MINI PROJECT REPORT

**On**

**Face Detection Based Attendance Management System**

**Submitted by**

**Punya Prakash sharma**

**(171500244)**

**Rameshwar Agrawal**

**(171500255)**

Department of Computer Engineering & Applications

**Institute of Engineering & Technology**



**GLA University**

**Mathura- 281406, INDIA**

**2019**

**Department of computer Engineering and Applications**

**GLA University, Mathura**

**17 km. Stone NH#2, Mathura-Delhi Road, P.O. – Chaumuha,**

**Mathura – 281406**



**Declaration**

I hereby declare that the work which is being presented in the Mini Project “**Attendance Management System using Open CV”,**in partial fulfillment of the requirements for Mini Project viva voce, is an authentic record of my own work carried under the supervision of “Mr. Vivek Kumar”

Signature :

Group Members: Punya Prakash Sharma(171500244)

Rameshwar Agrawal(171500255)

Course: B.tech. (Computer Science and Engineering)

Year: 3rd

Semester: 5th

**MINI PROJECT – I**

**(2018-19)**

# FACE DETECTION BASED ATTENDANCE

# MANAGEMENT SYSTEM

# 

**SYNOPSIS**



**Institute of Engineering & Technology**

**Team Members**

Punya Prakash Sharma

(171500244)

Rameshwar Agrawal

(171500255)

## Supervised By:

**Mr. Vivek Kumar**

**Asst. Professor**

**Department of Computer Engineering & Applications**

**About the Project:**

Face recognition is an important application of Image processing owing to its use in many fields. Identification of individuals in an organization for the purpose of attendance is one such application of face recognition.The purpose of developing attendance management system is to computerize the traditional way of taking attendance. Automated Attendance Management System performs the daily activities of attendance marking and analysis with reduced human intervention. The prevalent techniques and methodologies for detecting and recognizing face fail to overcome issues such as scaling, pose, illumination, variations, rotation, and occlusions. The proposed system aims to overcome the pitfalls of the existing systems and provides features such as detection of faces. Better accuracy is attained in results as the system takes into account the changes that occur in the face over the period of time . The system is tested for various use cases. The system is tested under varying lighting conditions, various facial expressions, presence of partial faces and presence or absence of beard and spectacles. this project reduces the chances of fake attendance as it save attendance after recognization of the saved faces

**Motivation:**

The motivation behind this project is to make attendance management system efficient and secure as the system saves attendance after recognization of the face which is stored in the database.

**Future Prospects:**

Further modification can be done in the system such as modifying the attendance after taking attendance.Viewing the past record of the attendance of the students can be graphed and analysed.

**Requirements:**

1. **Hardware:**

* **Hard disk – 5GB**
* **RAM – 1GB**
* **Webcam**
* **Processor(intel core i3)**

1. **Software:**

* **Python**
* **Numpy**
* **Open cv**
* **Matplotlib**

**ACKNOWLEDGEMENT**

I thank the almighty for giving me the courage and perseverance in completing the project.

This project itself is acknowledgements for all those people who have give us their heartfelt co-operation in making this project a grand success.I extend my sincere thanks to Mr.Vivek Kumar, Assistant Proffesor at “GLA University,Mathura” for providing valuable guidance at every stage of this project work. I am profoundly grateful towards the unmatched services rendered by his.Last but not least , we would like to express our deep sense of gratitude and earnest thanks giving to our dear parents for their moral support and heartfelt cooperation in doing the main project.

**Attendance Management System Using Open CV**

**Abstract**

The face is one of the easiest ways to distinguish the individual identity of each other. Face Detection is a personal identification system that uses personal characteristics of a person to identify the person's identity. Human face detection procedure basically consists of two phases, namely face detection, where this process takes place very rapidly in humans, except under conditions where the object is located at a short distance away, the next is the introduction, which detect a face as individuals.Stage is then replicated and developed as a model for facial image detection (face recognition) is one of the much-studied biometrics technology and developed by experts. There are two kinds of methods that are currently popular in developed face recognition pattern namely, Eigenface method and Fisherface method. Facial image detection Eigenface method is based on the reduction of face dimensional space using Principal Component Analysis (PCA) for facial features. The main purpose of the use of PCA on face detection using Eigen faces was formed (face space) by finding the eigenvector corresponding to the largest eigenvalue of the face image. The area of this project face detection system with face recognition is Image processing.

The software requirements for this project is matlab software.

Keywords: face detection, Eigen face, PCA, matlab

Extension: There are vast number of applications from this face detection project, this project can be extended that the various parts in the face can be detect which are in various directions and shapes.

**Contents**

**Acknowlegment…………………………………….........**06

**Abstract…………………………………………………**07

1. **Introduction:**

Introduction to face Detection and OpenCV………….09

Pre requisites…………………………………………..09

Theory of face detection classifiers…………………....09

1. **Software Requirement:**

Python…………………………………………………14

Introduction toMatlab…………………………………14

Modules and their functionality…………………….....14

**3.Training Images……………………………………...**18

**4.Conclusion……………………………………………**20

**5.Bibliography………………………………………….**22

**6.** **Appendices…………………………………………..**23

**Chapter 1**

**Introduction**

**Introduction to face Detection**

Face detection is a computer vision technology that helps to locate/visualize human faces in digital images. This technique is a specific use case of [object detection technology](https://en.wikipedia.org/wiki/Object_detection) that deals with detecting instances of semantic objects of a certain class (such as humans, buildings or cars) in digital images and videos. With the advent of technology, face detection has gained a lot of importance especially in fields like photography, security, and marketing.

Face detection involves separating image windows into two classes; one containing faces (tarning the background (clutter). It is difficult because although commonalities exist between faces, they can vary considerably in terms of age, skin colour and facial expression. The problem is further complicated by differing lighting conditions, image qualities and geometries, as well as the possibility of partial occlusion and disguise. An ideal face detector would therefore be able to detect the presence of any face under any set of lighting conditions, upon any background.

The face detection task can be broken down into two steps. The first step is a classification task that takes some arbitrary image as input and outputs a binary value of yes or no, indicating whether there are any faces present in the image. The second step is the face localization task that aims to take an image as input and output the location of any face or faces within that image as some bounding box with (x, y, width, height)

**Pre-requisite**

Hands-on knowledge of Numpy and Matplotlib is essential before working on the concepts of OpenCV. Make sure that you have the following packages installed and running before installing OpenCV.

**Theory or face detection classifiers**

A computer program that decides whether an image is a positive image (face image) or negative image (non-face image) is called a classifier. A classifier is trained on hundreds of thousands of face and non-face images to learn how to classify a new image correctly. OpenCV provides us with two pre-trained and ready to be used for face detection classifiers:

Haar Classifier

LBP Classifier

Both of these classifiers process images in gray scales, basically because we don't need color information to decide if a picture has a face or not (we'll talk more about this later on). As these are pre-trained in OpenCV, their learned knowledge files also come bundled with OpenCV opencv/data/.

To run a classifier, we need to load the knowledge files first, as if it had no knowledge, just like a newly born baby (stupid babies).

Each file starts with the name of the classifier it belongs to. For example, a Haar cascade classifier starts off as haarcascade\_frontalface\_alt.xml.

**These are the two types of classifiers we will be using to analyze Casper.**

**1. HAAR Classifier**

The [Haar Classifier](https://en.wikipedia.org/wiki/Haar-like_features) is a machine learning based approach, an algorithm created by Paul Viola and Michael Jones; which (as mentioned before) are trained from many many positive images (with faces) and negatives images (without faces).

It starts by extracting Haar features from each image as shown by the windows below:

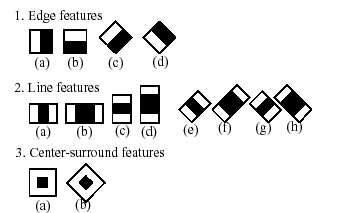


Fig 1.1

Each window is placed on the picture to calculate a single feature. This feature is a single value obtained by subtracting the sum of pixels under the white part of the window from the sum of the pixels under the black part of the window.

In the end, the algorithm considers the fact that generally: most of the region in an image is a non-face region. Considering this, it’s a better idea to have a simple method to check if a window is a non-face region, and if it's not, discard it right away and don’t process it again. So we can focus mostly on the area where a face is.

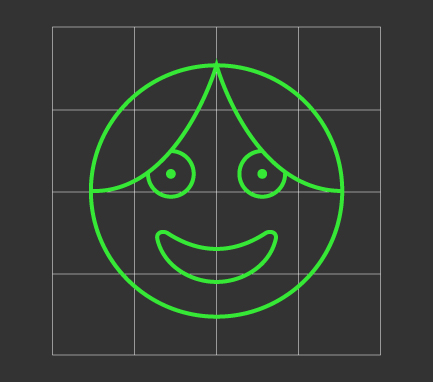
**2. LBP Cascade Classifier**

As any other classifier, the Local Binary Patterns, or LBP in short, also needs to be trained on hundreds of images. LBP is a visual/texture descriptor, and thankfully, our faces are also composed of micro visual patterns.

So, LBP features are extracted to form a feature vector that classifies a face from a non-face.

“But how are LBP features found?”

Each training image is divided into some blocks as shown in the picture below.



LBP Windows (disregard the first grader drawing)

Fig 1.2

For each block, LBP looks at 9 pixels (3×3 window) at a time, and with a particular interest in the pixel located in the center of the window.

Then, it compares the central pixel value with every neighbor's pixel value under the 3×3 window. For each neighbor pixel that is greater than or equal to the center pixel, it sets its value to 1, and for the others, it sets them to 0.

After that, it reads the updated pixel values (which can be either 0 or 1) in a clockwise order and forms a binary number. Next, it converts the binary number into a decimal number, and that decimal number is the new value of the center pixel. We do this for every pixel in a block.

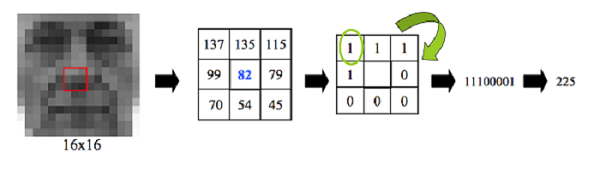


Fig 1.3

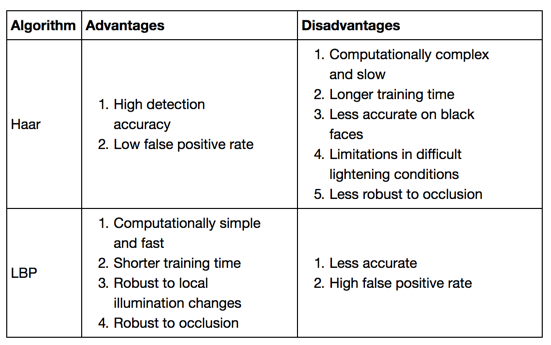


Table 1.1

Each OpenCV face detection classifier has its pros and cons, but the major differences are in accuracy and speed.

So, in case more accurate detections are required, Haar classifier is the way to go. This bad boy is more suitable in technology such as security systems or high-end stalking.

But the LBP classifier is faster, therefore, should be used in mobile applications or embedded systems.

**Chapter 2**

**Software Requirement**

**Python**

Python is an interpreted,high level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects. Python interpreters are available for many operating systems. A global community of programmers develops and maintains CPython, an open source reference implementation. A non-profit organization, the Python Software Foundation, manages and directs resources for Python and CPython development.

**Introduction to Matlab**

The name MATLAB stands for MATrix LABoratory. MATLAB was written originally to provide easy access to matrix software developed by the LINPACK (linear system package) and EISPACK (Eigen system package) projects. MATLAB is a high-performance language for technical computing. It integrates computation, visualization, and programming environment. MATLAB has many advantages compared to conventional computer languages (e.g., C, FORTRAN) for solving technical problems. MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. Specific applications are collected in packages referred to as toolbox. There are tool boxes for signal processing, symbolic computation, control theory, simulation, optimization, and several other fields of applied science and engineering.

**Modules and their functionality**

plt.imshow(gray\_img, cmap='gray')

cv2.imshow('Test Imag', gray\_img)

cv2.waitKey(0)

cv2.destroyAllWindows()

* **plt.imshow(img, color\_map):**This is a matplotlib function used to display an image. It takes two arguments; the first one is the image you want to post and the second is the colormap (gray, RGB) in which the image is in.
* **cv2.imshow(window\_name, image):** This is a cv2 function used to display the image. It also takes two arguments: the first one is the name of the window that will pop-up to show the picture and the second one is the image you want to display.
* **cv2.waitKey():** This is a keyboard binding function, which takes one argument: (x) time in milliseconds. The function delays for (x) milliseconds any keyboard event. If (0) is pressed, it waits indefinitely for a keystroke, if any other key is pressed the program continues.
* **cv2.destroyAllWindows():** This simply destroys all the windows we created using cv2.imshow(window\_name, image)

**Importing required libraries from OpenCV**

Let's import the necessary libraries first. Remember, the names of these libraries are self-descriptive so you can put 2 and 2 together.

#import required libraries

#import OpenCV library

import cv2

#import matplotlib library

import matplotlib.pyplot as plt

When you load an image using OpenCV, it loads it into BGR color space by default. To show the colored image using matplotlib we have to convert it to RGB space. The following is a helper function to do exactly that:

def convertToRGB(img):

    return cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

**cv2.cvtColor** is an OpenCV function to convert images to different color spaces. It takes as input an image to transform, and a color space code (like cv2.COLOR\_BGR2RGB) and returns the processed image.

To read/load our image and convert it to grayscale, I used OpenCV’s built in function cv2.imread(img\_path) and passed our image path as the input parameter.

#load test iamge

test1 = cv2.imread('data/test1.jpg')

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

gray\_img = cv2.cvtColor(test1, cv2.COLOR\_BGR2GRAY)

To display our image, I’ll use the plt.imshow(img, cmap) function of matplotlib.

#if you have matplotlib installed then

plt.imshow(gray\_img, cmap='gray')

# or display the gray image using OpenCV

# cv2.imshow('Test Imag', gray\_img)

# cv2.waitKey(0)

# cv2.destroyAllWindows()

load cascade classifier training file for haarcascade

haar\_face\_cascade = cv2.CascadeClassifier('data/haarcascade\_frontalface\_alt.xml')

* **detectMultiScale(image, scaleFactor, minNeighbors):** This is a general function to detect objects, in this case, it'll detect faces since we called in the face cascade. If it finds a face, it returns a list of positions of said face in the form “Rect(x,y,w,h).”, if not, then returns “None”.
* **Image:** The first input is the grayscale image. So make sure the image is in grayscale.
* **scaleFactor:** This function compensates a false perception in size that occurs when one face appears to be bigger than the other simply because it is closer to the camera.
* **minNeighbors:** This is a detection algorithm that uses a moving window to detect objects, it does so by defining how many objects are found near the current one before it can declare the face found.

The following code will try to detect a face from the image and, if detected, it will print the number of faces that it has found, which in our case should be 1. Only 1 since no other spiritual being is out there. Right?…

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

faces = haar\_face\_cascade.detectMultiScale(gray\_img, scaleFactor=1.1, minNeighbors=5);

#print the number of faces found

print('Faces found: ', len(faces))

Next, let's loop over the list of faces (rectangles) it returned and drew those rectangles using yet another built-in OpenCV **rectangle** function on our original colored image to see if it found the right faces:

#go over list of faces and draw them as rectangles on original colored

img for (x, y, w, h) in faces:

    cv2.rectangle(test1, (x, y), (x+w, y+h), (0, 255, 0), 2)

Let’s display the original image to see the rectangles we just drew and verify that detected faces are real ones and not any false positives.

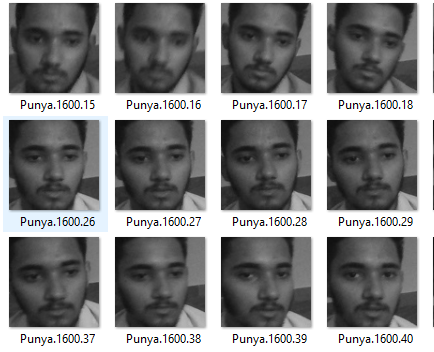
#convert image to RGB and show image

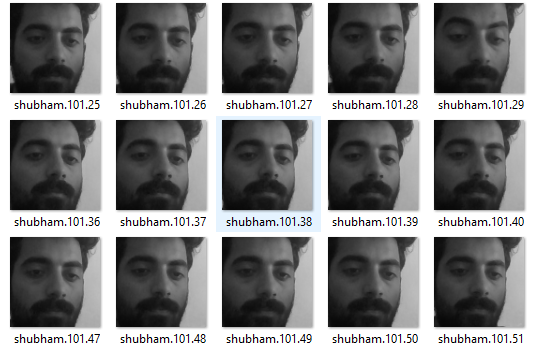
**Chapter 3**

**Training Images**

****

****

****

****

**Chapter 5**

**Conclusion**

The computational models, which were implemented in this project, were chosen after extensive research, and the successful testing results confirm that the choices made by the researcher were reliable.The system with manual face detection and automatic face recognition did not have a recognition accuracy over 90%, due to the limited number of eigenfaces that were used for the PCA transform. This system was tested under very robust conditions in this experimental study and it is envisaged that real-world performance will be far more accurate.The fully automated frontal view face detection system displayed virtually perfect accuracy and in the researcher's opinion further work need not be conducted in this area. The fully automated face detection and recognition system was not robust enough to achieve a high recognition accuracy. The only reason for this was the face recognition subsystem did not display even a slight degree of invariance to scale, rotation or shift errors of the segmented face image. This was one of the system requirements identified in section 2.3. However, if some sort of further processing, such as an eye detection technique, was implemented to further normalise the segmented face image, performance will increase to levels comparable to the manual face detection and recognition system. Implementing an eye detection technique would be a minor extension to the implemented system and would not require a great deal of additional research.All other implemented systems displayed commendable results and reflect well on the deformable template and Principal Component Analysis strategies.The most suitable real-world applications for face detection and recognition systems are for mugshot matching and surveillance. There are better techniques such as iris or retina recognition and face recognition using the thermal spectrum for user access and user verification applications since these need a very high degree of accuracy.The real-time automated pose invariant face detection and recognition system proposed in chapter seven would be ideal for crowd surveillance applications. If such a system were widely implemented its potential for locating and tracking suspects for law enforcement agencies is immense. The implemented fully automated face detection and recognition system (with an eye detection system) could be used for simple surveillance applications such as ATM user security, while the implemented manual face detection and automated recognition system is ideal of mugshot matching. Since controlled conditions are present when mugshots are gathered, the frontal view face recognition scheme should display a recognition accuracy far better than the results, which were obtained in this study, which was conducted under adverse conditions.

**Chapter 6**

**Bibliography**

[**www.google.com**](http://www.google.com)

[**www.geeksforgeeks.com**](http://www.geeksforgeeks.com)

[**www.numpy.org**](http://www.numpy.org)

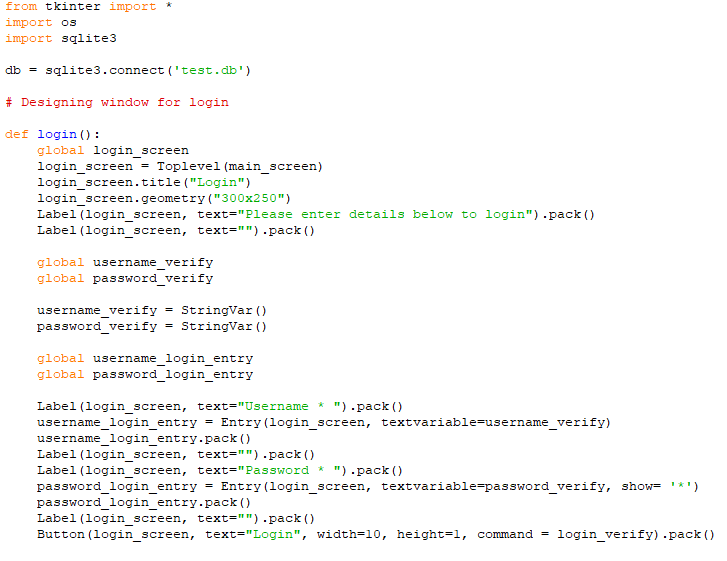
[**www.matplotlib.org**](http://www.matplotlib.org)

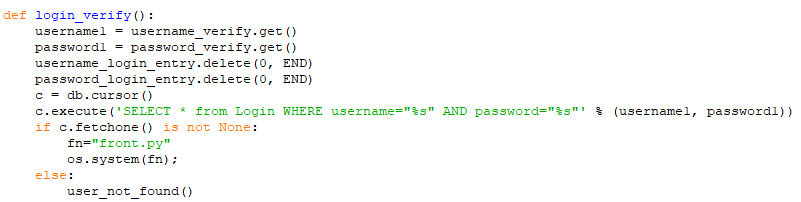
[**www.openCv.org**](http://www.openCv.org)

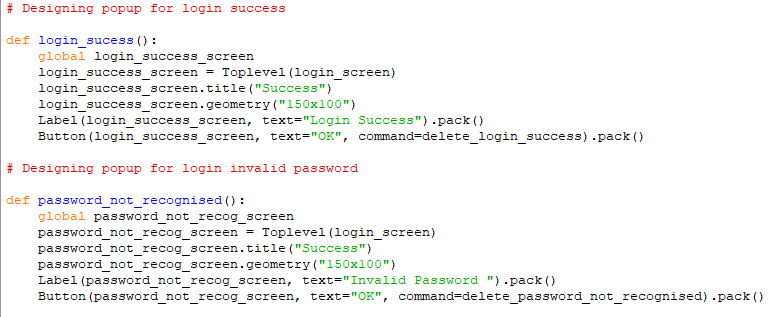
**Chapter 7**

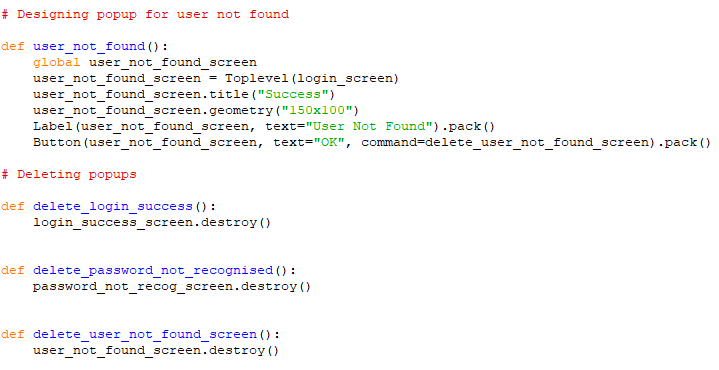
**Appendices**

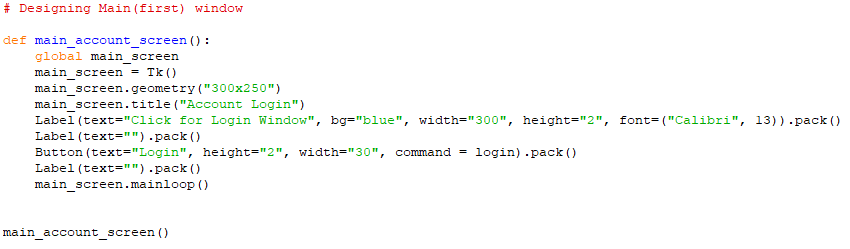
**login.py**

****

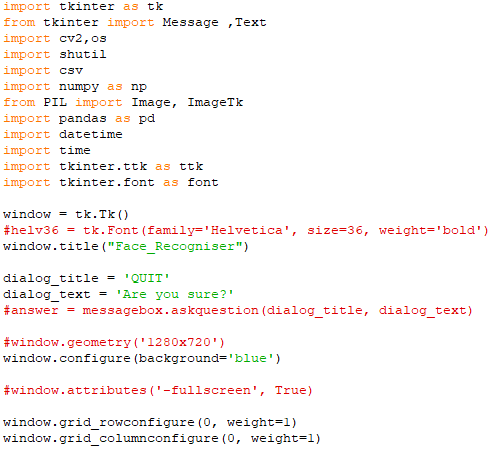
****

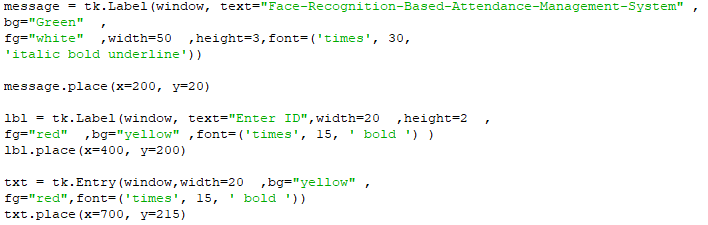
****

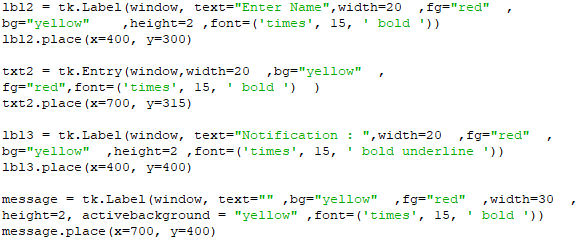


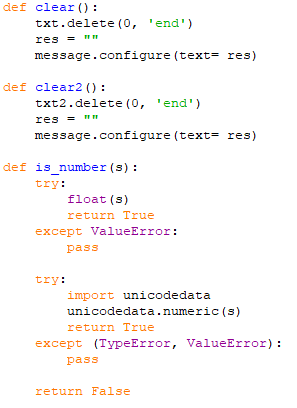


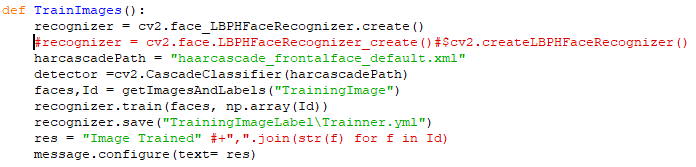
**front.py**

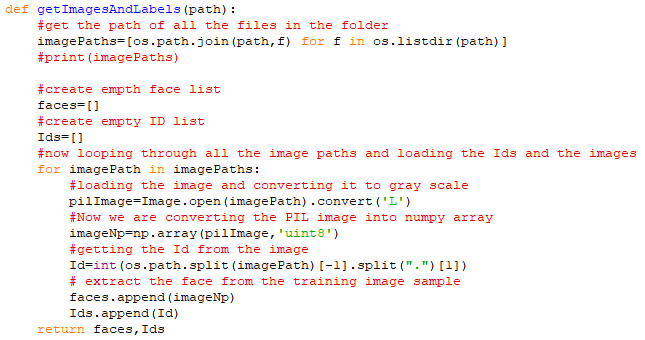
****

****

****

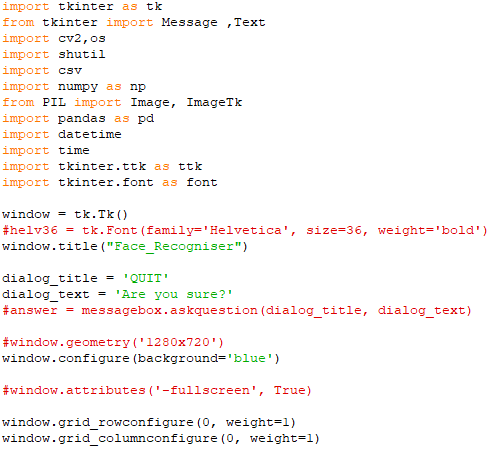
****

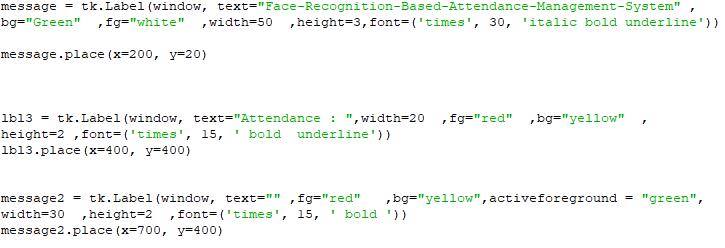
****

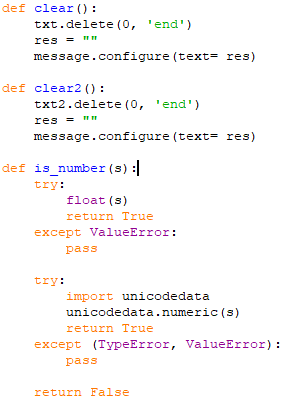
****

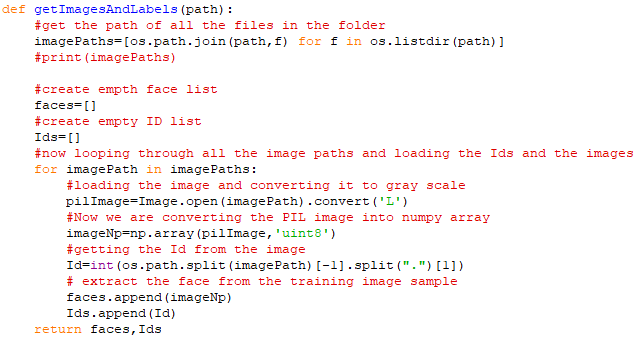
****

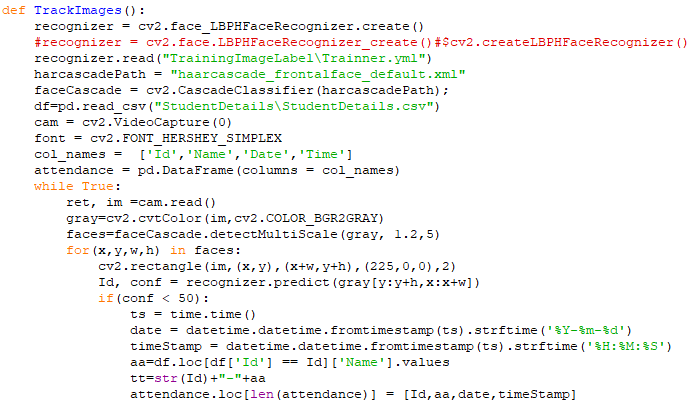
**Tk.py**

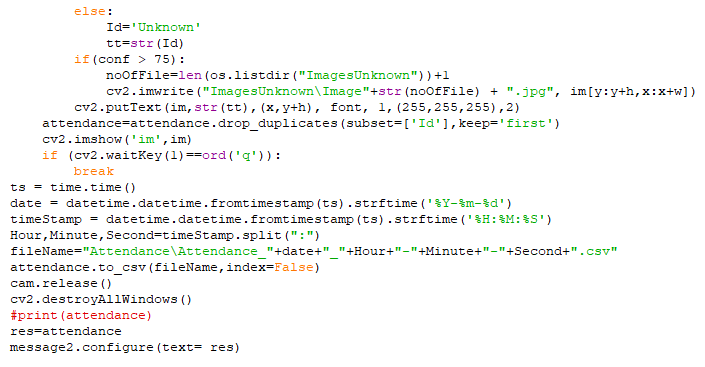
****

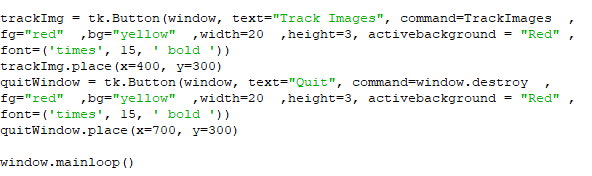
****

****

****

****

****

****