Real-Time Face Recognition

Group Members: Hrutuja Saswade (71), Harsh Singh (72), Arif Chaudhary (73)

ABSTRACT:-

The growing interest in computer vision of the past decade. Fueled by the steady doubling rate of computing power every 13 months, face detection and recognition has transcended from an esoteric to a popular area of research in computer vision and one of the better and successful applications of image analysis and algorithm based understanding. Because of the intrinsic nature of the problem, computer vision is not only a computer science area of research, but also the object of neuro-scientific and psychological studies, mainly because of the general opinion that advances in computer image processing and understanding research will provide insights into how our brain work and vice versa. Because of general curiosity and interest in the matter, the author has proposed to create an application that would allow user access to a particular machine based on an in-depth analysis of a person's facial features. This application will be developed using Intel's open source computer vision project, OpenCV and Microsoft's .NET framework.

INTRODUCTION:-

The goal of this project is to provide an easier human-machine interaction routine when user authentication is needed through face detection and recognition. With the aid of a regular web camera, a machine is able to detect and recognize a person’s face; a custom login screen with the ability to filter user access based on the users’ facial features will be developed. The objectives of this thesis are to provide a set of detection algorithms that can be later packaged in an easily portable framework amongst the different processor architectures we see in machines (computers) today. These algorithms must provide at least a 95% successful recognition rate, out of which less than 3% of the detected faces are false positives.

Face recognition is an important research problem spanning numerous fields and disciplines. This because face recognition, in additional to having numerous practical applications such as bankcard identification, access control, Mug shots searching, security monitoring, and surveillance system, is a fundamental human behavior that is essential for effective communications and interactions among people. A formal method of classifying faces was first proposed in. The author proposed collecting facial profiles as curves, finding their norm, and then classifying other profiles by their deviations from the norm. This classification is multi-modal, i.e. resulting in a vector of independent measures that could be compared with other vectors in a database. Progress has advanced to the point that face recognition systems are being demonstrated in real-world settings. The rapid development of face recognition is due to a combination of factors: active development of algorithms, the availability of a large databases of facial images, and a method for evaluating the performance of face recognition algorithms.

Problem Defination:

Over the past decade face detection and recognition have transcended from esoteric to popular areas of research in computer vision and one of the better and successful applications of image analysis and algorithm based understanding. Because of the intrinsic nature of the problem, computer vision is not only a computer science area of research, but also the object of neuro-scientific and psychological studies also, mainly because of the general opinion that advances in computer image processing and understanding research will provide insights into how our brain work and vice versa. A general statement of the face recognition problem (in computer vision) can be formulated as follows: given still or video images of a scene, identify or verify one or more persons in the scene using a stored database of faces. Facial recognition generally involves two stages: Face Detection where a photo is searched to find a face, then the image is processed to crop and extract the person's face for easier recognition. Face Recognition where that detected and processed face is compared to a database of known faces, to decide who that person is. Since 2002, face detection can be performed fairly easily and reliably with Intel's open source framework called OpenCV. This framework has an in-built Face Detector that works in roughly 90-95% of clear photos of a person looking forward at the camera. However, detecting a person's face when that person is viewed from an angle is usually harder, sometimes requiring 3D Head Pose Estimation. Also, lack of proper brightness of an image can greatly increase the difficulty of detecting a face, or increased contrast in shadows on the face, or maybe the picture is blurry, or the person is wearing glasses, etc. Face recognition however is much less reliable than face detection, with an accuracy of 30-70% in general. Face recognition has been a strong field of research since the 1990s, but is still a far way away from a reliable method of user authentication. More and more techniques are being developed each year. The Eigenface technique is considered the simplest method of accurate face recognition, but many other (much more complicated) methods or combinations of multiple methods are slightly more accurate. OpenCV was started at Intel in 1999 by Gary Bradski for the purposes of accelerating research in and commercial applications of computer vision in the world and, for Intel, creating a demand for ever more powerful computers by such applications. Vadim Pisarevsky joined Gary to manage Intel's Russian software OpenCV team. Over time the OpenCV team moved on to other companies and other Research. Several of the original team eventually ended up working in robotics and found their way to Willow Garage. In 2008, Willow Garage saw the need to rapidly advance robotic perception capabilities in an open way that leverages the entire research and commercial community and began actively supporting OpenCV, with Gary and Vadim once again leading the effort. Intel's open-source computer-vision library can greatly simplify computer-vision programming. It includes advanced capabilities - face detection, face tracking, face recognition, Kalman filtering, and a variety of artificial-intelligence (AI) methods - in ready-to-use form. In addition, it provides many basic computer-vision algorithms via its lower-level APIs. OpenCV has the advantage of being a multi-platform framework; it supports both Windows and Linux, and more recently, Mac OS X. OpenCV has so many capabilities it can seem overwhelming at first. A good understanding of how these methods work is the key to getting good results when using OpenCV. Fortunately, only a select few need to be known beforehand to get started. OpenCV's functionality that will be used for facial recognition is contained within several modules. Following is a short description of the key namespaces:

CXCORE namespace contains basic data type definitions, linear algebra and statistics methods, the persistence functions and the error handlers. Somewhat oddly, the graphics functions for drawing on images are located here as well. CV namespace contains image processing and camera calibration methods. The computational geometry functions are also located here. CVAUX namespace is described in OpenCV's documentation as containing obsolete and experimental code. However, the simplest interfaces for face recognition are in this module. The code behind them is specialized for face recognition, and they're widely used for that purpose. ML namespace contains machine-learning interfaces. HighGUI namespace contains the basic I/O interfaces and multi-platform windowing capabilities. CVCAM namespace contains interfaces for video access through DirectX on 32-bit Windows platforms.

LITERATURE SURVEY:-

This section gives an overview on the major human face recognition techniques that apply mostly to frontal faces, advantages and disadvantages of each method are also given. The methods considered are eigenfaces (eigenfeatures), neural networks, dynamic link architecture, hidden Markov model, geometrical feature matching, and template matching. The approaches are analyzed in terms of the facial representations they used. Literature review of face recognition techniques this section gives an overview on the major human face recognition techniques that apply mostly to frontal faces, advantages and disadvantages of each method are also given. The methods considered are eigenfaces (eigenfeatures), neural networks, dynamic link architecture, hidden Markov model, geometrical feature matching, and template matching. The approaches are analyzed in terms of the facial representations they used.

OpenCV was started at Intel in 1999 by Gary Bradski for the purposes of accelerating research in and commercial applications of computer vision in the world and, for Intel, creating a demand for ever more powerful computers by such applications. Vadim Pisarevsky joined Gary to manage Intel's Russian software OpenCV team. Over time the OpenCV team moved on to other companies and other Research. Several of the original team eventually ended up working in robotics and found their way to Willow Garage. In 2008, Willow Garage saw the need to rapidly advance robotic perception capabilities in an open way that leverages the entire research and commercial community and began actively supporting OpenCV, with Gary and Vadim once again leading the effort [2]. Intel's open-source computer-vision library can greatly simplify computer vision programming. It includes advanced capabilities - face detection, face tracking, face recognition, Kalman filtering, and a variety of artificial intelligence (AI) methods - in ready-touse form. In addition, it provides many basic computer-vision algorithms via its lower-level APIs. OpenCV has the advantage of being a multi-platform framework; it supports both Windows and Linux, and more recently, Mac OS X. OpenCV has so many capabilities it can seem overwhelming at first. A good understanding of how these methods work is the key to getting good results when using OpenCV. Fortunately, only a select few need to be known beforehand to get started. OpenCV's functionality that will be used for facial recognition is contained within several modules

SYSTEM DESCRIPTION:-

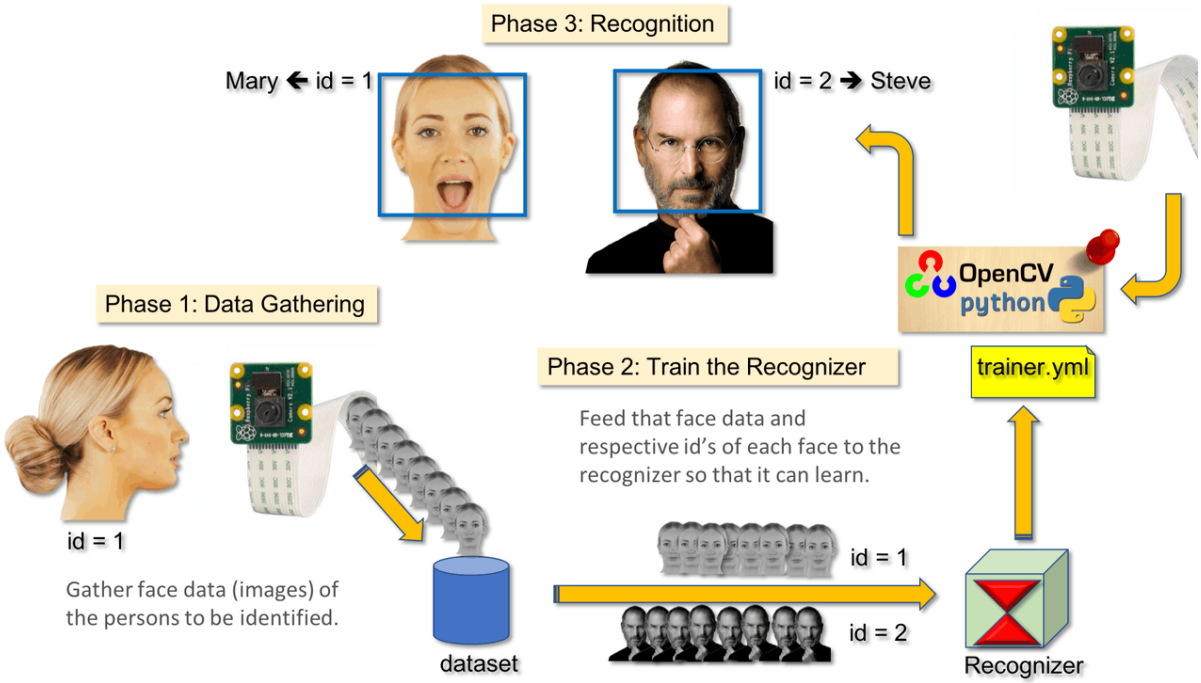
This project was done with this fantastic "Open Source Computer Vision Library", the OpenCV. On this project, we will be focusing on OpenCV and Python, but I also tested the code on My Mac and it also works fine. OpenCV was designed for computational efficiency and with a strong focus on real-time applications. So, it's perfect for real-time face recognition using a camera.

3 PHASES

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

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

The below block diagram resumes those phases:



Step 1: Face Detection

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

The most common way to detect a face (or any objects), is using the “Haar Clascade Classifier" Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

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

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

faceDetection.py

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

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

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

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

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

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

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

```
faces = faceCascade.detectMultiScale(
    gray,

    scaleFactor=1.2,
    minNeighbors=5
    ,
    minSize=(20, 20)
)
```

Where,

● **gray** is the input grayscale image.

● **scaleFactor** is the parameter specifying how much the image size is reduced at each image scale. It is used to create the scale pyramid.

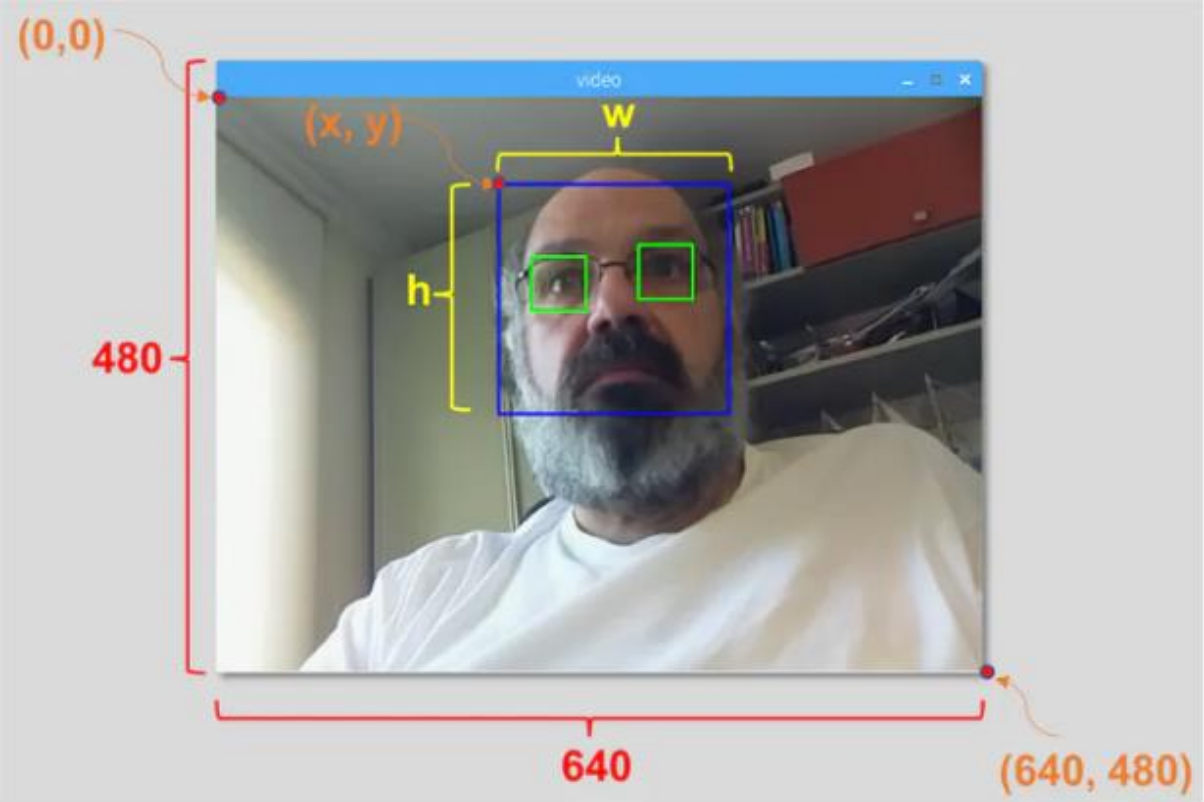
● **minNeighbors** is a parameter specifying how many neighbors each candidate rectangle should have, to retain it. A higher number gives lower false positives.

● **minSize** is the minimum rectangle size to be considered a face.

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

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

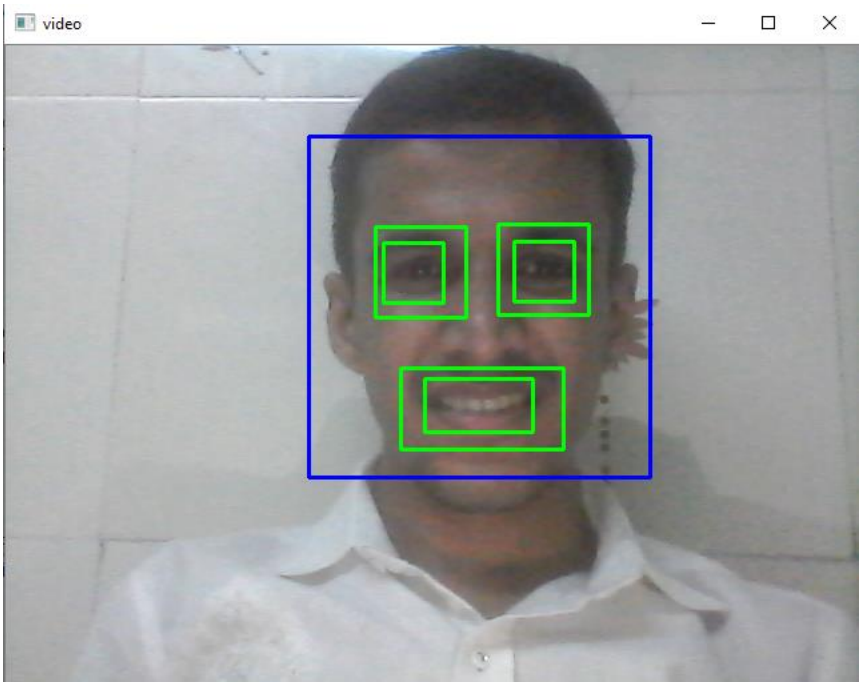
If faces are found, it returns the positions of detected faces as a rectangle with the left up corner (x,y) and having "w" as its Width and "h" as its Height ==> (x,y,w,h). Please see the picture.



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

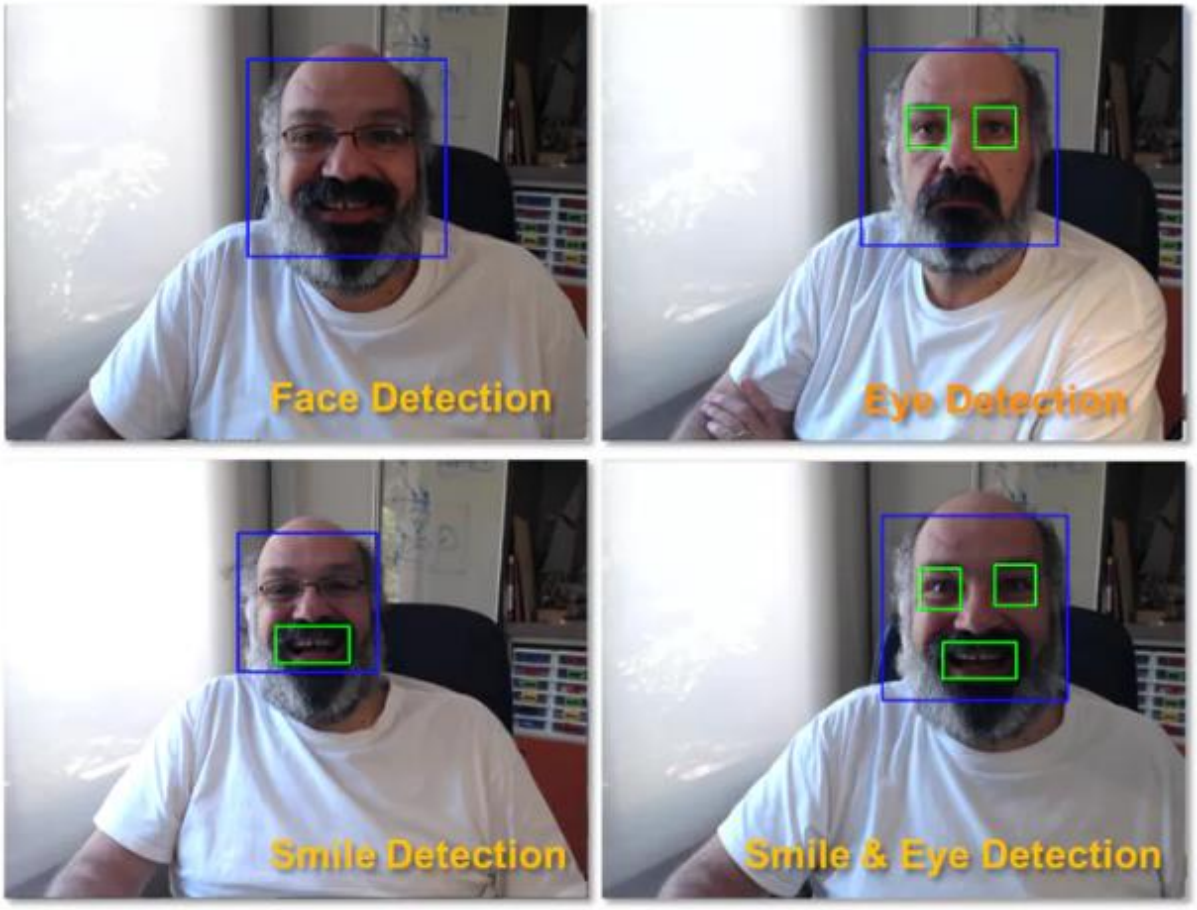
Run the module:

The Result:



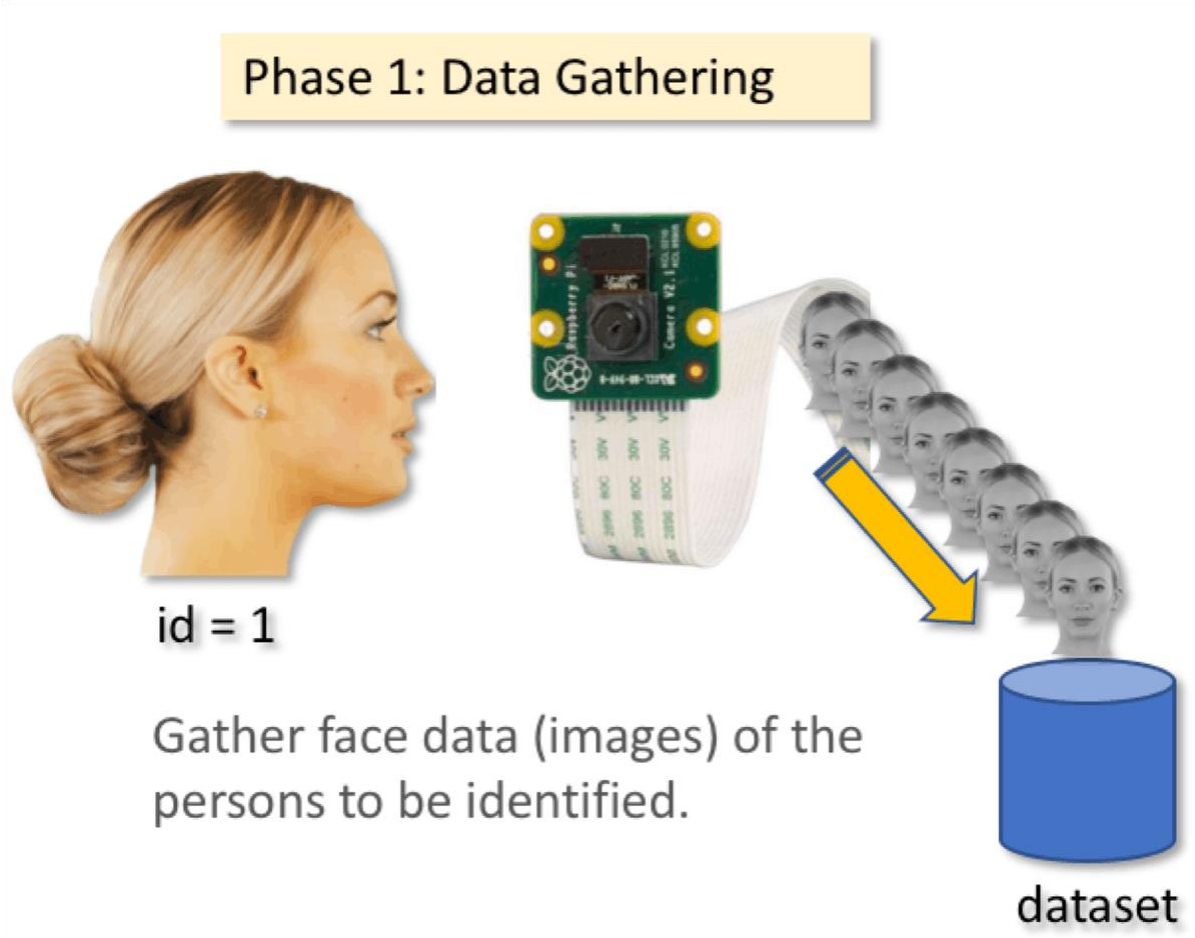
We also included classifiers for "eyes detection" or even "smile detection". On those cases, we will include the classifier function and rectangle draw inside the face loop, because would be no sense to detect an eye or a smile outside of a face.

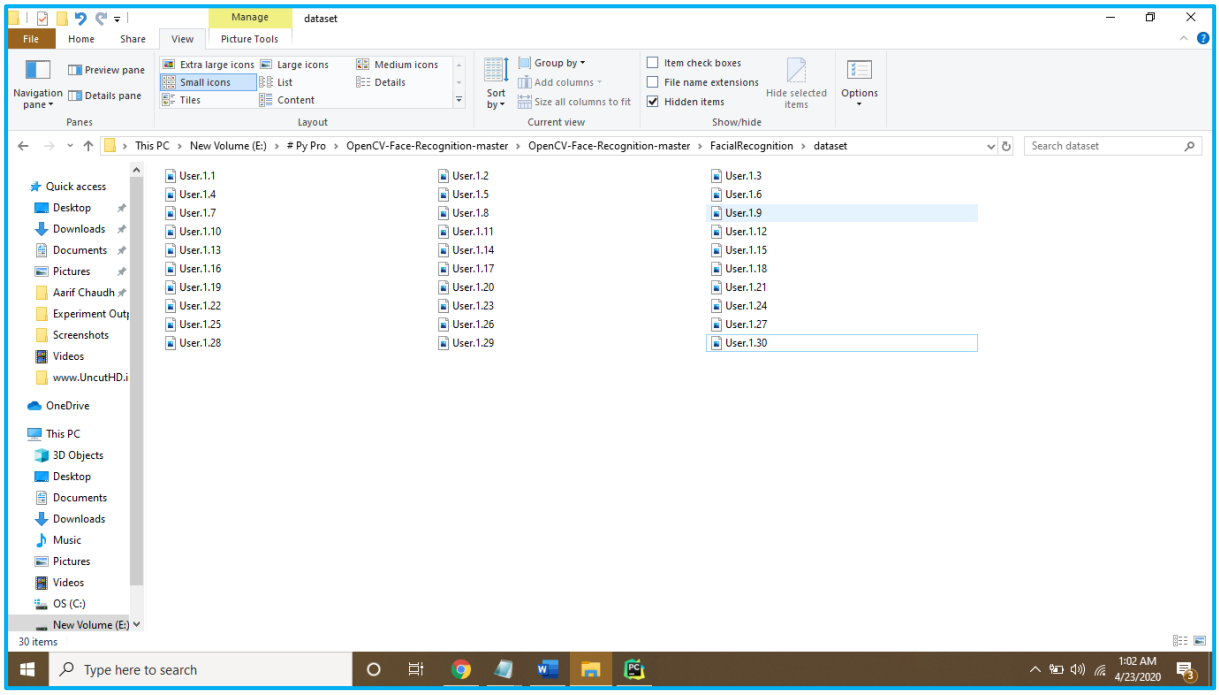
And in the picture, you can see the result.



Step 2: DATA GATHERING

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



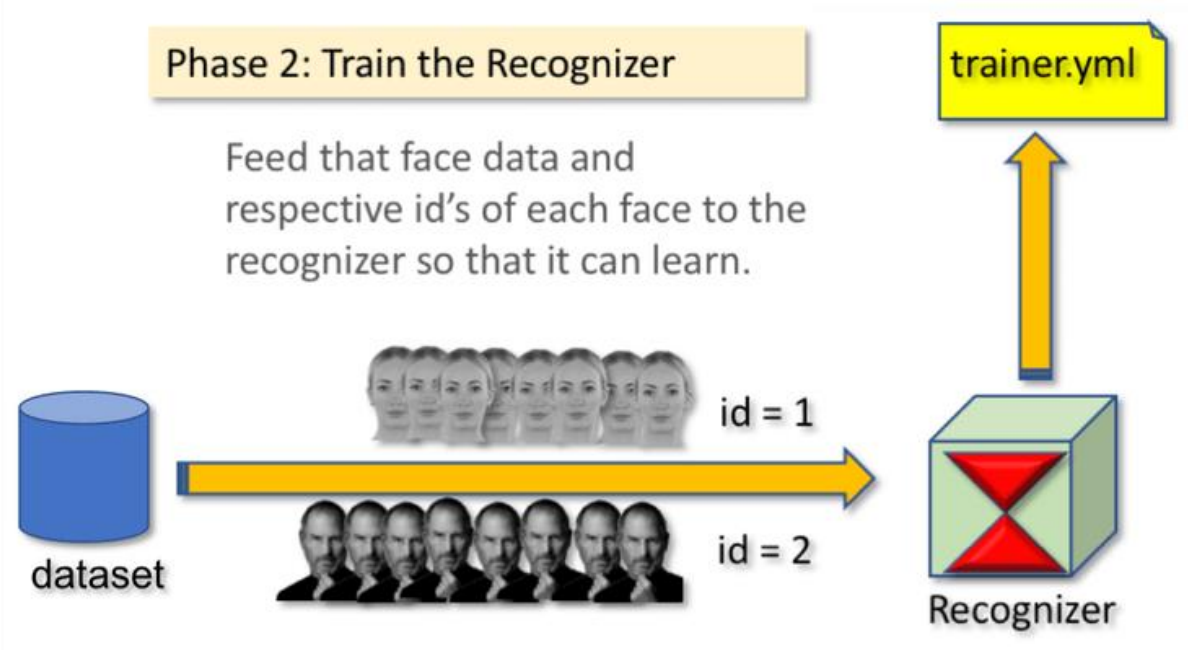


On my code, I am capturing 30 samples from each id. We can change it on the last "elif". The number of samples is used to break the loop where the face samples are captured.

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

STEP 3: TRAINER:-

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



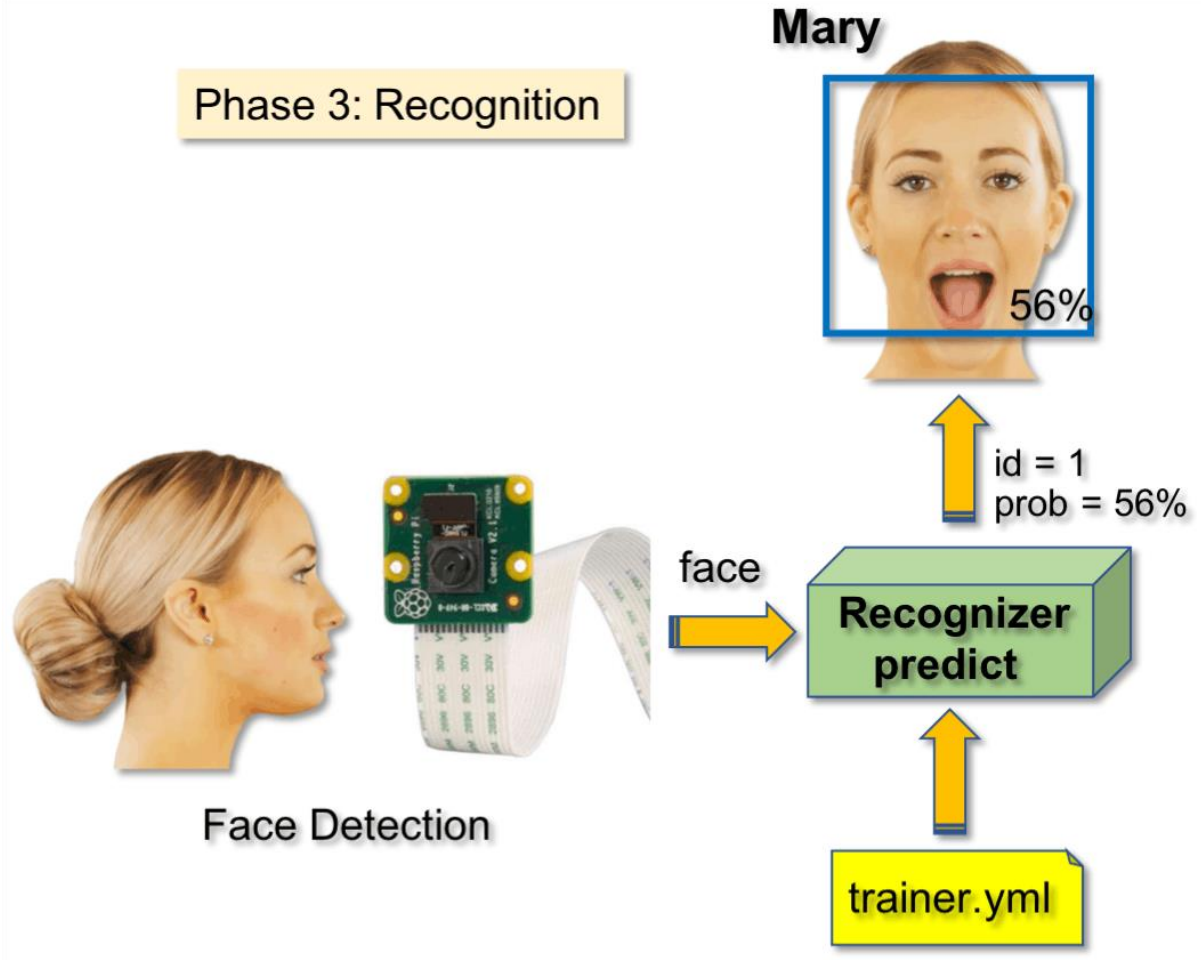
As a result, a file named "trainer.yml" will be saved in the trainer directory that was previously created by us.

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

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

Step 4: RECOGNITION

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



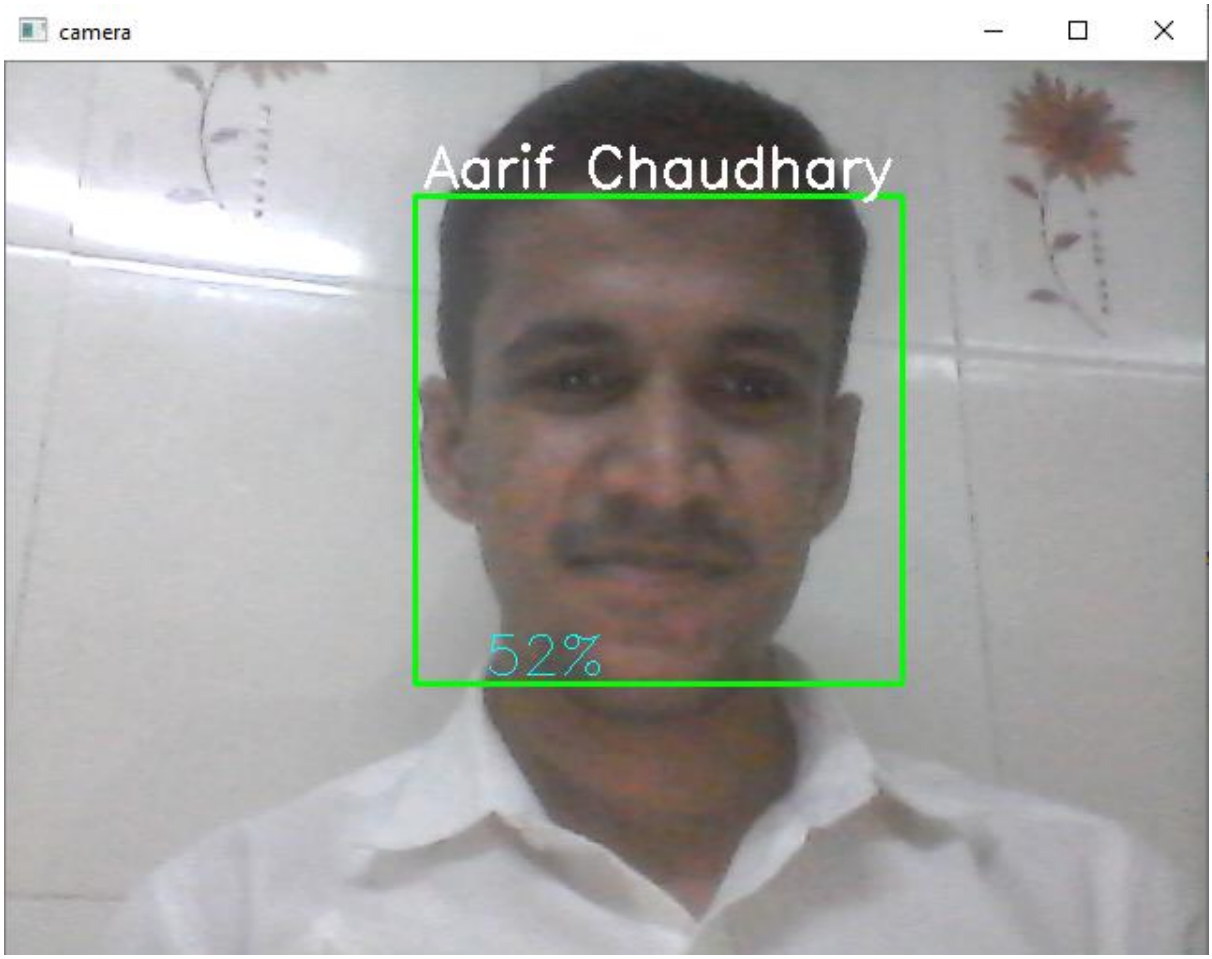
Next, we will detect a face, same we did before with the haasCascade classifier. Having a detected face we can call the most important function in the above code:

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

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

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

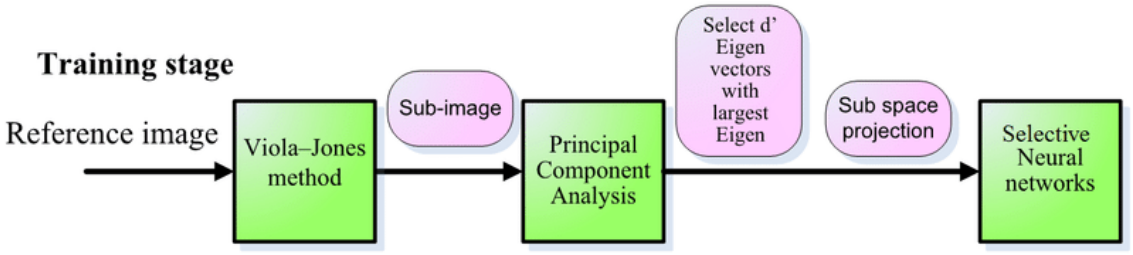
Below is the result:

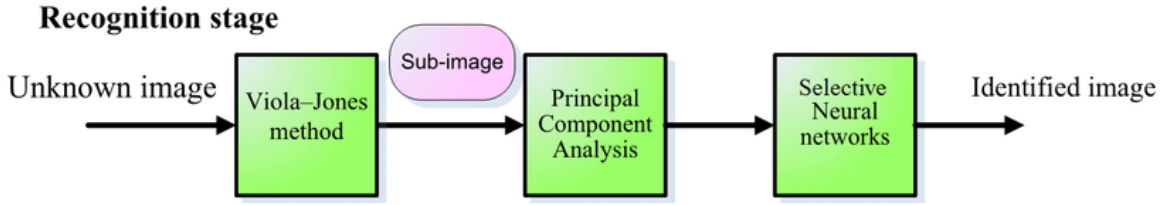


On the picture, I show some tests done with this project, where I also have used photos to verify if the recognizer works.

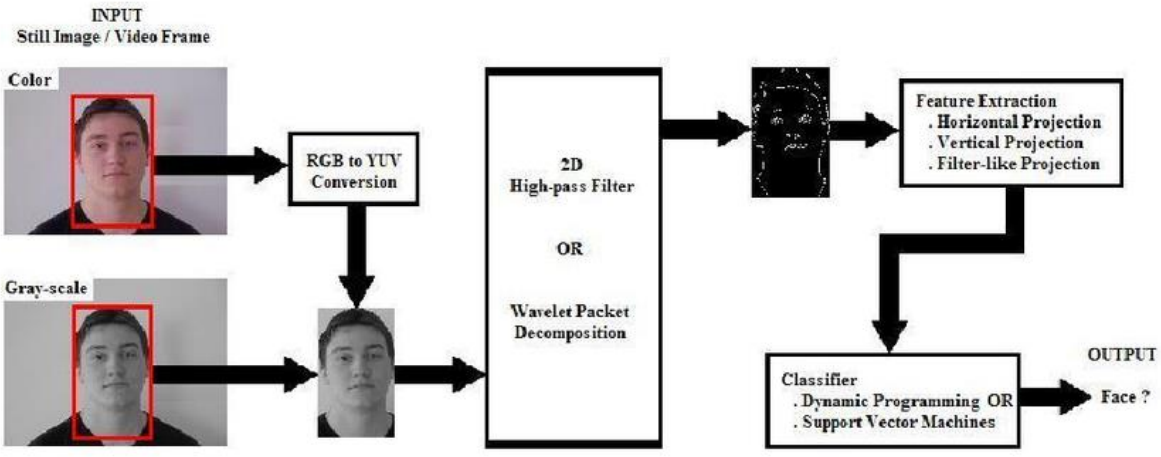
Block Diagram:

1] FACE RECOGNITION:-

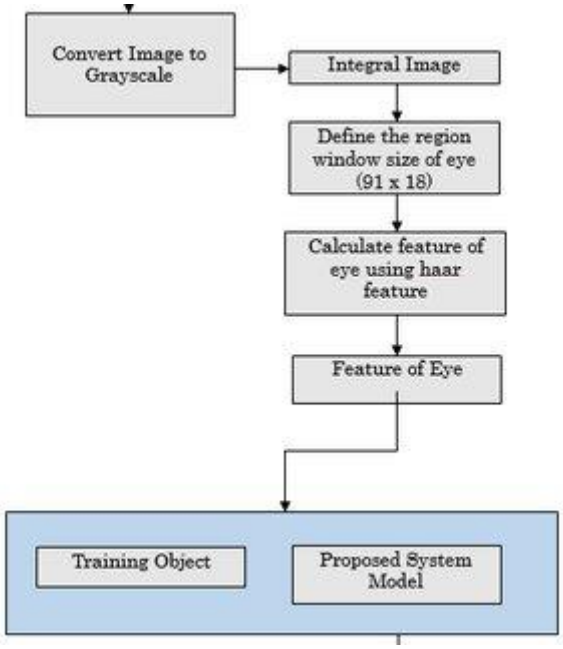




2] FACE DETECTION:-



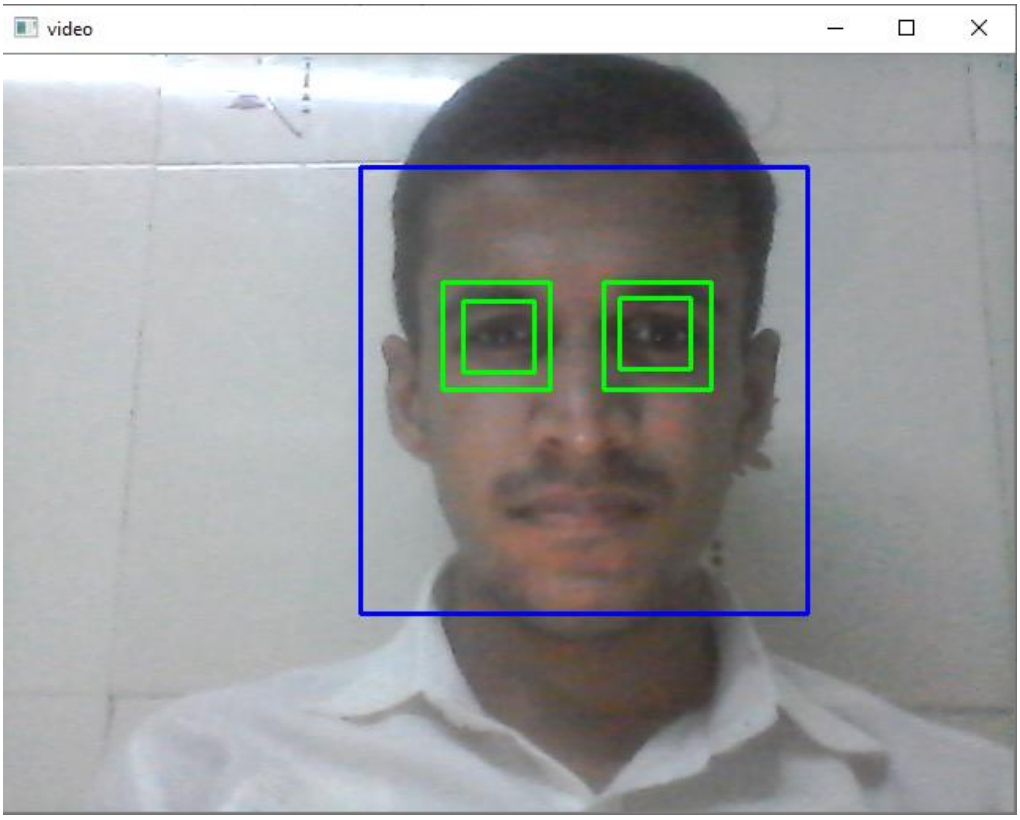
3] EYE RECOGNITION:-



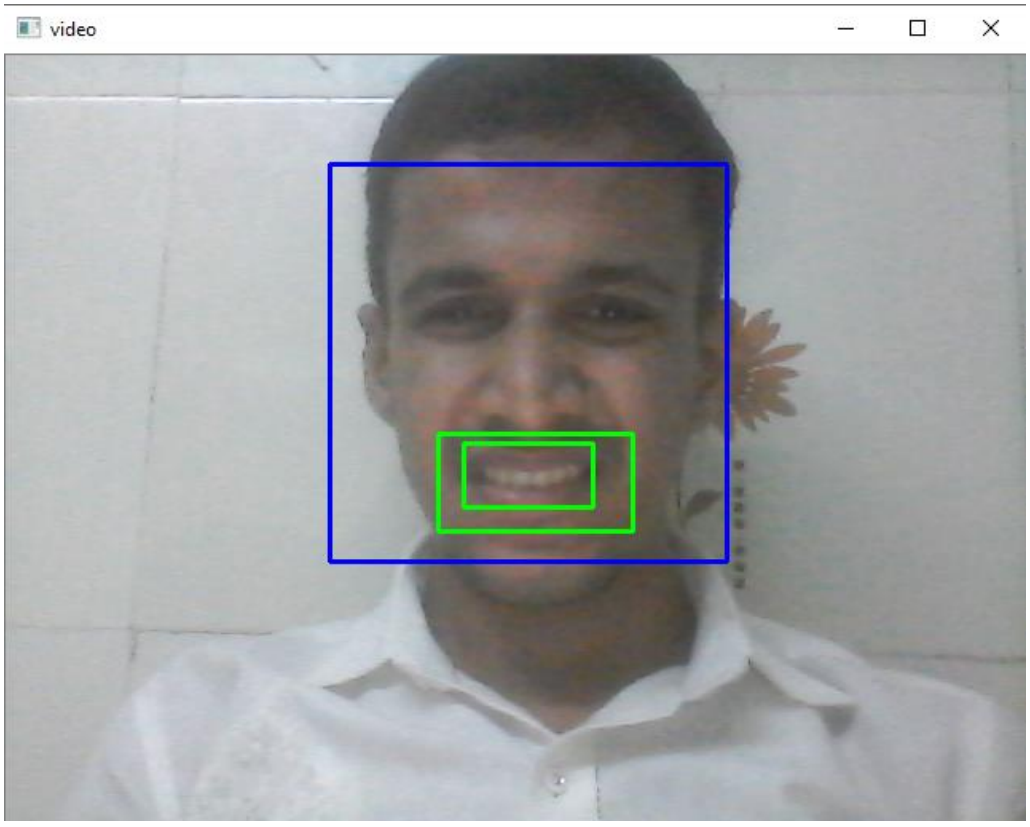
RESULT & DISCUSSION:

Face Detection:

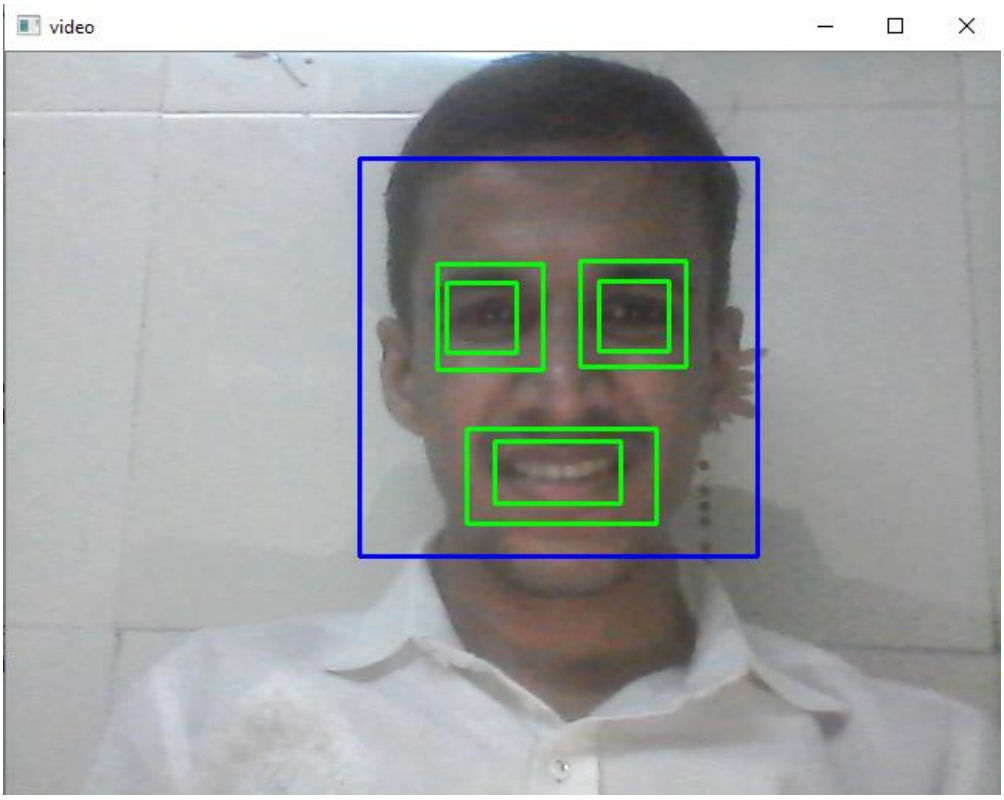
Eye Detection:



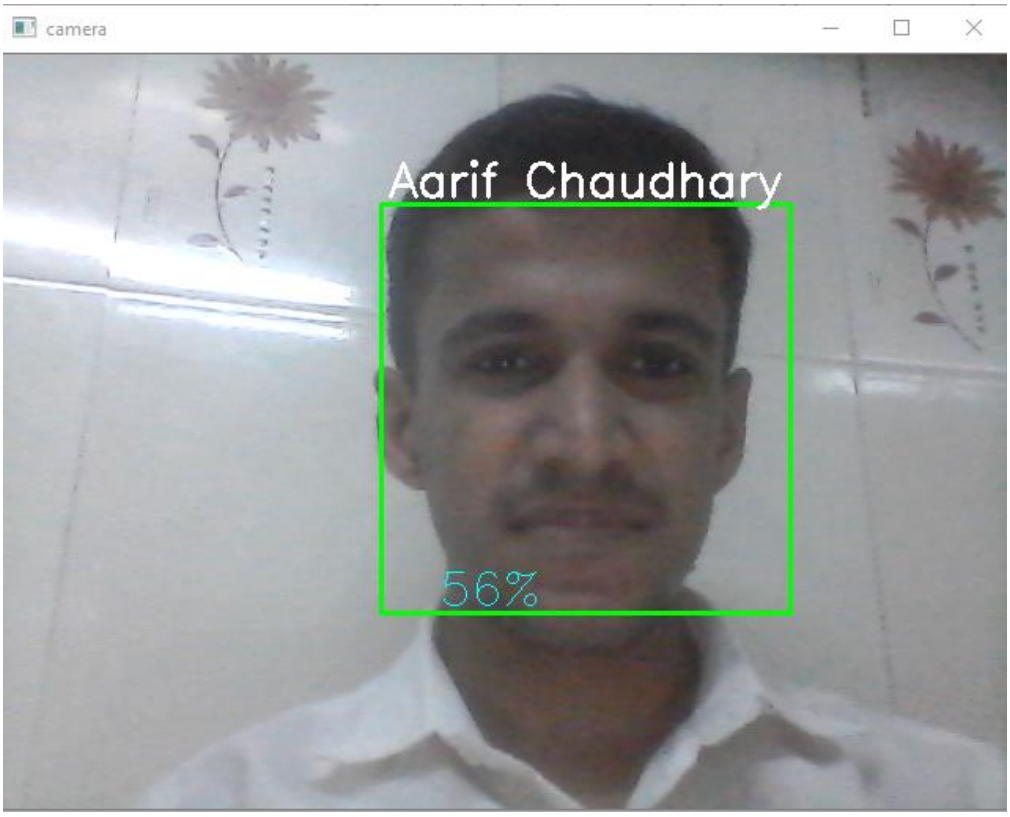
Smile Detection:



Face Eye Smile Detection:



Face Regognition:



CONCLUSION:-

This project describes the mini-project for visual perception and autonomy module. Next, it explains the technologies used in the project and the methodology used. Finally, it shows the results, discuss the challenges and how they were resolved followed by a discussion. Using Haar-cascades for face detection worked extremely well even when subjects wore spectacles. Real time video speed was satisfactory as well devoid of noticeable frame lag. Considering all factors, LBPH combined with Haar-cascades can be implemented as a cost effective face recognition platform. An example is a system to identify known troublemakers in a mall or a supermarket to provide the owner a warning to keep him alert or for automatic attendance taking in a class.

FUTURE SCOPE:-

The use of spherical canonical images allows us to perform matching in the spherical harmonic transform domain, which does not require preliminary alignment of the images. The errors introduced by embedding into an expressional space with some predefined geometry are avoided. In this facial expression recognition setup, end-to-end processing comprises the face surface acquisition and reconstruction, smoothening, sub sampling to approximately 2500 points. Facial surface cropping measurement of large positions of distances between all the points using a parallelized parametric version is utilized.

The general experimental evaluation of the face expressional system guarantees better face recognition rates. Having examined techniques to cope with expression variation, in future it may be investigated in more depth about the face classification problem and optimal fusion of color and depth information. Further study can be laid down in the direction of allele of gene matching to the geometric factors of the facial expressions. The genetic property evolution framework for facial expressional system can be studied to suit the requirement of different security models such as criminal detection, governmental confidential security breaches etc.

REFERENCES:-

[1] Francis Galton, “Personal identification and description,” In Nature, pp. 173-177, June 21, 1888. [2] W. Zaho, “Robust image based 3D face recognition,” Ph.D. Thesis, Maryland University, 1999.

[3] R. Chellappa, C.L. Wilson and C. Sirohey, ”Humain and machine recognition of faces: A survey,” Proc. IEEE, vol. 83, no. 5, pp. 705- 740, may 1995.

[4] T. Fromherz, P. Stucki, M. Bichsel, “A survey of face recognition,” MML Technical Report, No 97.01, Dept. of Computer Science, University of Zurich, Zurich, 1997.

[5] T. Riklin-Raviv and A. Shashua, “The Quotient image: Class based recognition and synthesis under varying illumination conditions,” In CVPR, P. II: pp. 566-571,1999.

[6] G.j. Edwards, T.f. Cootes and C.J. Taylor, “Face recognition using active appearance models,” In ECCV, 1998.