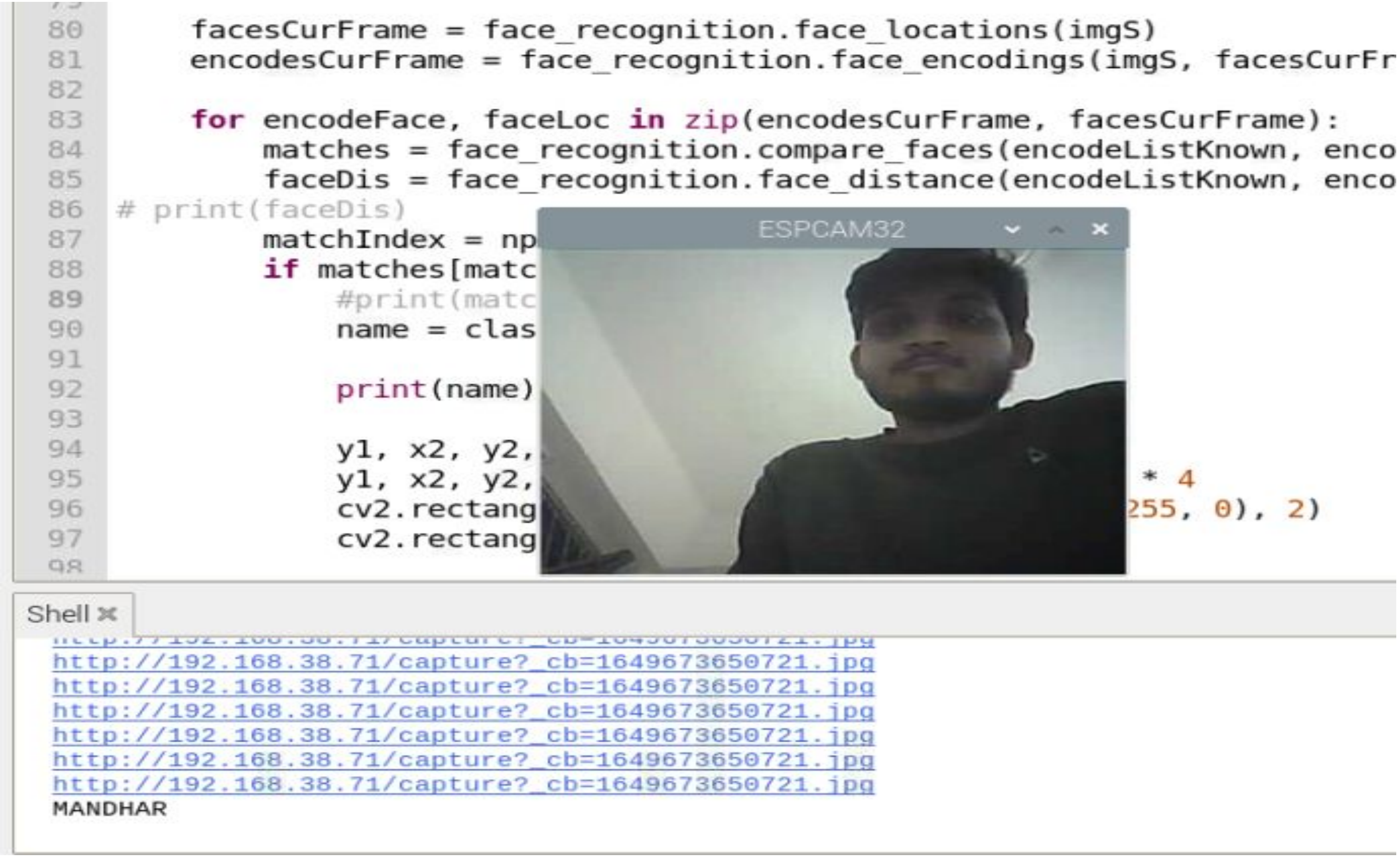




Final Results

Below attached are the screenshots of verdict given by our Machine Learning Model.

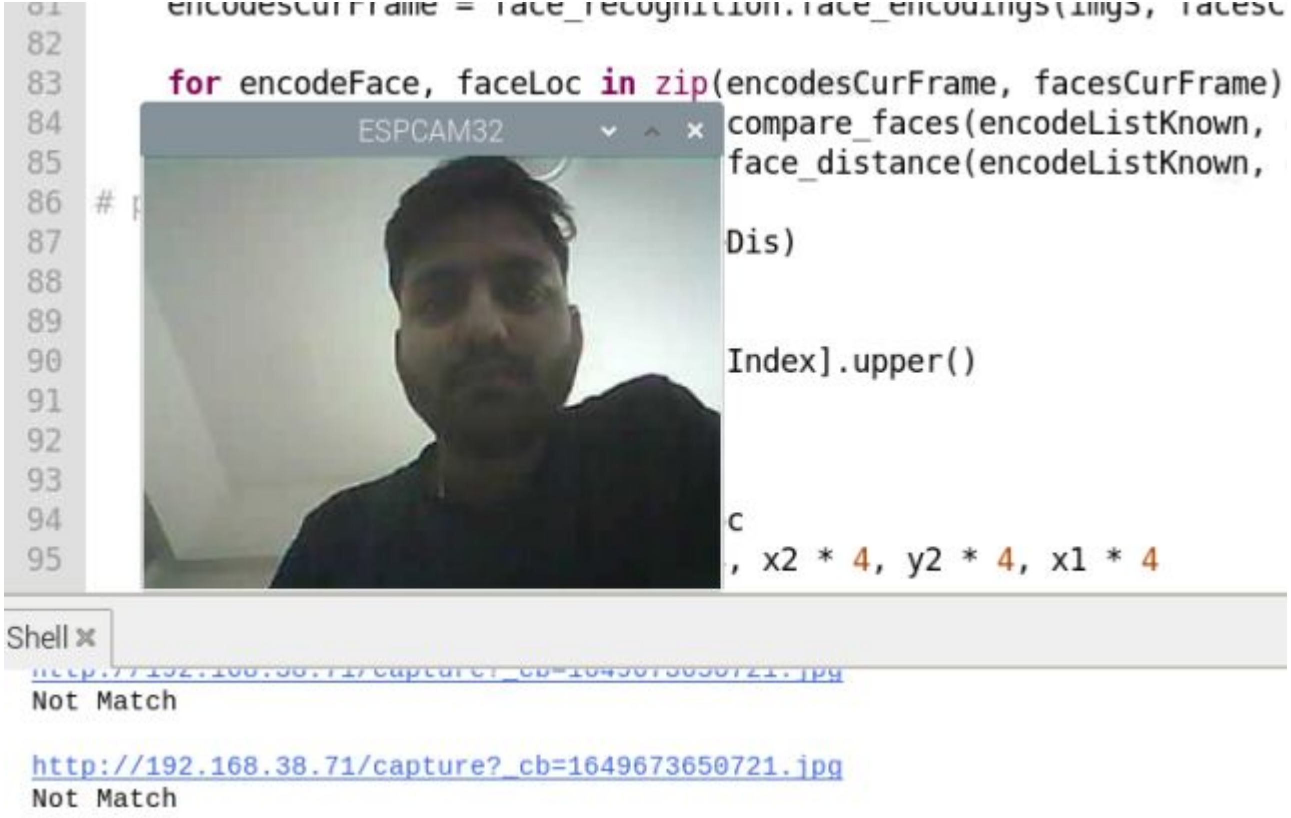
Known person Mandar in front of Cam



```
    facesCurFrame = face_recognition.face_locations(imgS)
    encodesCurFrame = face_recognition.face_encodings(imgS, facesCurFr

    for encodeFace, faceLoc in zip(encodesCurFrame, facesCurFrame):
        matches = face_recognition.compare_faces(encodeListKnown, enco
        faceDis = face_recognition.face_distance(encodeListKnown, enco
# print(faceDis)
        matchIndex = np
        if matches[matc
            #print(matc
            name = clas

            print(name)

            y1, x2, y2,
            y1, x2, y2,                                        * 4
            cv2.rectang                                       255, 0), 2)
            cv2.rectang
```

```
http://192.168.38.71/capture?_cb=1649673650721.jpg
http://192.168.38.71/capture?_cb=1649673650721.jpg
http://192.168.38.71/capture?_cb=1649673650721.jpg
http://192.168.38.71/capture?_cb=1649673650721.jpg
http://192.168.38.71/capture?_cb=1649673650721.jpg
http://192.168.38.71/capture?_cb=1649673650721.jpg
MANDHAR
```

When an unknown person comes in front of camera

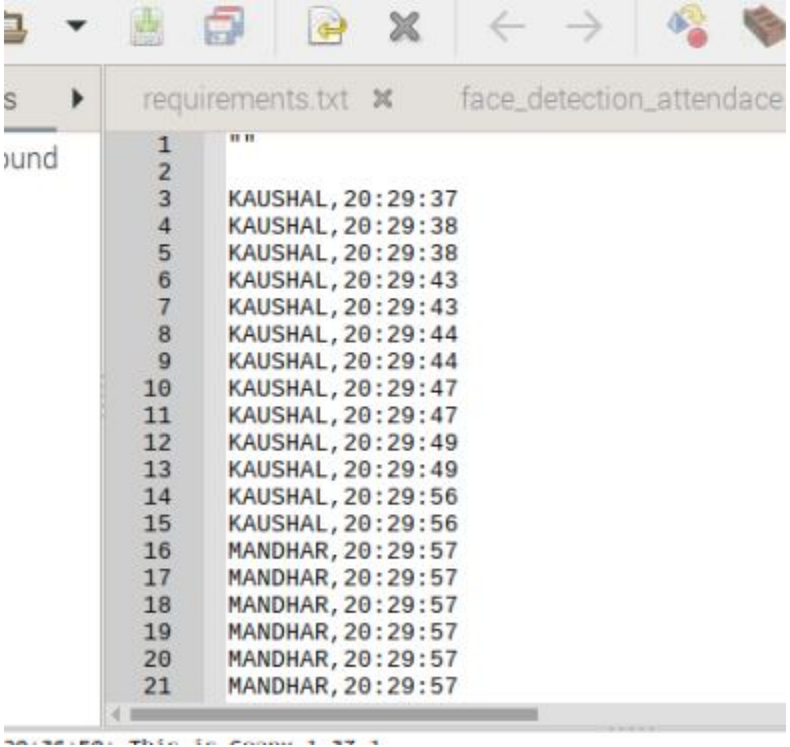


```
    encodesCurFrame = face_recognition.face_encodings(imgS, facesC

    for encodeFace, faceLoc in zip(encodesCurFrame, facesCurFrame)
                                    compare_faces(encodeListKnown,
                                    face_distance(encodeListKnown,
# p
                                    Dis)


                                    Index].upper()




                                    c
                                   , x2 * 4, y2 * 4, x1 * 4
```

Shell

```
http://192.168.38.71/capture?_cb=1649673650721.jpg
Not Match

http://192.168.38.71/capture?_cb=1649673650721.jpg
Not Match
```

EntryTime.csv maintaining all records



```
""

KAUSHAL,20:29:37
KAUSHAL,20:29:38
KAUSHAL,20:29:38
KAUSHAL,20:29:43
KAUSHAL,20:29:43
KAUSHAL,20:29:44
KAUSHAL,20:29:44
KAUSHAL,20:29:47
KAUSHAL,20:29:47
KAUSHAL,20:29:49
KAUSHAL,20:29:49
KAUSHAL,20:29:56
KAUSHAL,20:29:56
MANDHAR,20:29:57
MANDHAR,20:29:57
MANDHAR,20:29:57
MANDHAR,20:29:57
MANDHAR,20:29:57
MANDHAR,20:29:57
```

```
20:36:50: This is Geany 1.37.1
```

Challenges Faced

Since it was an IoT project involving lots of hardware and software components, collaborating the hardware with the required software was a major challenge we faced.

The computation power of Raspi 3b+ was not enough to handle big machine learning models like face recognition. Also the modules required for the program took hours for installation in Raspi.

The connection of esp32 with wired was on breadboard and thus was not tight. So it would disassemble any time. Each time it did, we have to reset the whole ESP32 cam module and start again.

Each time the ESP32 was disconnected from internet, the IP address gets changed and therefore we have to reset it in the code.

Future Scope

This project is in ready-to-use state. We can just put the system on any door and it will work just fine.

In future, there is a huge scope in it,

1. We can always make the app better, giving more functionalities like blocking a particular intruder, emergency calling(in case of breach), sending the notification to multiple users.
2. We can improve the computational efficiency of raspberry which therefore improves the speed and we can also improve the face recognition algorithm for better accuracy.

README.MD
ESP32 Arduino code