*A project report on*

**USE OF FACE RECOGNITION FOR SMART HOME**

*Submitted in partial fulfillment for the award of the degree of*

**Bachelor of Technology in Information Technology**

*by*

**YOGESH CHOUDHARY(14BIT0160)**



**SCHOOL OF INFORMATION TECHNOLOGY AND ENGINEERING**

April, 2018

**DECLARATION**

I hereby declare that the project report entitled “USE OF FACE RECOGNITION OFR SMART HOME” submitted by me, for the award of the degree of Bachelor of Technology to VIT is a record of bonafide work carried out by me under the supervision of Guide Jayalakshmi P.

I further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Vellore

Date: **Signature of the Candidate**

**CERTIFICATE**

This is to certify that the Project Report entitled “USE OF FACE RECOGNITION OFR SMART HOME” submitted by YOGESH CHOUDHARY (14BIT0160) SCHOOL OF INFORMATION TECHNOLOGY AND ENGINEERING VIT, Vellore for the award of the degree of Bachelor of Technology is a record of bonafide work carried out by him under my supervision.

The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university. The Project report fulfills the requirements and regulations of VIT and in my opinion meets the necessary standards for submission.

**Signature of the Guide Signature of the Hod**

**Internal Examiner External Examiner**

**ABSTRACT**

Nowadays people use keys, security cards, password or pattern to open the door. The aim of this paper is to help people for improvement of the door security of sensitive locations by using face detection and recognition. Face is a complex multidimensional structure and needs enhanced computing techniques for detection and recognition. This project comprises of three subsystems: namely face detection, face recognition and automatic door access control. Image acquisition is the process of capturing an image. The face detection and recognition is implemented by using the . Face Recognition based on dlib, opencv and python.The door will open automatically for the authenticated person due to the command of the microcontroller. Since it reduces the dimensions of face images without losing important features, facial images for many persons can be stored in the database. Although many training images are used, computational efficiency cannot be decreased significantly. Therefore , face recognition using this can be more useful for door security system than other face recognition schemes.

**ACKNOWLEDGEMENT**

It is my pleasure to express with deep sense of gratitude to Jayalakshmi P, Assistant Professor(Senior), School of Information Technology And Engineering, Vellore Institute of Technology, for her constant guidance, continual encouragement, understanding; more than all, she taught me patience in my endeavour. My association with her is not confined to academics only, but it is a great opportunity on my part of work with an intellectual and expert in the field of IOT.

I would like to express my gratitude to Dr.G Viswanathan(Chancellor), Sankar Viswanathan(VP), Sekar Viswanathan(VP), G V Selvam(VP), Dr.Anand Samuel(VC), Dr.S Narayan(Pro-VC) and Dr.Ashwini Kumar Ch(Dean), School of Information Technology and Engineering, for providing with an environment to work in and for his inspiration during the tenure of the course.

In jubilant mood I express ingeniously my whole-hearted thanks to M Dinakaran, HOD, B.Tech, School of Information Technology and Engineering, all teaching staff and members working as limbs of our university for their not-self-centered enthusiasm coupled with timely encouragements showered on me with zeal, which prompted the acquirement of the requisite knowledge to finalize my course study successfully. I would like to thank my parents for their support.

It is indeed a pleasure to thank my friends who persuaded and encouraged me to take up and complete this task. At last but not least, I express my gratitude and appreciation to all those who have helped me directly or indirectly toward the successful completion of this project.

Place: Vellore

Date: **YOGESH CHOUDHARY**

**CONTENTS**

**CONTENTS** ...............................................................................................

**LIST OF FIGURES** ...................................................................................

**CHAPTER 1**

**INTRODUCTION**......................................................................................................9

1.1 MOTIVATION

1.2 CONTRIBUTION SUMMARY

1.3PROJECT ORIENTATION

1.4 THORY

**CHAPTER 2**

**BACK GROUND**.......................................................................................................18

2.1 SYSTEM ARCHITECTURE

2.2 CHOICE OF ARDUINO

2.3 SYSTEM CONNECTION WITH BLUETOOTH

2.4CODE SNIPPET

2.5 ALGORITHMS

**CHAPTER 3**

**METHODS AND IMPLEMENTATION DETAIL**..............................................30

3.1 INTRODUCTION

3.2 PROCESS EXECUTION

3.3 DOOR LOCK

3.4 CODING FACE RECOGNITION WITH OPENCV STEP BY STEP

**CHAPTER 4**

**EXPERIMENTAL RESULTS**................................................................................50

**CHAPTER 5**

**CONCLUSION**........................................................................................................58

**LIST OF FIGURES**

1.3 TEMPLATE MATCHING.........................................................10

1.4 FACE DETECTION...................................................................12

1.5 FACIAL LANDMARKS............................................................15

2.1 SYSTEM DESIGN......................................................................20

2.2 WEBCAMERA............................................................................22

2.3 HARDWARE IAMGE................................................................ 22

2.4 FISHER FACE.............................................................................30

3.1 SETUP..........................................................................................35

3.2 IMAGE RECONGITION WITH SINGLE PERSON.................40

3.3 IMAGE RECONGITION WITH MULTIPLE PERSON............44

4.1 CODE RUN ..................................................................................45

4.2 CODE RUN2.................................................................................46

4.3 BLUETOOTH CONNECTIVITY................................................ 48

4.4 RED LED SIGNAL.......................................................................48

4.5 GREENLED SIGNAL...................................................................49

**Chapter 1**

**INTRODUCTION**

What is face recognition? Or what is recognition? When you look at an apple fruit, your mind immediately tells you that this is an apple fruit. This process, your mind telling you that this is an apple fruit is recognition in simple words. So what is face recognition then? I am sure you have guessed it right. When you look at your friend walking down the street or a picture of him, you recognize that he is your friend Paulo. Interestingly when you look at your friend or a picture of him you look at his face first before looking at anything else. Ever wondered why you do that? This is so that you can recognize him by looking at his face. Well, this is you doing face recognition.

But the real question is how does face recognition works? It is quite simple and intuitive. Take a real life example, when you meet someone first time in your life you don't recognize him, right? While he talks or shakes hands with you, you look at his face, eyes, nose, mouth, color and overall look. This is your mind learning or training for the face recognition of that person by gathering face data. Then he tells you that his name is Paulo. At this point your mind knows that the face data it just learned belongs to Paulo. Now your mind is trained and ready to do face recognition on Paulo's face. Next time when you will see Paulo or his face in a picture you will immediately recognize him. This is how face recognition work. The more you will meet Paulo, the more data your mind will collect about Paulo and especially his face and the better you will become at recognizing him.

Now the next question is how to code face recognition with OpenCV, after all this is the only reason why you are reading this article, right? OK then. You might say that our mind can do these things easily but to actually code them into a computer is difficult? Don't worry, it is not. Thanks to OpenCV, coding face recognition is as easier as it feels. The coding steps for face recognition are same as we discussed it in real life example above.

* **Training Data Gathering:** Gather face data (face images in this case) of the persons you want to recognize
* **Training of Recognizer:** Feed that face data (and respective names of each face) to the face recognizer so that it can learn.
* **Recognition:** Feed new faces of the persons and see if the face recognizer you just trained recognizes them.

OpenCV comes equipped with built in face recognizer, all you have to do is feed it the face data. It's that simple and this how it will look once we are done coding it.

* 1. MOTIVATION

Understanding the necessity of security system this idea has been proposed. This system is structured in a way so that maximum security is ensured. The door locks now used in most places can be easily opened with master key but this system consists of special features which require conditions to open a door lock. A step towards technology making life simpler and secure is what brought up the motivation to build it. The System consists of two sections where each has a criteria or condition to fulfill. As these days everyone can afford camera so this system won‟t be difficult to use. With a help of a simple setup this system can be executed.

* 1. CONTRIBUTION SUMMARY

The summary of the main contributions is as follows:

Arduino Uno has been used with Bluetooth module to establish the connection with Bluetooth of Android phone.

An electric door lock that requires 12V to be operated.

The face detection program is done in OpenCV using Python

By using different algorithm the whole process become faster in execution time as this algorithm makes sure that each face is trained separately giving a complete accuracy.

* 1. PROJECT ORIENTATION

The rest of the PROJECT REPORT is organized as follows:

Chapter 02 includes the necessary background information of the system explaining Bluetooth connectivity and face detection method.

Chapter 03 presents the methods and implementation details for the system.

Chapter 04 demonstrates the experimental results and comparison.

Chapter 05 concludes the REPORT and states the future research directions.

* 1. THEORY

As can be assumed, detecting a face is simpler than recognizing a face of a specific person. In order to be able to determine that a certain picture contains a face (or several) we need to be able to define the general structure of a face. Luckily human faces do not greatly differ from each other; we all have noses, eyes, foreheads, chins and mouths; and all of these compose the general structure of a face.

Consider the following 5 figures:



Each of these figures represents a general feature of a human face. Combining all the features together we, indeed, receive something that resembles a face.



By determining if each of these features is similar to some part of our picture, we can conclude if the picture contains a face or not. Notice that this does not have to be an accurate match; we just need to know if, roughly, each of these features corresponds to some part of the image. The technique used for this purpose is [Template Matching](http://en.wikipedia.org/wiki/Template_matching).

By gathering statistics about which such features compose faces and how, we can train our algorithm to use the right features in the right positions; and thus detect faces.

Let's see an example. See in the figures below how the above features can be used to detect a face (namely, the face of President Barack Obama).



FIG- 1.41 TEMPLATE MATCHING

In order for this process be quick, we design it in such a way that we first check the coarse features which represent the coarse structure of a face; and only if these features match, we continue to the next iteration and use finer features. In each such iteration we can quickly reject areas of the picture which do not match a face, and keep checking those which we are not sure about. In every iteration we increase the certainty that the checked area is indeed a face, until finally we stop and make our determination.

In other words, rather than determining if the image does contain a face, we can more quickly determine if the image does not contain a face; because eliminations can be done quickly, while acceptance of faces will require more time. We call such a process a cascading process.

1.41 FACIAL LANDMARKS

# Facial landmarks with dlib, OpenCV, and Python

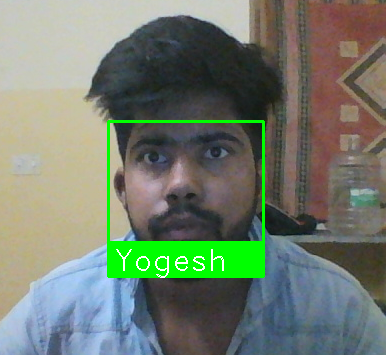


FIG 1.42 FACE DETECTION

we are going to use dlib and OpenCV to detect **facial landmarks** in an image.

Facial landmarks are used to localize and represent salient regions of the face, such as:

Eyes

Eyebrows

Nose

Mouth

Jawline

Facial landmarks have been successfully applied to face alignment, head pose estimation, face swapping, blink detection and much more.

In THIS we’ll be focusing on the **basics of facial landmarks**, including:

Exactly what facial landmarks are and how they work. How to detect and extract facial landmarks from an image using dlib, OpenCV, and Python.

### What are facial landmarks?

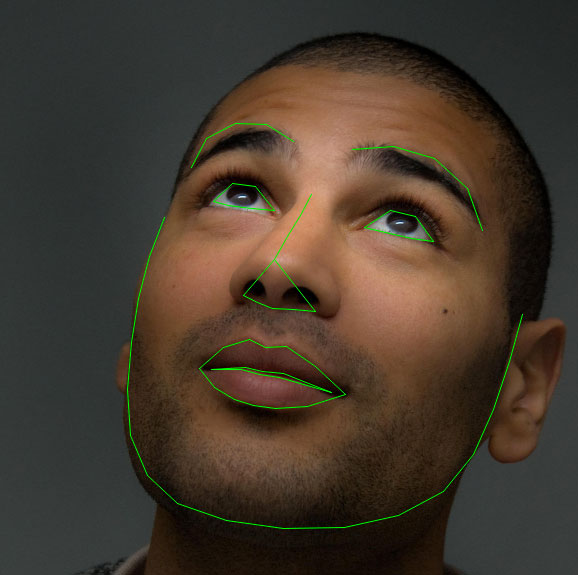


FIG 1.43 FACIAL LANDMARKS

Detecting facial landmarks is a *subset* of the *shape prediction* problem. Given an input image (and normally an ROI that specifies the object of interest), a shape predictor attempts to localize key points of interest along the shape.

In the context of facial landmarks, our goal is detect important facial structures on the face using shape prediction methods. Face detection (Step #1) can be achieved in a number of ways.We could use OpenCV’s built-in Haar cascades.We might apply a pre-trained [HOG + Linear SVM object detector](https://www.pyimagesearch.com/2014/11/10/histogram-oriented-gradients-object-detection/) specifically for the task of face detection.

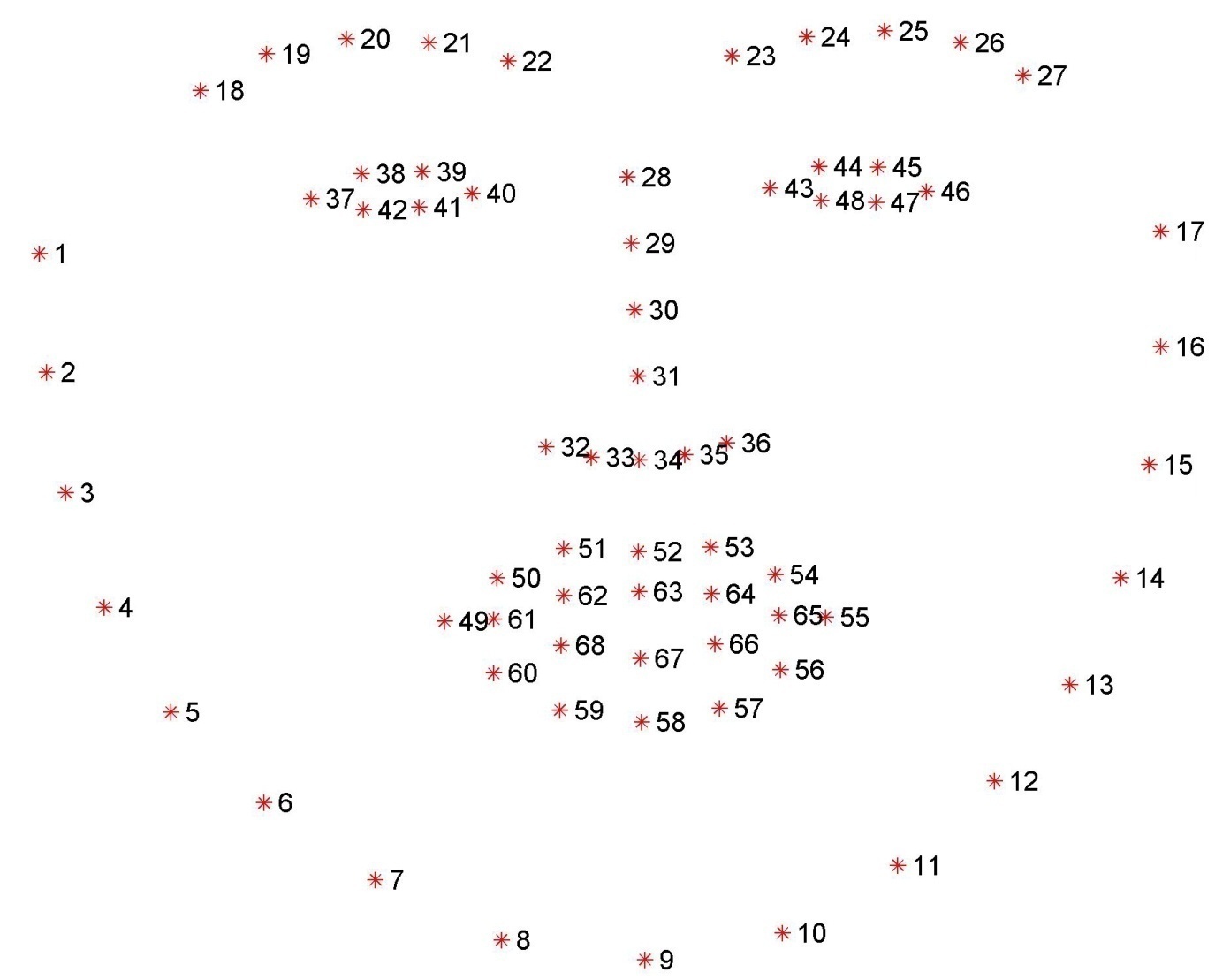
Or we might even use deep learning-based algorithms for face localization. In either case, the actual algorithm used to detect the face in the image doesn’t matter. Instead, what’s important is that through some method we obtain the face bounding box (i.e., the *(x, y)*-coordinates of the face in the image). Given the face region we can then apply **Step #2: detecting key facial structures in the face region.**

There are a variety of facial landmark detectors, but all methods essentially try to localize and label the following facial regions.The facial landmark detector included in the dlib library is an implementation of the [*One Millisecond Face Alignment with an Ensemble of Regression Trees*](https://pdfs.semanticscholar.org/d78b/6a5b0dcaa81b1faea5fb0000045a62513567.pdf) paper by Kazemi and Sullivan (2014). A training set of labeled facial landmarks on an image. These images are *manually labeled*, specifying **specific** *(x, y)*-coordinates of regions surrounding each facial structure. *Priors*, of more specifically, the *probability on distance* between pairs of input pixels.Given this training data, an ensemble of regression trees are trained to estimate the facial landmark positions directly from the *pixel intensities themselves* For more information and details on this specific technique, be sure to read the paper by Kazemi and Sullivan linked to above, along with the [official dlib announcement](http://blog.dlib.net/2014/08/real-time-face-pose-estimation.html).

### Understanding dlib’s facial landmark detector

The pre-trained facial landmark detector inside the dlib library is used to estimate the location of ***68 (x, y)-coordinates*** that map to facial structures on the face.

The indexes of the 68 coordinates can be visualized on the image below:



These annotations are part of the 68 point  which the dlib facial landmark predictor was trained on. It’s important to note that other flavors of facial landmark detectors exist, including the 194 point model that can be trained on the [HELEN dataset](http://www.ifp.illinois.edu/~vuongle2/helen/). Regardless of which dataset is used, the same dlib framework can be leveraged to train a shape predictor on the input training data — this is useful if you would like to train facial landmark detectors or custom shape predictors of your own.

Detecting facial landmarks in an image is a two step process: First we must localize a face(s) in an image. This can be accomplished using a number of different techniques, but normally involve either Haar cascades or HOG + Linear SVM detectors Apply the shape predictor, specifically a facial landmark detector, to obtain the *(x, y)*-coordinates of the face regions in the face ROI. Given these facial landmarks we can apply a number of computer vision techniques, including:

1. Face part extraction (i.e., nose, eyes, mouth, jawline, etc.)
2. Facial alignment
3. Head pose estimation
4. Face swapping
5. Blink detection
6. *…and much more!*

**Chapter 2**

**BACKGROUND**

The security system consists of two different parts so their architectures are separate and different. The Bluetooth Connection consists most of the hardware section so it has the architecture set properly.

2.1 SYSTEM ARCHITECTURE

The System architecture of the external webcam, Bluetooth , arduino ,dc motor and along with laptop for Opencv code implementation using python.

Here as it is shown face is detected using the webcam and then , using webcam frame by frame stream is detected and it detect all faces in every frame, it creates face encoding of each face then it matches incoming image encoding with user image encoding.this finally shows the result which is sent to arduino via Bluetooth and finally it shows servo output,which tells the dc motor enabled gate which acc to face recognition opens or remains closed.

**Arduino Uno** is a microcontroller board based on the ATmega328P (datasheet). It has 14digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz quartz crystal, a USB connection, a power jack, an ICSP header and a reset button.

**Servomotor** is a rotary actuator or linear actuator that allows for precise control of angular orlinear position, velocity and acceleration. It consists of a suitable motor coupled to a sensor for position feedback.

**Bluetooth module HC-05** module is an easy to use Bluetooth SPP (Serial PortProtocol) module, designed for transparent wireless serial connection setup. Serial port Bluetooth module is fully qualified Bluetooth V2.0+EDR (Enhanced Data Rate) 3Mbps Modulation with complete 2.4GHz radio transceiver and baseband.

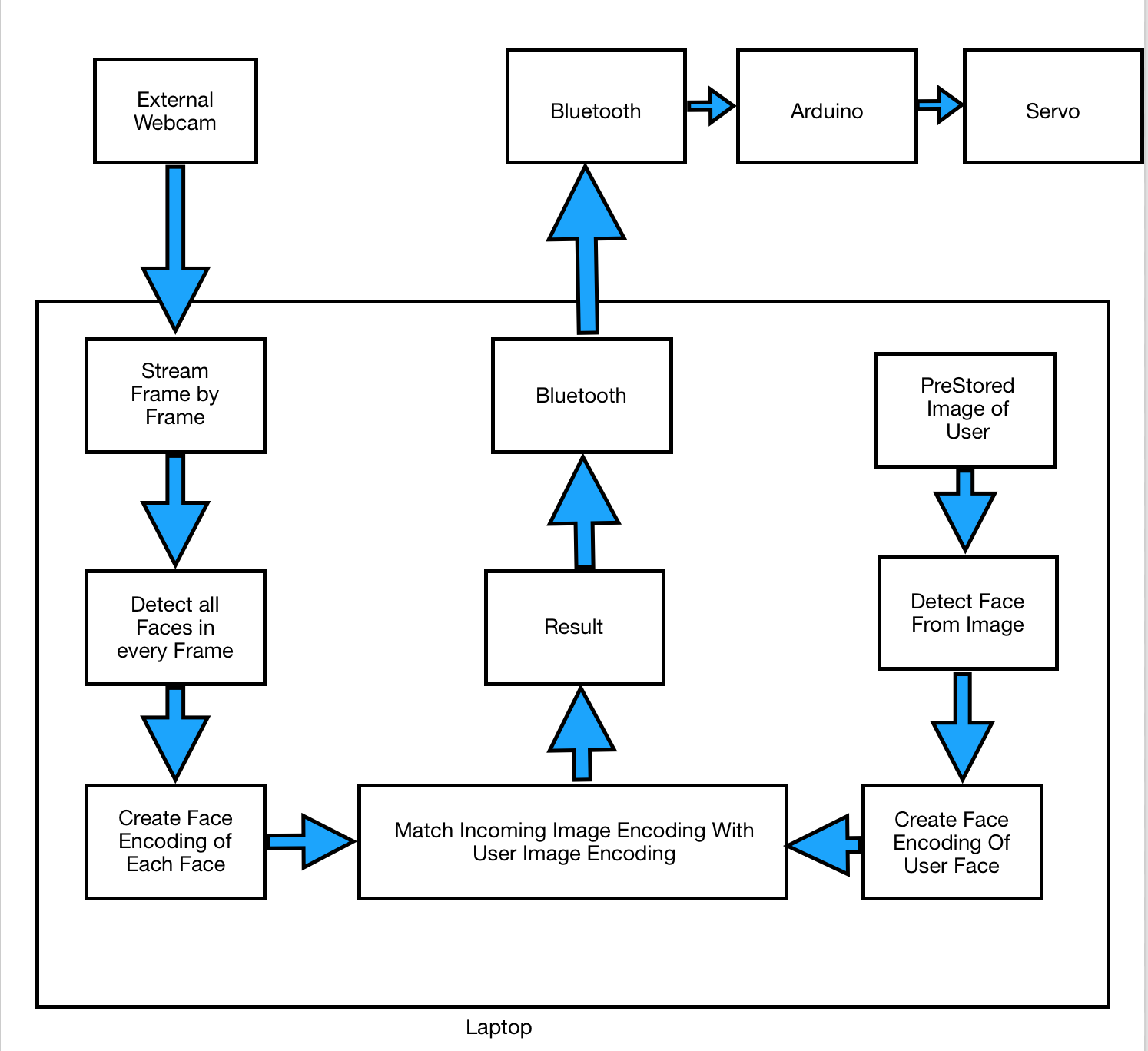
****

FIG- 2.11 SYSTEM DESIGN

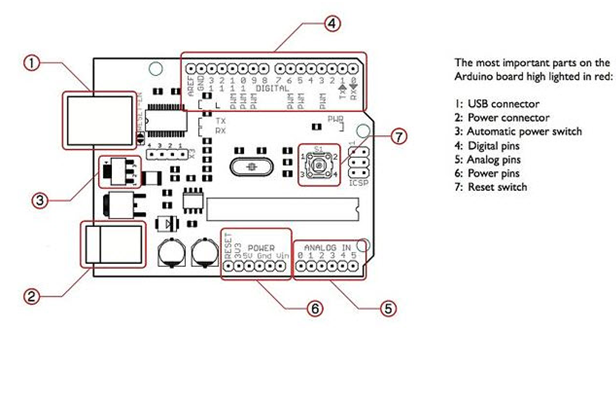


FIG 2.12 ARDUINO MODULE

2.2 CHOICE OF ARDUINO

Arduino Uno has been chosen for this system as Arduino uno supports the connection of Bluetooth whereas Arduino Mega despite of having more program space does not allow establishing Bluetooth Connection via Bluetooth Module. Comparison between Arduino Uno and Arduino Mega is shown in the Table 2.1 below. The choice of choosing an arduino is a major decision in establishing a connection.

**Table 2.2:** Comparison between Arduino Uno and Arduino Mega

**COMPARISON**

|  |  |
| --- | --- |
| **Arduino Uno** | **Arduino Mega** |
| Allows Bluetooth Connection | Does Not Allow Bluetooth |
|  | Connection |
| Runs most programs | Does not Run most programs |
| Very compact | Not very compact |
| Support Wi-Fi | Do Not Support Wi-Fi |
| Space 32KB Program Space | Space 256 KB Program Space |

2.3 SYSTEM CONNECTION WITH BLUETOOTH

The former idea was to establish the connection with the system with a Wi-Fi but viewing pros and cons of both Bluetooth and Wi-Fi, it has been later decided that the connectivity of the system with that of the device will be done using Bluetooth Module. This is because the major con of Wi-Fi is that it needs internet connectivity. In most cases Bluetooth acts better that Wi-Fi in terms of connection. The Connectivity speed although is much greater in Wi-Fi but there can be loss of connection due to absence of internet, so Bluetooth has been choosed in the system connection.

2.4 CODE SNIPPET

2.4.1 HARDWARE CODE

#include<SoftwareSerial.h>

#include<Servo.h>

Servo gate;

SoftwareSerial port(8,9);

int rled = 2;

int gled = 3;

char ch = 0;

void setup() {

gate.attach(10);

port.begin(9600);

pinMode(rled,OUTPUT);

pinMode(gled,OUTPUT);

gate.write(0);

}

voidloop() { while(port.available()) {

ch = port.read();

port.write("Smart Door");

if(ch == 'o') {

digitalWrite(rled,LOW);

digitalWrite(gled,HIGH);

gate.write(90);

delay(5000);

digitalWrite(rled,HIGH);

digitalWrite(gled,LOW);

gate.write(0);

}

}

digitalWrite(rled,HIGH);



FIG 2.4- HARDWARE IMAGE

2.4.2 FACIAL DETECTION, RECOGNITION CODE

FACE ID GUI

from tkinter import \*

import createEncoding

import runFaceid

root = Tk()

def run():

runFaceid.runCamera()

def updatedb():

createEncoding.update()

photo = PhotoImage(file="UIBG.png")

label0 = Label(root,image=photo)

#label1 = Label(root,text="Update Changes in Database")

button1 = Button(root,text="Update Database",fg="red", command=updatedb)

button2 = Button(root,text="Run Camera",bg="black",fg="white",command=run)

label0.grid(row=0,column=0)

button1.grid(row=0,column=0)

button2.grid(row=1,column=0)

root.mainloop()

CREATE FACE ENCODING

import face\_recognition

import os

import pickle

def update():

photos = []

names = []

encodings = []

for f in os.listdir("Images/"):

if f.endswith('.jpg'):

photos.append(f)

for labels in photos:

mname = labels[:-4]

names.append(mname)

mimage = mname + "\_image"

mencoding = mname + "\_face\_encoding"

mimage = face\_recognition.load\_image\_file("Images/{}".format(labels))

mencoding = face\_recognition.face\_encodings(mimage)[0]

RUN FACE ID

import face\_recognition

import cv2

import pickle

def runCamera():

count = 0

gateOpen = False

names = []

encodings = []

with open('faceEncodings', 'rb') as file1:

encodings = pickle.load(file1)

with open('faceNames', 'rb') as file2:

names = pickle.load(file2)

known\_faces = encodings

video\_capture = cv2.VideoCapture(0)

#video\_capture = cv2.VideoCapture('<http://192.168.1.105:4747/mjpegfeed>')

face\_locations = []

face\_encodings = []

process\_this\_frame = True

print("Press 'Q' to close camera!!")

while True:

ret, frame = video\_capture.read()

if process\_this\_frame:

face\_locations = face\_recognition.face\_locations(frame)

face\_encodings = face\_recognition.face\_encodings(frame, face\_locations)

face\_names = []

for face\_encoding in face\_encodings:

match = face\_recognition.compare\_faces(known\_faces, face\_encoding)

name = "Unknown"

for itr in match:

if match[itr]:

name = names[itr]

face\_names.append(name)

process\_this\_frame = not process\_this\_frame

for (top, right, bottom, left), name in zip(face\_locations, face\_names):

if(name == "Unknown"):

cv2.rectangle(frame, (left, top), (right, bottom), (0, 0, 255), 2)

cv2.rectangle(frame, (left, bottom - 35), (right, bottom), (0, 0, 255), cv2.FILLED)

#print("\a Unknown!!!")

else:

cv2.rectangle(frame, (left, top), (right, bottom), (0, 255, 0), 2)

cv2.rectangle(frame, (left, bottom - 35), (right, bottom), (0, 255, 0), cv2.FILLED)

font = cv2.FONT\_HERSHEY\_DUPLEX

cv2.putText(frame, name, (left + 6, bottom - 6), font, 1.0, (255, 255, 255), 1)

cv2.imshow('Video', frame)

if cv2.waitKey(1) & 0xFF == ord('q'):

break

video\_capture.release()

cv2.destroyAllWindows()

SEND BLUETOOTH COMMAND

import bluetooth

import sys

bd\_addr = "00:21:13:00:F9:BA"  # itade address

port = 1

sock = bluetooth.BluetoothSocket(bluetooth.RFCOMM)

sock.connect((bd\_addr, port))

print('Connected')

def sendData(comm):

sock.send(comm)

def recieveData():

adata = sock.recv(3)

return adata

def closeSocket():

sock.close()

2.5 ALGORITHMS

OEPNCV FACE RECOGNIZER

OpenCV has three built in face recognizers and thanks to OpenCV's clean coding, you can use any of them by just changing a single line of code. Below are the names of those face recognizers and their OpenCV calls. EigenFaces Face Recognizer Recognizer- cv2.face.createEigenFaceRecognizer()FisherFaces Face Recognizer Recognizer- cv2.face.createFisherFaceRecognizer()Local Binary Patterns Histograms (LBPH) Face Recognizer - cv2.face.createLBPHFaceRecognizer()

We have got three face recognizers but do you know which one to use and when? Or which one is better? I guess not. So why not go through a brief summary of each, what you say? I am assuming you said yes :) So let's dive into the theory of each.

EIGENFACES FACE RECOGNIZER

This algorithm considers the fact that not all parts of a face are equally important and equally useful. When you look at some one you recognize him/her by his distinct features like eyes, nose, cheeks, forehead and how they vary with respect to each other. So you are actually focusing on the areas of maximum change (mathematically speaking, this change is variance) of the face. For example, from eyes to nose there is a significant change and same is the case from nose to mouth. When you look at multiple faces you compare them by looking at these parts of the faces because these parts are the most useful and important components of a face. Important because they catch the maximum change among faces, change the helps you differentiate one face from the other. This is exactly how EigenFaces face recognizer works.

EigenFaces face recognizer looks at all the training images of all the persons as a whole and try to extract the components which are important and useful (the components that catch the maximum variance/change) and discards the rest of the components. This way it not only extracts the important components from the training data but also saves memory by discarding the less important components. These important components it extracts are called **principal components**. Below is an image showing the principal components extracted from a list of faces.

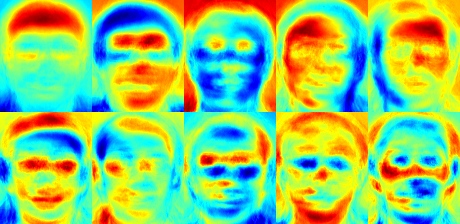
[](https://github.com/informramiz/opencv-face-recognition-python/blob/master/visualization/eigenfaces_opencv.png)

FIG-2.5 LIST OF FACES

PRINCIPAL COMPONENTS

You can see that principal components actually represent faces and these faces are called **eigen faces** and hence the name of the algorithm.So this is how EigenFaces face recognizer trains itself (by extracting principal components). Remember, it also keeps a record of which principal component belongs to which person. One thing to note in above image is that **Eigenfaces algorithm also considers illumination as an important component**.Later during recognition, when you feed a new image to the algorithm, it repeats the same process on that image as well. It extracts the principal component from that new image and compares that component with the list of components it stored during training and finds the component with the best match and returns the person label associated with that best match component.

### FISHERFACES FACE RECOGNIZER

This algorithm is an improved version of EigenFaces face recognizer. Eigenfaces face recognizer looks at all the training faces of all the persons at once and finds principal components from all of them combined. By capturing principal components from all the of them combined you are not focusing on the features that discriminate one person from the other but the features that represent all the persons in the training data as a whole.

This approach has drawbacks, for example, **images with sharp changes (like light changes which is not a useful feature at all) may dominate the rest of the images** and you may end up with features that are from external source like light and are not useful for discrimination at all. In the end, your principal components will represent light changes and not the actual face features.Fisherfaces algorithm, instead of extracting useful features that represent all the faces of all the persons, it extracts useful features that discriminate one person from the others. This way features of one person do not dominate over the others and you have the features that discriminate one person from the others.

Below is an image of features extracted using Fisherfaces algorithm.

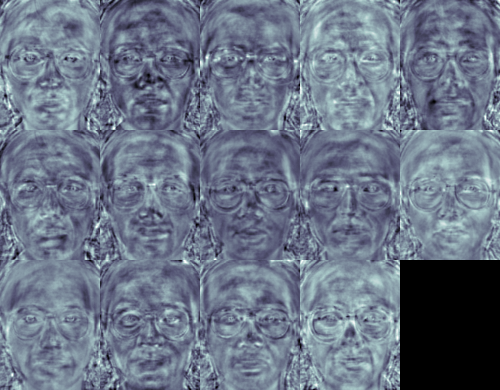
[](https://github.com/informramiz/opencv-face-recognition-python/blob/master/visualization/fisherfaces_opencv.png)

FIG 2.51 FISHER FACES

FISHER FACES

You can see that features extracted actually represent faces and these faces are called **fisher faces** and hence the name of the algorithm.One thing to note here is that **even in Fisherfaces algorithm if multiple persons have images with sharp changes due to external sources like light they will dominate over other features and affect recognition accuracy**.Getting bored with this theory? Don't worry, only one face recognizer is left and then we will dive deep into the coding part.

LOCAL BINARY PATTERN HISTOGRAM (LBPH) FACE RECOGNIZER

I wrote a detailed explaination on Local Binary Patterns Histograms in my previous article on [face detection](https://www.superdatascience.com/opencv-face-detection/) using local binary patterns histograms. So here I will just give a brief overview of how it works.We know that Eigenfaces and Fisherfaces are both affected by light and in real life we can't guarantee perfect light conditions. LBPH face recognizer is an improvement to overcome this drawback.Idea is to not look at the image as a whole instead find the local features of an image. LBPH alogrithm try to find the local structure of an image and it does that by comparing each pixel with its neighboring pixels.

Take a 3x3 window and move it one image, at each move (each local part of an image), compare the pixel at the center with its neighbor pixels. The neighbors with intensity value less than or equal to center pixel are denoted by 1 and others by 0. Then you read these 0/1 values under 3x3 window in a clockwise order and you will have a binary pattern like 11100011 and this pattern is local to some area of the image. You do this on whole image and you will have a list of local binary patterns.

LBP LABELLING

Now you get why this algorithm has Local Binary Patterns in its name? Because you get a list of local binary patterns. Now you may be wondering, what about the histogram part of the LBPH? Well after you get a list of local binary patterns, you convert each binary pattern into a decimal number (as shown in above image) and then you make a HISTOGRAM of all of those values. A sample histogram looks like this.

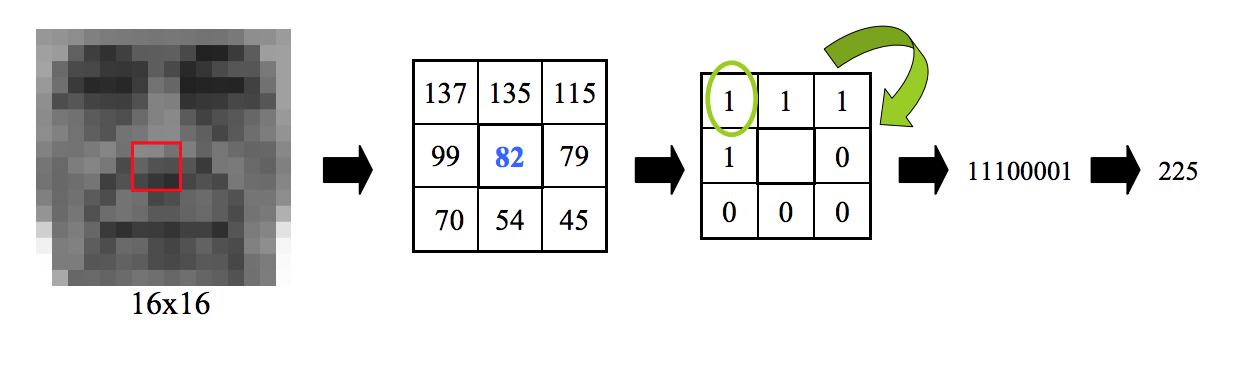
[](https://github.com/informramiz/opencv-face-recognition-python/blob/master/visualization/lbp-labeling.png)

FIG 2.6 - SAMPLE HISTOGRAM

I guess this answers the question about histogram part. So in the end you will have **one histogram for each face** image in the training data set. That means if there were 100 images in training data set then LBPH will extract 100 histograms after training and store them for later recognition. Remember, **algorithm also keeps track of which histogram belongs to which person**.Later during recognition, when you will feed a new image to the recognizer for recognition it will generate a histogram for that new image, compare that histogram with the histograms it already has, find the best match histogram and return the person label associated with that best match histogram.Below is a list of faces and their respective local binary patterns images.

You can see that the LBP images are not affected by changes in light conditions.

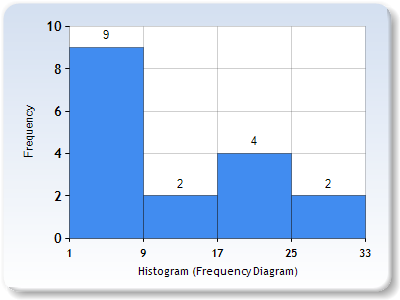
[](https://github.com/informramiz/opencv-face-recognition-python/blob/master/visualization/histogram.png)

FIG-2.7 HISTOGRAM

LBP FACES

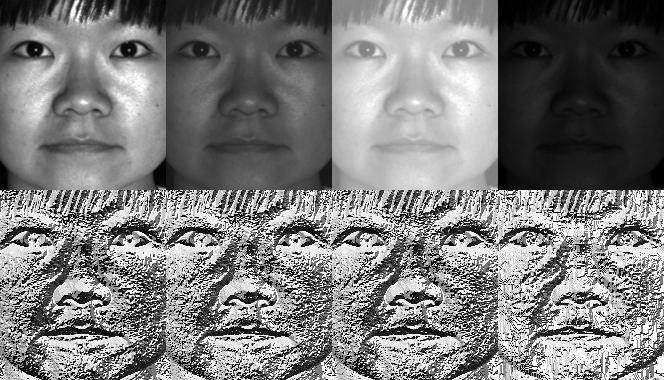
[](https://github.com/informramiz/opencv-face-recognition-python/blob/master/visualization/lbph-faces.jpg)

FIG-2.6 LBP FACES

The Face Recognition process in this tutorial is divided into three steps.

**Prepare training data:** In this step we will read training images for each person/subject along with their labels, detect faces from each image and assign each detected face an integer label of the person it belongs to.

**Train Face Recognizer:** In this step we will train OpenCV's LBPH face recognizer by feeding it the data we prepared in step 1.

**Testing:** In this step we will pass some test images to face recognizer and see if it predicts them correctlyTo detect faces, I will use the code on FACE DETECTION. So if you have not read it, I encourage you to do so to understand how face detection works and its Python coding.

CONNECTING THE CAMERA

The camera used in the system can be any USB HD camera that needs to be properly connected with the system for execution. It should be also kept in mind that the megapixel of the camera is good enough to detect a face properly. The kind of camera used in this system is shown in figure 2.7.



FIG 2.7-A USB HD CAMERA

INPUTTING MODULES

The next step is to import the modules: CV2, OS, Image and Numphy

cv2 - For face detection and recognition the OpenCV module and contains the functions.os - First, this module is used to extract the image names in the database directory. From these names individual number, which will be used as a label for the face in that image is extracted. This module will be used to maneuver with image and directory names.Image – It will use Image module from Python Imaging Library (PIL) to read the image in grayscale format. Since, the dataset images are in gif format and as of now, OpenCV does not support gif format.numpy - The images will be stored in numpy arrays.

TRAINING RECOGNIZER FOR A FACE DETECTION

First, an image database has been created, containing faces of the users with whom the face on the camera can be matched. In each image, the individual has a different facial expression. For example, there will be several images for the first individual. The database will use these images of each individual to training the recognizer.

**Chapter 3**

**METHODS AND IMPLEMENTATION DETAIL**

3.1 INTRODUCTION

Recognize and manipulate faces from Python or from the command line with

the world’s simplest face recognition library. Built using [dlib](http://dlib.net/)’s state-of-the-art face recognition built with deep learning. The model has an accuracy of 99.38% on the

[Labeled Faces in the Wild](http://vis-www.cs.umass.edu/lfw/) benchmark.

In this era of time, security issues are given utmost priority as the situations around places aren‟t suitable often. Every business owner strives to keep their employees, assets, and office space as safe as possible and same goes for homes of people. A number of incidents occurred recently in this country due to lack of safety that caused a stir among people.Crimes have increased rapidly due to lack of security measures. The growing crime rates across cities reflect the bitter reality. Many people overlook, ignore, and underestimate the need of taking appropriate home security measures. A burglary or theft can lead to devastating consequences, both emotionally and financially. While the financial loss may be recoverable, the trauma inflicted on the family may last forever. There are several ways to help increase the security at doors; one of the most effective is to install a security system that is simple as well as protective. Smart Security System is thus proposed viewing the need of an advanced security system.

3.2 PROCESS EXECUTION

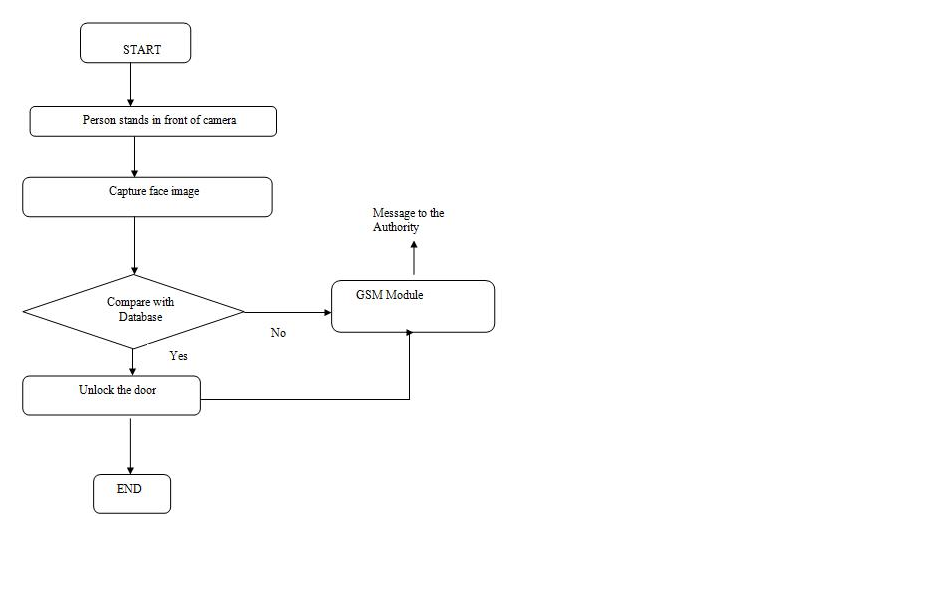


FIG 3.2 FLOW DIAGRAM

3.3 THE DOOR LOCK



3.4 CODING FACE RECOGNITION WITH OPENCV STEPBY STEP

### IMPORT REQUIRED MODULES

Before starting the actual coding we need to import the required modules for coding. So let's import them first.**cv2:** is OpenCV module for Python which we will use for face detection and face recognition.**os:** We will use this Python module to read our training directories and file names.**numpy:** We will use this module to convert Python lists to numpy arrays as OpenCV face recognizers accept numpy arrays.

* #import OpenCV module
* import cv2
* #import os module for reading training data directories and paths
* import os
* #import numpy to convert python lists to numpy arrays as
* #it is needed by OpenCV face recognizers
* import numpy as np
* #matplotlib for display our images
* import matplotlib.pyplot as plt
* %matplotlib inline

### TRAINING DATA

The more images used in training the better. Normally a lot of images are used for training a face recognizer so that it can learn different looks of the same person, for example with glasses, without glasses, laughing, sad, happy, crying, with beard, without beard etc. To keep our tutorial simple we are going to use only 12 images for each person. So our training data consists of total 2 persons with 12 images of each person. All training data is inside *training-data*folder. *training-data* folder contains one folder for each person and **each folder is named with format sLabel (e.g. s1, s2) where label is actually the integer label assigned to that person**. For example folder named s1 means that this folder contains images for person 1. The directory structure tree for training data is as follows:

training-data

|-------------- s1

| |-- 1.jpg

| |-- ...

| |-- 12.jpg

|-------------- s2

| |-- 1.jpg

| |-- ...

| |-- 12.jpg

The *test-data* folder contains images that we will use to test our face recognizer after it has been successfully trained. As OpenCV face recognizer accepts labels as integers so we need to define a mapping between integer labels and persons actual names so below I am defining a mapping of persons integer labels and their respective names.

**Note:** As we have not assigned label 0 to any person so **the mapping for label 0 is empty**.

#there is no label 0 in our training data so subject name for index/label 0 is empty

subjects = ["", "yogesh choudhary", "amit chaberwal"]

PREPARE TRAINING DATA

You may be wondering why data preparation, right? Well, OpenCV face recognizer accepts data in a specific format. It accepts two vectors, one vector is of faces of all the persons and the second vector is of integer labels for each face so that when processing a face the face recognizer knows which person that particular face belongs too.

For example, if we had 2 persons and 2 images for each person.

PERSON-1 PERSON-2

img1 img1

img2 img2

Then the prepare data step will produce following face and label vectors.

FACES LABELS

person1\_img1\_face 1

person1\_img2\_face 1

person2\_img1\_face 2

person2\_img2\_face 2

Preparing data step can be further divided into following sub-steps.

Read all the folder names of subjects/persons provided in training data folder. So for example, in this tutorial we have folder names: s1, s2. For each subject, extract label number. **Do you remember that our folders have a special naming convention?** Folder names follow the format sLabel where Label is an integer representing the label we have assigned to that subject. So for example, folder name s1 means that the subject has label 1, s2 means subject label is 2 and so on.

The label extracted in this step is assigned to each face detected in the next step. Read all the images of the subject, detect face from each image. Add each face to faces vector with corresponding subject label (extracted in above step) added to labels vector. do so right now because to detect faces, I am going to use the code from my previous article on [face detection](https://www.superdatascience.com/opencv-face-detection/). So if you have not read it, I encourage you to do so to understand how face detection works and its coding. Below is the same code.

def detect\_face(img):

#convert the test image to gray image as opencv face detector expects gray images

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

#load OpenCV face detector, I am using LBP which is fast

#there is also a more accurate but slow Haar classifier

face\_cascade = cv2.CascadeClassifier('opencv-files/lbpcascade\_frontalface.xml')

#let's detect multiscale (some images may be closer to camera than others) images

#result is a list of faces

faces = face\_cascade.detectMultiScale(gray, scaleFactor=1.2, minNeighbors=5);

#if no faces are detected then return original img

if (len(faces) == 0):

return None, None

#under the assumption that there will be only one face,

#extract the face area

(x, y, w, h) = faces[0]

#return only the face part of the image

return gray[y:y+w, x:x+h], faces[0]

I am using OpenCV's **LBP face detector**. On line 4, I convert the image to grayscale because most operations in OpenCV are performed in gray scale, then on line 8 I load LBP face detector using cv2.CascadeClassifier class. After that on line 12 I use cv2.CascadeClassifier class' detectMultiScale method to detect all the faces in the image. on line 20, from detected faces I only pick the first face because in one image there will be only one face (under the assumption that there will be only one prominent face).

As faces returned by detectMultiScale method are actually rectangles (x, y, width, height) and not actual faces images so we have to extract face image area from the main image. So on line 23 I extract face area from gray image and return both the face image area and face rectangle. Now you have got a face detector and you know the 4 steps to prepare the data, so are you ready to code the prepare data step? Yes? So let's do it.

#this function will read all persons' training images, detect face from each image

#and will return two lists of exactly same size, one list

# of faces and another list of labels for each face

def prepare\_training\_data(data\_folder\_path):

#------STEP-1--------

#get the directories (one directory for each subject) in data folder

dirs = os.listdir(data\_folder\_path)

#list to hold all subject faces

faces = []

#list to hold labels for all subjects

labels = []

#let's go through each directory and read images within it

for dir\_name in dirs:

#our subject directories start with letter 's' so

#ignore any non-relevant directories if any

if not dir\_name.startswith("s"):

continue;

#------STEP-2--------

#extract label number of subject from dir\_name

#format of dir name = slabel

#, so removing letter 's' from dir\_name will give us label

label = int(dir\_name.replace("s", ""))

#build path of directory containin images for current subject subject

#sample subject\_dir\_path = "training-data/s1"

subject\_dir\_path = data\_folder\_path + "/" + dir\_name

#get the images names that are inside the given subject directory

subject\_images\_names = os.listdir(subject\_dir\_path)

#------STEP-3--------

#go through each image name, read image,

#detect face and add face to list of faces

for image\_name in subject\_images\_names:

#ignore system files like .DS\_Store

if image\_name.startswith("."):

continue;

#build image path

#sample image path = training-data/s1/1.pgm

image\_path = subject\_dir\_path + "/" + image\_name

#read image

image = cv2.imread(image\_path)

#display an image window to show the image

cv2.imshow("Training on image...", image)

cv2.waitKey(100)

#detect face

face, rect = detect\_face(image)

#------STEP-4--------

#for the purpose of this tutorial

#we will ignore faces that are not detected

if face is not None:

#add face to list of faces

faces.append(face)

#add label for this face

labels.append(label)

cv2.destroyAllWindows()

cv2.waitKey(1)

cv2.destroyAllWindows()

return faces, labels

I have defined a function that takes the path, where training subjects' folders are stored, as parameter. This function follows the same 4 prepare data substeps mentioned. **(step-1)** On line 8 I am using os.listdir method to read names of all folders stored on path passed to function as parameter. On line 10-13 I am defining labels and faces **(step-2)** After that I traverse through all subjects' folder names and from each subject's folder name on line 27 I am extracting the label information. As folder names follow the sLabel naming convention so removing the letter s from folder **(step-3)** On line 34, I read all the images names of of the current subject being traversed and on line 39-66 I traverse those images one by one. On line 53-54 I am using The waitKey(interval) method pauses the code flow for the given interval (milliseconds), I am using it with 100ms interval so that we can view the image window for 100ms. On line 57, I detect face from the current image . **(step-4)** On line 62-66, I add the detected face and label to their respective vectors. But a function can't do anything unless we call it on some data that it has to prepare, right? Don't worry, I have got data of two beautiful and famous celebrities. I am sure you will recognize them!

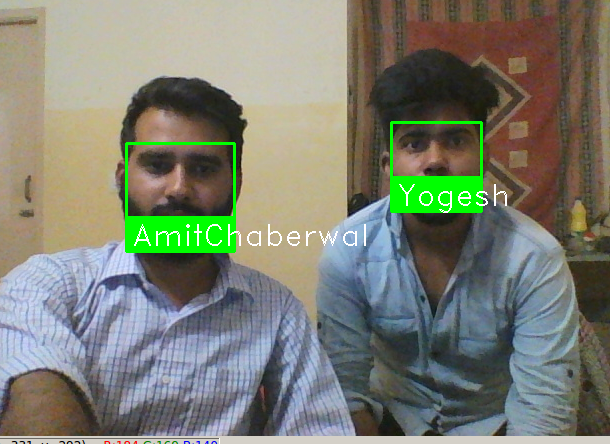
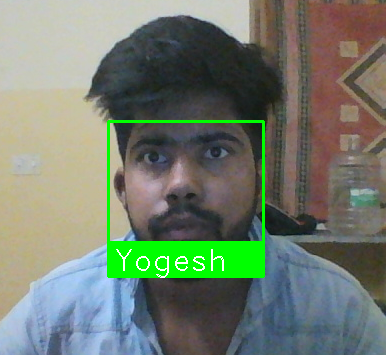


FIG3.41 - FACE DETECTION

Let's call this function on images of us to prepare data for training of our Face Recognizer. Below is a simple code to do that.

#let's first prepare our training data

#data will be in two lists of same size

#one list will contain all the faces

#and other list will contain respective labels for each face

print("Preparing data...")

faces, labels = prepare\_training\_data("training-data")

print("Data prepared")

#print total faces and labels

print("Total faces: ", len(faces))

print("Total labels: ", len(labels))

Preparing data...

Data prepared

Total faces: 23

Total labels: 23

This was probably the boring part, right? Don't worry, the fun stuff is coming up next. It's time to train our own face recognizer so that once trained it can recognize new faces of the persons it was trained on. Read? Ok then let's train our face recognizer.

TRAIN FACE RECOGNIZER

As we know, OpenCV comes equipped with three face recognizers. EigenFace Recognizer: This can be created with cv2.face.createEigenFaceRecognizer(). FisherFace Recognizer: This can be created with  cv2 .face. create Fisher Face Recognizer(). Local Binary Patterns Histogram (LBPH): This can be created with cv2.face.LBPHFisherFaceRecognizer().I am going to use LBPH face recognizer but you can use any face recognizer of your choice. No matter which of the OpenCV's face recognizer you use the code will remain the same. You just have to change one line, the face recognizer initialization line given below.

#create our LBPH face recognizer

face\_recognizer = cv2.face.createLBPHFaceRecognizer()

#or use EigenFaceRecognizer by replacing above line with

#face\_recognizer = cv2.face.createEigenFaceRecognizer()

#or use FisherFaceRecognizer by replacing above line with

#face\_recognizer = cv2.face.createFisherFaceRecognizer()

Now that we have initialized our face recognizer and we also have prepared our training data, it's time to train the face recognizer. We will do that by calling the train(faces-vector, labels-vector) method of face recognizer.

#train our face recognizer of our training faces

face\_recognizer.train(faces, np.array(labels))

**Did you notice** that instead of passing labels vector directly to face recognizer I am first converting it to **numpy** array? This is because OpenCV expects labels vector to be a numpy array.

PREDICTION

Now comes my favorite part, the prediction part. This is where we actually get to see if our algorithm is actually recognizing our trained subjects's faces or not. We will take two test images of our celeberities, detect faces from each of them and then pass those faces to our trained face recognizer to see if it recognizes them. Below are some utility functions that we will use for drawing bounding box (rectangle) around face and putting celeberity name near the face bounding box.

#function to draw rectangle on image

#according to given (x, y) coordinates and

#given width and heigh

def draw\_rectangle(img, rect):

(x, y, w, h) = rect

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

#function to draw text on give image starting from

#passed (x, y) coordinates.

def draw\_text(img, text, x, y):

cv2.putText(img, text, (x, y), cv2.FONT\_HERSHEY\_PLAIN, 1.5, (0, 255, 0), 2)

First function draw\_rectangle draws a rectangle on image based on passed rectangle coordinates. It uses OpenCV's built in function cv2.rectangle(img, topLeftPoint, bottomRightPoint, rgbColor, lineWidth) to draw rectangle. We will use it to draw a rectangle around the face detected in test image. Second function draw\_text uses OpenCV's built in function cv2.putText(img, text, startPoint, font, fontSize, rgbColor, lineWidth) to draw text on image. Now that we have the drawing functions, we just need to call the face recognizer's predict(face) method to test our face recognizer on test images. Following function does the prediction for us.

#this function recognizes the person in image passed

#and draws a rectangle around detected face with name of the

#subject

def predict(test\_img):

#make a copy of the image as we don't want to chang original image

img = test\_img.copy()

#detect face from the image

face, rect = detect\_face(img)

#predict the image using our face recognizer

label= face\_recognizer.predict(face)

#get name of respective label returned by face recognizer

label\_text = subjects[label]

#draw a rectangle around face detected

draw\_rectangle(img, rect)

#draw name of predicted person

draw\_text(img, label\_text, rect[0], rect[1]-5)

return img

**line-6** read the test image, **line-7** detect face from test image, **line-11** recognize the face by calling face recognizer's predict(face) method. This method will return a lable, **line-12** get the name associated with the label, **line-16** draw rectangle around the detected face, **line-18** draw name of predicted subject above face rectangle.Now that we have the prediction function well defined, next step is to actually call this function on our test images and display those test images to see if our face recognizer correctly recognized them. So let's do it. This is what we have been waiting for.

print("Predicting images...")

#load test images

test\_img1 = cv2.imread("test-data/test1.jpg")

test\_img2 = cv2.imread("test-data/test2.jpg")

#perform a prediction

predicted\_img1 = predict(test\_img1)

predicted\_img2 = predict(test\_img2)

print("Prediction complete")

#create a figure of 2 plots (one for each test image)

f, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 5))

#display test image1 result

ax1.imshow(cv2.cvtColor(predicted\_img1, cv2.COLOR\_BGR2RGB))

#display test image2 result

ax2.imshow(cv2.cvtColor(predicted\_img2, cv2.COLOR\_BGR2RGB))

#display both images

cv2.imshow("yogesh test", predicted\_img1)

cv2.imshow("amit test", predicted\_img2)

cv2.waitKey(0)

cv2.destroyAllWindows()

cv2.waitKey(1)

cv2.destroyAllWindows()

**Chapter 4**

**EXPERIMENTAL RESULTS**

The result in the creation of real time database are recorded. The real time database is created by using python. While executing it produces 30 images of each subject. . Likewise, databases should be created for at least 10 individuals and it creates each image size of about 100\*130 pixels of height and width.

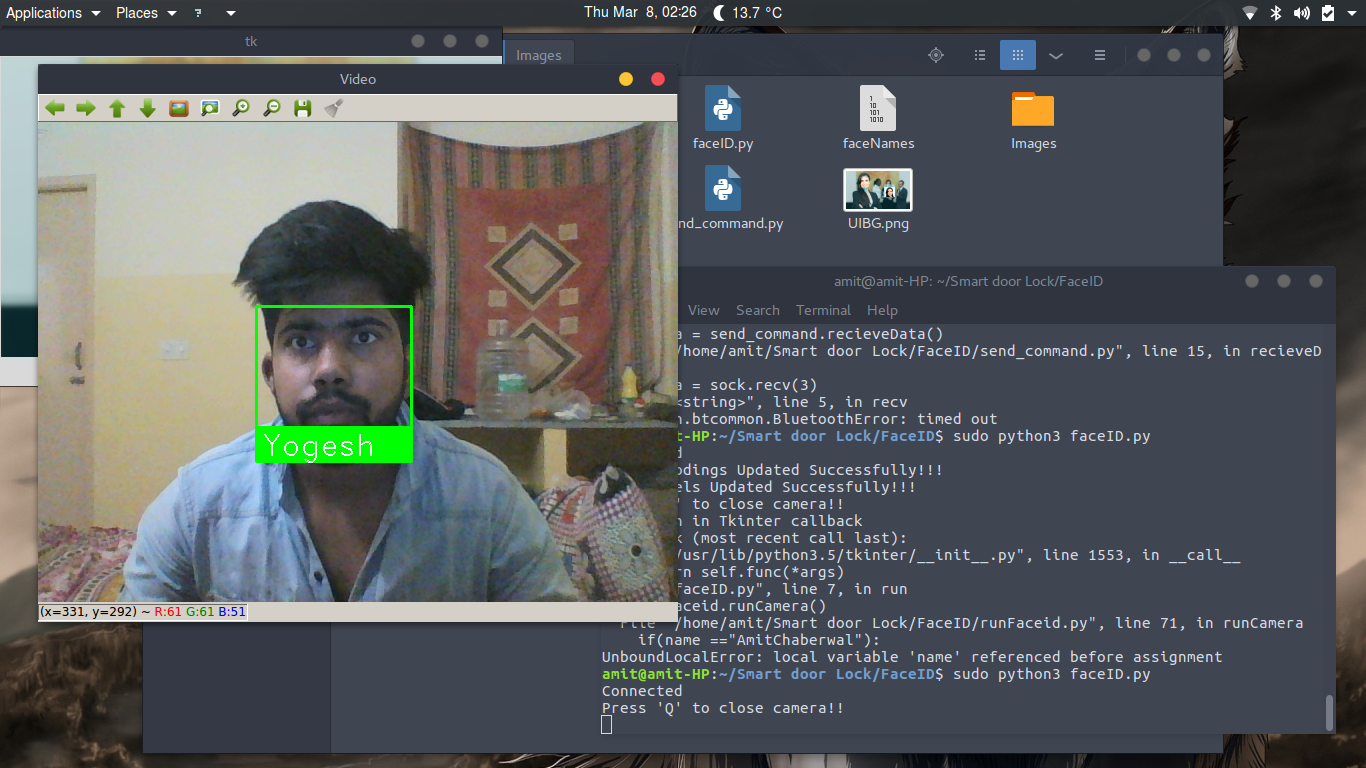
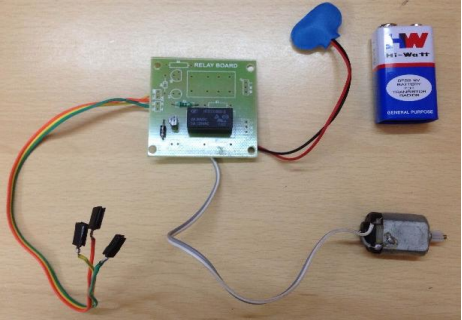


FIG 3.11- SINGLE FACE DETECTION

So when single image recognition is taken then this detect, otherwise shows unknown

Source.Its not so that it only recognizes single face, but it also recognizes multiple faces which comes in front of opencv.

FINAL DOOR LOCK SETUP WORKING



SO HERE USING THIS 5V BATTERY

So by using this battery along with a model based gate is used to show the opening and closing of door as per face recognition.

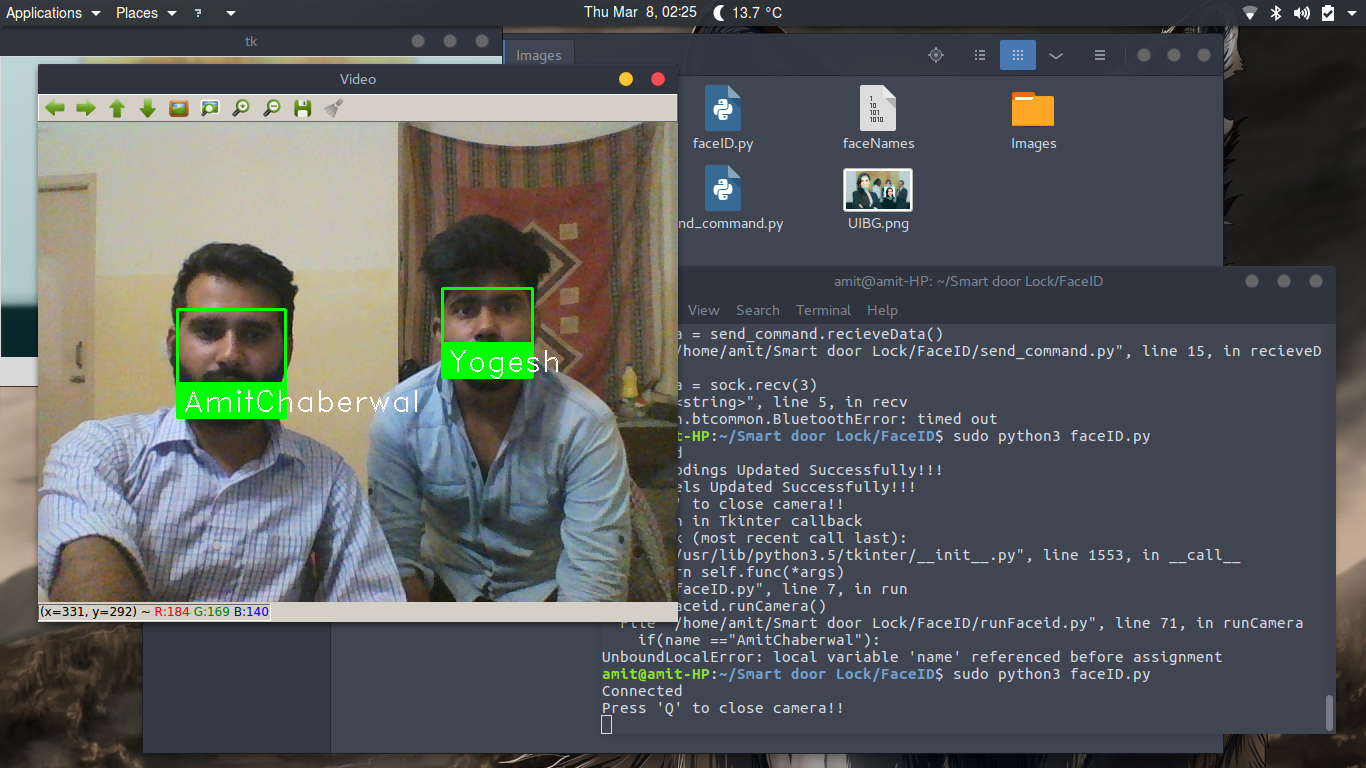


FIG 3.12 MULTIPLE FACE DETECTION

So this is facial recognition. Now we will have a look at hardware part, how when face is detected and then hardware acts. So if face is detected then via Bluetooth how it was performed.



FIG 3.2- BLUETOOTH CONNECTION

So this Bluetooth connection which is used as trial bases. Then a gain code is linked with hardware, and we make this Bluetooth work as signal to LED when face is been recognized or not recognized via green and red LED respectively.



FIG 3.3-FACE DETECTION INVALID-SIGNAL RED AND GATE DOESNT OPEN

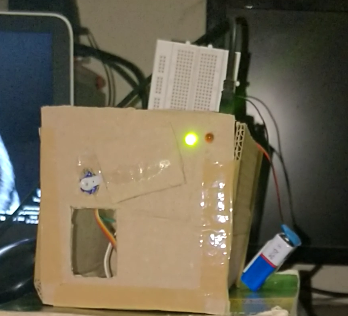


FIG3.4 FACE DETECTION VALID SIGNAL GREEN AND GATE OPENS

So accordingly faces are detected and as per that led are signalled via Bluetooth, and in further the gate either opens or remain closed.

PROS

1) No More Time Fraud–One of the big benefits of using facial biometric systems in your company is that you won’t have to worry about time fraud. It will be impossible for buddy punching to occur, since everyone has to have go throughout face scanning biometrics devices to clock in.

2) Enhanced Security–You’ll also enjoy enhanced security with a face biometrics system. Not only can you track employees thru biometrics time attendance tracking, but any visitors can be added to the system and tracked throughout the area too. Anyone that is not in the system will not be given access.

CONS

1) Image quality-Image quality affects how well facial-recognition algorithms work. The image quality of scanning video is quite low compared with that of a digital camera. Even high-definition.

2) Image Size: When a face-detection algorithm finds a face in an image or in a still from a video capture, the relative size of that face compared with the enrolled image size affects how well the face will be recognized.

APPLICATIONS

1) Image and face recognition is bringing whole new dimension to gaming

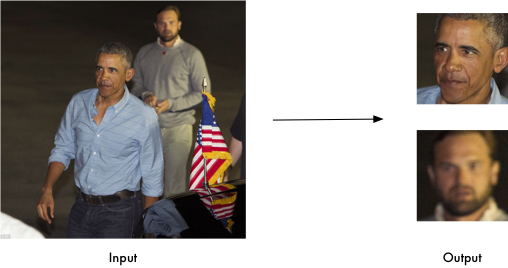
2) It can also be used for price comparison by taking pictures of the items and comparing with the social sites.

3) and this one smart door locking system

FEATURES

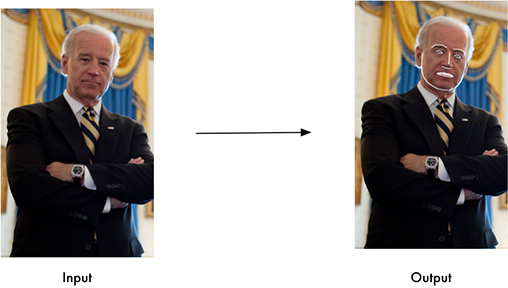
### FIND FACES IN PICTURE

Find all the faces that appear in a picture:



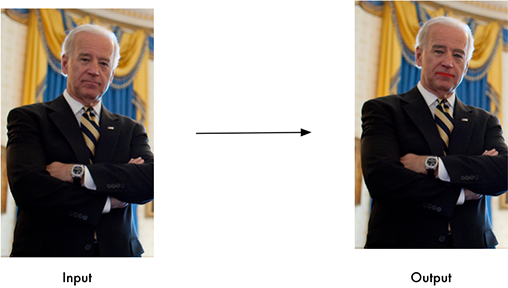
FIND AND MANIPULATE FACIAL FEATURES IN PICTURES

Get the locations and outlines of each person’s eyes, nose, mouth and chin.

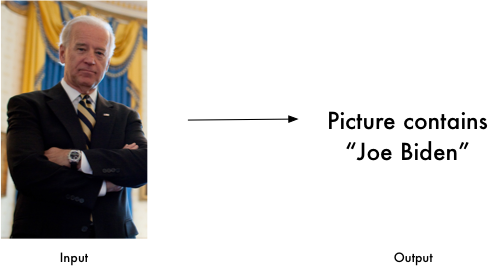


Finding facial features is super useful for lots of important stuff. But you can also use for really stupid stuff

like applying [digital make-up](https://github.com/ageitgey/face_recognition/blob/master/examples/digital_makeup.py) (think ‘Meitu’)



IDENTIFY FACES IN PICTURE



**Chapter 5**

**CONCLUSION**

In this project, security system based on Bluetooth connectivity and face detection is proposed. The Bluetooth connectivity with facial recognition is tested with other several algorithms and the result comes negative unless that particular face is given in database. Followed by the establishment of the Bluetooth connection, the face detection part comes where the test result was able to give a complete accuracy.

Face detection method was done using OpenCv, deep learning and python. This way we analyzes each face in the training set separately and independently and thus is the optimum choice.

This system is a mass solution to the common masses due to its affordability and lack of training or expertise required so instead of spending money for a costly unit of surveillance this is a better choice. As such, it can be regarded as an elegant as well as a practical solution keeping the budget in mind.

**REFERENCES**

N. Anukrishnan, B. Ramya, S. Mohan, Design and development of car ignition access control system based on face recognition technique, SAS TECH Journal 9 (2) (2016)63-70

Stapathy, A. and Das, D.P., "A system for remote operation of devices: Helpful for elderly and disabled people" in Proc. Of EEE International Conf on Advanced Electronic Systems, pp. 350-353, 2015.

Kuang-Yow Lian, Sung-Jung Hsiao and Wen-Tsai Sung, "Home Safety Handwriting Pattern Recognition System" in Proc. of IEEE International Conf on Cognitive Informatics and Cognitive Computing, pp. 477-483, 2015.

Y. Wang, J. Lu and N. Ge, "Priority-based adaptive frequency hopping for Bluetooth in multi-piconet environments - IEEE Xplore Document", Ieeexplore.ieee.org, 2015.

Fahim Slauddin and Tarif Riyad Rahman. “A Fuzzy based low-cost monitoring module built with raspberry pi – python – java architecture”. International Conference on Smart Sensors and Application (ICSSA), 2015.

Open Source Computer Vision Library, “The OpenCV Tutorial” https://docs.opencv.org/ 2.4/opencv tutorials.pdf

C. L. Huang and S. H. Jeng, ”A model-based hand gesture recognition system, ”Machine Vision and Appl., vol.12,Pp.243-258, 2017 J. Hui and D. Culler, “Extending IP to Low-Power Wireless Personal Area Networks,”

IEEE Internet Computing, July 2008, pp. 37–45.

Robert C. Schultz, Robert W. Ives, “Biometric Data Acquisition using MATLAB GUIs,”

35th ASEE/IEEE Frontiers in Education Conference, Oct 19-22,2016.

Michal Coras, “Perspective Methods of Biometric Human Identification,” Inst.

R. Chand and P. Felber, “A scalable protocol for content-based routing in overlay networks,” in Network Computing and Applications, 2016. NCA 2009. Second IEEE International Symposium on, April 2003, pp. 123–130.

A. Shukla and N. Tyagi, “A new route maintenance in dynamic source routing protocol,” in Wireless Pervasive Computing, 2015 1st International Symposium on, Jan. 2015, pp. 4

Kuang-Yow Lian, Sung-Jung Hsiao and Wen-Tsai Sung, "Home Safety Handwriting Pattern Recognition System" in Proc. of IEEE International Conf on Cognitive Informatics and Cognitive Computing, pp. 477-483, 2016.

Y. Wang, J. Lu and N. Ge, "Priority-based adaptive frequency hopping for Bluetooth in multi-piconet environments - IEEE Xplore Document", Ieeexplore.ieee.org, 2015.

Fahim Slauddin and Tarif Riyad Rahman. “A Fuzzy based low-cost monitoring module built with raspberry pi – python – java architecture”. International Conference on Smart Sensors and Application (ICSSA), 2016.

Open Source Computer Vision Library, “The OpenCV Tutorial” https://docs.opencv.org/ 2.4/opencv tutorials.pdf