Major Project Report

On

Accident prevention system using facial analysis based on SVM algorithm

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

Bachelor of Technology In

Information Technology

Guide:
Dr. Anjali Kapoor

Submitted by:
Ritvik Shandilya(02410403115)
Ayush Kumar(40410403115)

Department of Information Technology Amity School of Engineering and Technology

(Affiliated to Guru Gobind Singh Indraprastha University, New Delhi)

(2015-2019)

CERTIFICATE

This is to certify that the major project entitled "***Accident prevention system using facial analysis based on SVM algorithm***" submitted by Ritvik Shandilya (Roll no. 02410403115) and Ayush Kumar(40410403115) has been accomplished under my guidance. This is submitted for partial fulfilment for the award of a degree in Bachelor of Technology in Information Technology at Amity School of Engineering and Technology, affiliated to Guru Gobind Singh Indraprastha University, Delhi.

Date: April 11, 2019

Dr. Anjali Kapoor
CSE/IT Department
Amity School of
Engineering and
Technology, Delhi

ACKNOWLEDGEMENT

I am thankful to **Prof. Dr. Rekha Aggarwal**, Director for her moral support and providing me with good infrastructure facilities of the institute which helped me throughout the journey.

It gives me immense pleasure to express my deepest sense of gratitude and sincere thanks to our respected **Dr. Pinki Nayak**, HOD, Department of CSE/IT and esteemed guide, **Dr. Anjali Kapoor**, Assistant Professor, Department of CSE / IT for their valuable guidance, encouragement and help for completing this work. I am also thankful to all the staff members of the Department of CSE/IT for their help in our work.

At the end, I would like to express my sincere thanks to all others who helped me directly or indirectly during this major project.

Accident prevention system using facial analysis based on SVM algorithm

Abstract:

In recent years, accidents caused due to fatigue in drivers is one of the most common reasons for driver fatalities. An efficient method of calculating driver fatigue is by measuring the level of indicators that might suggest the presence of drowsiness. So it's integral to know if the driver is suitable to continue driving for the well being of life and properties around. This project aims to develop a system that can notify the driver if symptoms of drowsiness or fatigue are detected. This system is a real-time system that captures images continuously and measures the state of the eyes, mouth and head pose according to the specified algorithm and gives a warning if required.

There are multiple methods to take the measure of fatigue or drowsiness in the driver; this approach is non-intrusive and does not cause discomfort to the driver in any way, hence giving the exact condition of the driver. For detection of drowsiness, the per-closure value of eyes, opening of mouth and pose of the head is considered in real-time. So when the closure of eye or mouth or head pose estimates exceeds a particular value, the driver is identified as unsuitable for driving and is alerted with noise. Several OpenCV libraries are used for implementing this system, including haar-cascade and facial features trained with SVM. The entire system is implemented using a standard portable computer and can be further done using a Raspberry Pi to make it a cheap solution for people to use and protect themselves from any possible accidents.

Department of Information Technology
Amity School of Engineering and Technology
(Affiliated to G.G.S.I.P. University,New Delhi)

(2015-2019)

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

Understanding the Problem

1.1 Introduction

Drivers' attention levels deteriorate as a result of insufficient sleep, lengthy periods of continuous driving, or other medical conditions such as brain illnesses. According to several studies on road accidents, driver weariness is responsible for roughly 30% of all collisions. Excessive exhaustion and tiredness are induced when a motorist drives for longer than is typical for a human. This causes the driver to become sleepy or lose consciousness.

Drowsiness is a complicated phenomena characterised by a reduction in the driver's alertness and consciousness. Although there is no direct way to detect sleepiness, there are various indirect approaches that can be applied.

In Chapter 1, numerous sorts of methods for evaluating a driver's sleepiness are discussed, including vehicle-based assessments, physiological measures, and behavioural measures. Using these principles, an intelligence system might be created that would inform the driver if they were sleepy, preventing accidents. Each system's pros and downsides are discussed. The most appropriate strategy is picked and offered based on advantages and disadvantages. The full system development technique is then illustrated via a flow chart, which involves continually recording the image in real time and then separating it into frames. Then each frame is examined to see which face appears first. If a face is found, the next step is to find the eyes and lips. Following a positive detection eye result, the quantity of eye closure is assessed and compared to reference values for the sleepy state of the eye, and the same is done with the mouth. If drowsiness is detected by the system, the driver is alerted; otherwise, the cycle of locating face and detecting drowsiness is repeated.

The PCA is used in the Eigenface technique to recognise the pictures. The method works by projecting a face picture that has already been retrieved onto a set of face space that reflects substantial variances among known face images. Eigenvalues and eigenvectors, face image representation, mean and mean centered images, covariance matrix, and eigenface space are all part of the eigenface method.

The implementation phase is discussed in Chapter 4. The Raspberry Pi is the system's main piece of hardware. As a result, a quick introduction to the Raspberry Pi is provided, followed by a detailed discussion of the theoretical approach to sleepiness detection, which comprises Haar Cascade, constructing Integral Image, Adaboost, and Cascading. All four approaches mentioned above were utilised to determine the status of the eyes and create an algorithm for it. The proposed approach was then implemented using suitable code. The Raspberry Pi was correctly configured for use. Several people were used to record the system's response and operation. Circular shapes denoted the opening of the eyes. If drowsiness is detected, the circle does not show, suggesting the driver's eye closure or drowsiness. Several images were used to demonstrate the results, which included both open and closed eyes.

In Chapter 6, the system's flaws were discussed, as well as the work that will be necessary in the future to eliminate those flaws and construct a strong intelligent driver aid system. Finally, the overall performance of the planned and realised system is discussed in the concluding section.

Chapter 2

Drowsiness and Measures of Detection

2.1 Drowsiness

Drowsiness is characterised as a reduced degree of consciousness manifested by tiredness and difficulty remaining awake, yet the person is awakened by simple stimuli. It might be caused by a lack of sleep, medication, substance abuse, or a neurological problem. It is mostly caused by exhaustion, which can be both mental and physical. Physical exhaustion, often known as muscular weariness, is the temporary inability of a muscle to work at its best. Mental weariness is a brief inability to maintain optimal psychological performance. The development of mental tiredness during any intellectual activity is gradual, and it is dependent on an individual's psychological capability as well as other factors such as lack of sleep and general well-being. [4] Physical performance has also been shown to be harmed by mental weariness. It might manifest as drowsiness, dormancy, or a lack of coordinated contemplation.

According to current data, driver fatigue has become one of the leading causes of fatalities, severe bodily injuries, and economic losses on the road in recent years. A motorist who falls asleep in the driver's seat is on the verge of losing control of the car, perhaps resulting in a collision with another vehicle or immovable objects. With the goal of reducing or eliminating the frequency of accidents, the driver's state of drowsiness should be monitored on a regular basis.

2.2 Measures for detection of Drowsiness

According to the study, the cause of a catastrophe may be classified into one of three categories: (1) human, (2) vehicle, and (3) environmental element. [8] 91 percent of the accidents were caused by driver error. The other two categories of causative aspects were referred to as 4 percent for vehicle type and 5 percent for environmental conditions. Drowsiness can be measured using a variety of methods, including the ones listed below:

i. **Vehicle based measures**
ii. **Physiological measures**
iii. **Behavioural measures**

1. Vehicle based measures.

Survey route position, which analyses the vehicle's location as it identifies with path markers, to evaluate driver weakness and collect steering wheel movement data to describe tiredness from low to high. [8] Researchers have employed this method to identify weariness in a variety of studies, showing the non-intrusive and cost-effective nature of this monitoring methodology.

This is done by:

i. Vehicle drifts out of the lane unexpectedly

ii. Steering wheels move suddenly

iii. Acceleration paddles are under pressure

Each measure has a set of threshold values that, when crossed, indicate that the driver is sleepy

Advantages:

i. It has a non-intrusive character

ii. Provides a near-perfect outcome

Disadvantages:

i. Vehicle-based measures are mostly influenced by road design, which might result in the alarm system being activated needlessly

ii. For the system to be efficient, the present driver's driving behaviour must be learnt and modelled

iii. Microsleeping, which is most common on straight roadways, is difficult to detect

2. Physiological measures.

Physiological metrics are objective assessments of the bodily changes that occur as a result of weariness in our bodies. [14] These physiological changes can be easily measured using the following instruments:

i. ECG (electrocardiogram)
ii. EMG (electromyogram)
iii. EOG (electro-oculogram)
iv. EEG (electroencephalogram)

Monitoring Heart Rate: ECG sensor can be fitted in a car's steering wheel to monitor a driver's pulse, which provides an indication of the driver's level of exhaustion and, hence, the condition of sleepiness. The ECG sensor can also be installed at the back of the seat.

Monitoring Brain Waves: Special hats with electrodes monitor brain waves to detect driver weariness and report the results in real time. Drowsiness can then be identified by categorising each brain wave.

Monitoring muscle fatigue: Drowsiness is closely connected to muscular tiredness. We know that when the pressure on the steering wheel decreases and the reaction of various muscles decreases, tiredness may be detected by installing pressure sensors on the steering wheel or monitoring the muscle response to applied stimuli.

Monitoring eye movements: Electro-oculogram can be used to quantify invasive eye movement and eye closure, but it will be highly difficult for the driver to deal with.

Though this approach provides the most precise sleepiness findings. However, it necessitates the installation of numerous electrodes on the driver's head, chest, and face, which is inconvenient. They must also be positioned with great care in their designated locations for a flawless result.

3. **Behavioural measures.**

Certain behavioural changes take place during drowsing like:

i. Yawning

ii. Amount of eye closure

iii. Blinking of eyes

iv. Position of head

2.3 Proposed Method

The most exact approach among these four relies on human physiological measurements [1]. This process is carried out in two ways: evaluating changes in physiological indications such as brain waves, heart rate, and eye flickering; and measuring physical alterations such as drooping posture, head inclination, and eye open/close conditions [1]. Despite the fact that this approach is the most exact, it is not practical since detecting electrodes would have to be placed directly on the driver's body, causing irritation and distraction. Long periods of driving can also cause perspiration to form on the sensors, decreasing their ability to screen accurately. As a result, this method will concentrate on the amount of eye closure, yawning, and head position in real time, as this offers the most reliable information on sleepiness. It is also non intrusive in nature, thus it has no effect on the driver's state of mind, and the motorist is completely at ease with it. This method is unaffected by environmental factors such as road conditions. According to the set threshold value, the case of micro sleep is also identified. Face recognition and tracking, human eye detection and location, human eye tracking, eye state detection, and driver tiredness testing are all part of the system's development. The detection framework's main components were the detection and positioning of human eyeballs, as well as driver tiredness testing. The revised approach for detecting tiredness by assessing the driver's Eye aspect ratio, Mouth aspect ratio, and head posture estimate has a high accuracy of more than 90% [19].

2.4 Drowsiness detection approach Flowchart of the proposed system

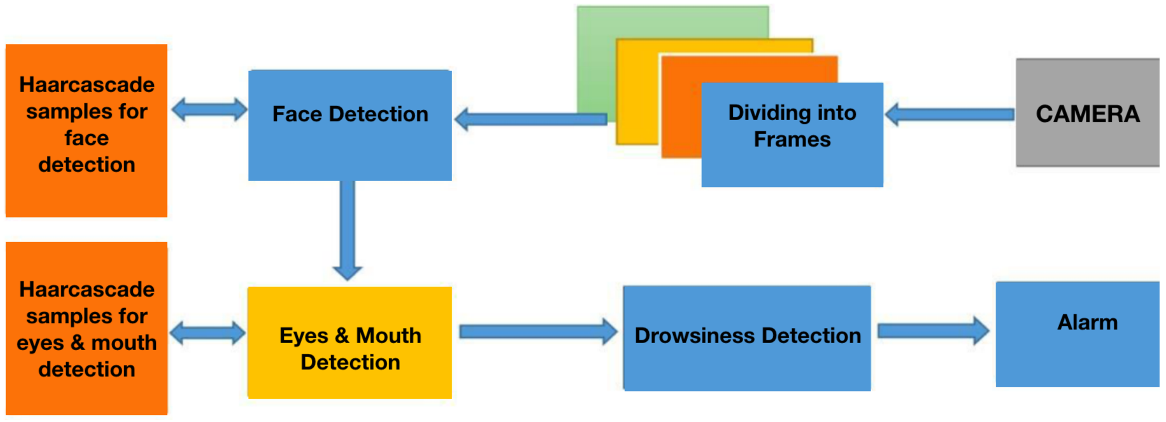


Figure 2.1: Flow chart showing the full sleepiness detection system method

Algorithm Stages:

Step 1: Capturing images:

We can acquire an image of the driver by installing a web camera inside the vehicle. Despite the fact that the camera produces a video clip, the created algorithm must be applied to each edge of the video stream. This research focuses solely on the application of the suggested technique to a single frame. The camera utilised is a low-cost web camera that shoots at 30 frames per second in VGA format. Figure 2.2 depicts the Logitech Camera utilised in this operation.

Figure 2.2: Drowsiness detection technique is implemented using a camera.

Step 2: Dividing into Frames:

We're working with a real-time situation in which footage is being recorded and analysed. However, only a picture may be used to process or apply an algorithm. As a result, the acquired video must be separated into frames in order to be analysed.

Step 3: Face Detection:

We detect the region enclosing the driver's face at this step. Every frame has a specific algorithm for detecting faces. Face detection refers to the process of detecting a person's face in a frame, or in other words, locating facial characteristics using computer technology. The frame might be any frame at all. Only face structures or characteristics are identified, with all other things such as buildings, trees, and bodies being disregarded. The 68-point model from dlib is used in this.

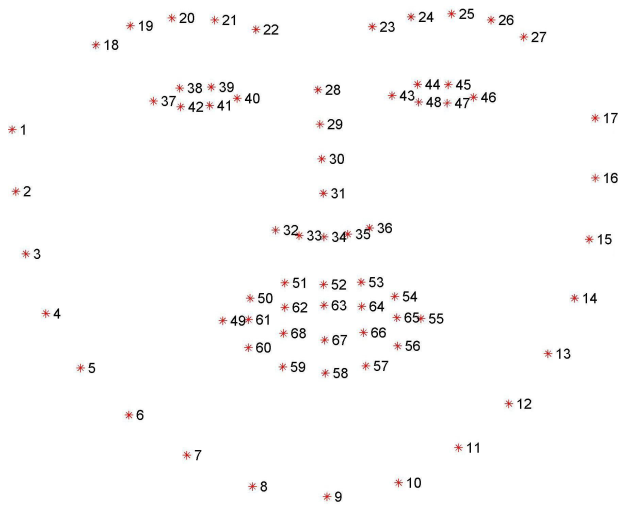
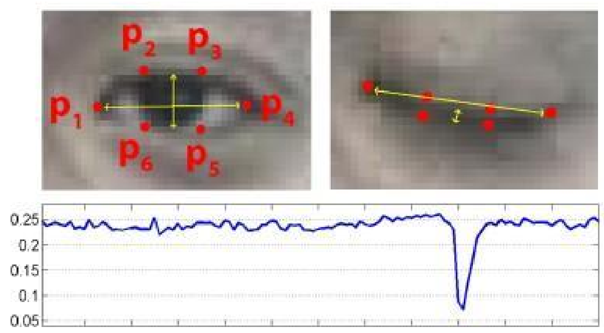


Figure 2.2: 68 Point dlib model

Step 4: Eye & Mouth Detection:

Following successful face detection, the eye must be identified for further processing. The decision parameters for determining the driver's condition in our technique are the eyes, lips, and head attitude. Though eye detection may appear to be simpler, it is actually rather difficult. At this stage, it uses numerous characteristics to recognise an eye in the needed specific location. For this technique, the Eigen method is commonly utilised. It is a lengthy procedure. When eye detection is completed, the result is compared to the reference or threshold value to determine the driver's status.

Step 5: State of eye:



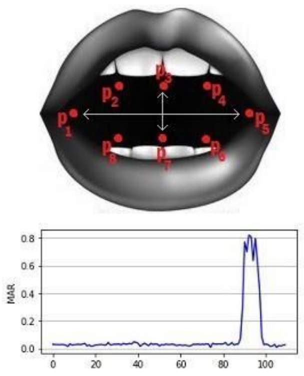
$$\text{EAR} = \frac{\|p_2 - p_6\| + \|p_3 - p_5\|}{2\|p_1 - p_4\|},$$

Figure 2.3: Eye Aspect ratio Calculation

We determine the true status of the eye at this point, whether it is closed, open, semi-closed, or open. The most significant criterion is the detection of ocular status. It is accomplished using an algorithm that will be explained in further detail in the next sections. If we detect that the eyes are open or semi-open up to a certain threshold value, we send out a warning message. If it is detected by the system that the eyes are open, the processes are repeated until a closed eye is detected.

Step 6: State of mouth:

When the mouth opens, the MAR value increases, same like the EAR value. Similar thoughts apply to this statistic as well.



$$\text{MAR} = \frac{\|p_2 - p_8\| + \|p_3 - p_7\| + \|p_4 - p_6\|}{2\,\|p_1 - p_5\|}$$

Figure 2.4: Mouth Aspect ratio Calculation

Step 7: Head Pose estimation from face detection:

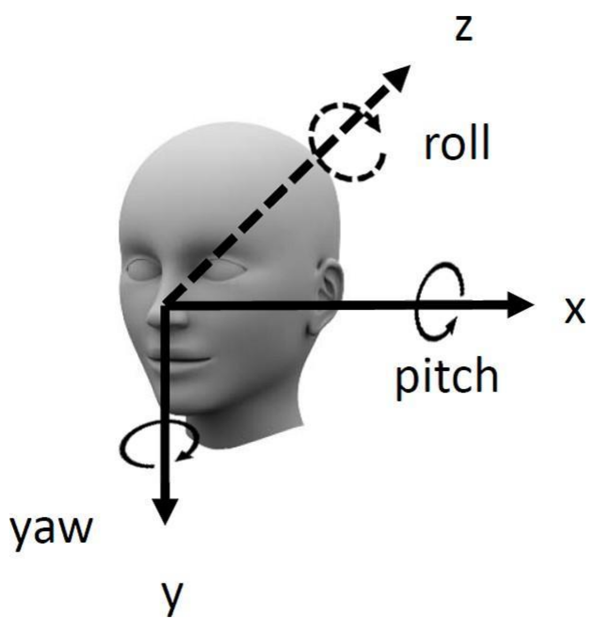


Figure 2.5: Human head in 3d space

The following is fetched to determine the 3D posture of an item in an image:

1. A few 2D (x,y) coordinates: You'll need the 2D (x,y) coordinates of a few spots in the image. You may pick the corners of the eyes, the tip of the nose, the corners of the mouth, and so on in the case of a face. The face landmark detector in Dlib gives us a lot of options to pick from.
2. 3D locations of the same points: The 3D locations of the 2D feature points are also required. You could suppose that to acquire the 3D positions, you'll need a 3D model of the person in the shot. In theory, yes, but in practise, no. It will be sufficient to use a generic 3D model. The following 3D points will be used:
    a. Nose Tip : (0.0, 0.0, 0.0)
    b. Chin : (0.0, -330.0, -65.0)
    c. Left corner of the left eye : (-225.0f, 170.0f, -135.0)
    d. Right corner of the right eye : (225.0, 170.0, -135.0)
    e. Left corner of the mouth : (-150.0, -150.0, -125.0)

f. Right corner of the mouth : (150.0, -150.0, -125.0)
g. The above positions are in an arbitrary coordinate system / reference frame. This is referred to as the World Coordinates (also known as Model Coordinates in OpenCV documentation)

The solvePnP function in OpenCV is used to estimate pose. SolvePnP provides a number of pose estimation techniques, which may be chosen using the parameter flag. The flag SOLVEPNP ITERATIVE is used by default, which is the DLT solution followed by Levenberg-Marquardt optimization. SOLVEPNP P3P calculates the posture using just three points and should be used only when using solvePnPRansac.

Chapter 3

Landmark Detection

3.1 Introduction

The first issue discussed in this chapter is the detection of objects in space. For the reason why a face is considered an object, the technique of object detection using OpenCv is explored. In the next section, it examines ways to object-based face detection. Face detection will be employed only for our objectives, despite the fact that object detection may be used to identify a range of other things. The next portion of this chapter is devoted to the eye detection strategy, since it is the most important step of drowsiness detection and also has a considerable impact on the subsequent phase of assessing the state of the eye, which is discussed below.

3.2 Object Detection

It is a method for detecting and recognising the presence of objects that correspond to a given class of objects. It may also be conceived of as an image processing strategy for recognising a specific object from a collection of photographs. There are many different strategies for categorising and finding items within a frame of reference. One strategy might be based on the identification of different colours. However, because of the possibility of multiple different sizes of the same colour item being present, it is not a very effective method of detecting the object. Because of this, Haar-like traits, which were developed by Viola and Jones in 1998 on the basis of a recommendation made by Papageorgiou et al, are a more efficient technique of classification. Object detection is accomplished via the use of digital image characteristics known as Haar-like features. Or, to put it another way, there are rectangle-shaped dark and bright patches that have characteristics that are comparable to those of our own faces. Several steps comprise the cascade classifier, each of which has a huge number of weak attributes. The system locates objects by moving a window over the whole picture and building a sophisticated classifier from the information. The output of each step is labelled as either positive or negative, with positive indicating that an item was detected in the image and negative indicating that the needed object could not be discovered in the picture.

3.3 Face detection

We already know that a face is a kind of object, as previously stated. Because of this, we may consider face detection to be a subset of object detection. Attempts are made to ascertain where and how the things in the interest picture are situated as well as their sizes in order to establish whether or not they are members of a given class. When it comes to face identification algorithms, the majority of their time is spent identifying the front side of the face. Recent algorithms, on the other hand, have been designed to handle more general cases. A slanted face or any other element of the face may be detected in our circumstance, and it can identify the potential of a large number of faces at the same time. In this case, it is the rotation axis with respect to the present observer, as determined by a particular face reference. A vertical rotation plane may also be utilised to achieve the same result if one exists in the scene. With this new algorithmic approach, images and videos are considered variables, which means that changing circumstances in them, such as colour contrast, may modify the variance of an image or a video. It's possible that the quantity of light in the room has an effect as well. The output may be influenced by the location of the input as well as the kind of input. In many calculations, the face-detection issue is recast as a two-way pattern-differentiation task, which is more efficient. It indicates that the contextual signals in the interest picture are turned into features on a recurrent basis, resulting in the construction of the necessary classifier on the reference faces, which decides whether the provided area is a face or some other object. If we are successful in detecting a face, the method proceeds to the next phase; if we are unsuccessful, the algorithm is programmed to take further photographs until a hint of a face is recognised. The Viola Jones algorithm is the most important of the algorithms used in this method. A particular output may be obtained via the usage of the cascade component of openCV. The OpenCV Cascade file consists of 24 stages and 2913 weak classifiers, which is a large number. Its window is originally 24 by 24 pixels in size, although this may be increased. The starting scale must be set to 1.0, and the step size of each scale must be set to 1.1, with the position step size set to 1.0. The step size of each scale must be set to 1.1, with the position step size set to 1.0. This method uses a total of 32 scales, resulting in an extremely large detection window of more than 1.8 million possible outcomes. Cascade has been educated by OpenCv, which makes it quite easy to operate.

3.4 Eye and Mouth Detection

When it comes to detecting, eyes with poor contrast provide a number of difficulties. Following successful face identification, it is necessary to identify the eye in order to proceed with additional processing. In our method, the decision parameter for detecting the driver's state is called the "eye." Although the process of recognising an eye may not seem to be difficult, it is incredibly frantic in its execution in actuality. In this case, it makes advantage of feature detection to determine whether or not an eye is present inside the designated zone. The Eigen method is a technique that is often used for this purpose. It is a time-consuming technique. [14] When the eye detection process is complete, the result is compared to a reference or threshold value in order to identify the driver's state in the vehicle. Eye detection may be classified into two categories: eye contour detection and eye location detection. The assumption that eyes are darker than the rest of the face is the basis for the detection of eyes. As a consequence of this, Haar It is possible to move similar characteristics over the top region of the face in order to match a property of the eye that determines the location of the eye. The non-skin areas inside the face district are regarded as possible eye locations. Obviously, eyes should be located inside a face area, and the skin identifier does not distinguish between eyes and skin in this case. Therefore, we must discover eye-simple sets from a decreased number of feasible eye regions as a consequence of this constraint. Eye detection technologies have been created in recent years, including iris recognition and eye tracking. The use of a deformable template to recognise the human eye is one of the most widely used ways of recognising the human eye. It is necessary to first develop an eye model, after which the position of the eye is computed using a recursive method. It should be noted that this strategy is very dependent on the starting location of the eye, which must be near to the actual position of the eye. Template matching is performed using eigenfeatures and neural networks to extract eyeballs from grey-level face photos using the suggested approach, which is accomplished by rectangle fitting. This method, which makes use of eigenfeatures and a sliding window, does not need the usage of a large number of training images. This algorithm, on the other hand, will fail if the user is wearing glasses or has a beard. We already know that including Haar features into AdaBoost increases both the processing efficiency and the accuracy of the algorithm when compared to other face recognition techniques. However, there is a disadvantage to the Haar characteristic: it lacks discriminant capabilities. Despite the fact that Haar features are available in a range of patterns, sizes, and locations, they can only be used to represent rectangular objects in their original form. In our scenario of eye detection, on the other hand, both the eye and the iris have a circular shape. It is possible to represent eyes as a consequence of developing discerning traits to characterise eye patterns via practice.

Chapter 4

Eigen Face Approach

4.1 Introduction

Several techniques to face and eye recognition are discussed in detail in Chapter 4, which focuses on the theoretical and mathematical basis of these approaches. This chapter began with an introduction to the Principal Component Analysis (PCA) approach, which is a kind of data analysis. Using the Eigenface approach, which contains a mathematical description of Eigenvalues and Eigenvectors, as well as facial image representation, mean and mean-centered pictures, covariance matrix, and Eigenface space, the latter is discussed.

4.2 Principal Component Analysis (PCA)

In 1901, Karl Pearson developed the principal component analysis (PCA) method. If the generated data is repeated or includes redundancy, PCA may aid in reducing the amount of redundancy in the data set. PCA is a technique for eliminating duplicate variables. We will have fewer variables, which will be referred to as Principal Components when we have reduced the number of variables we have. Primary components are often used to represent all the variables that are included in a given variable's principal components. However, just lowering the number of variables does not provide the desired result. In order to recognise faces in a more complicated and high-dimensional environment, we must first learn to recognise faces in a simple and low-dimensional environment. The major purpose of principal component analysis (PCA) is to lower the number of dimensions while maintaining an increasing amount of accessible variation in the given data. Although it may seem surprising, we already know that diminishing dimension results in information loss since dimension is fundamentally linked to information. As a consequence, we may prevent data loss by picking the best principle components, since the main principal components determine the low dimension, we can avoid data loss by selecting the best principle components. Despite the fact that PCA has a number of advantages, it is most typically used in combination with the eigenface approach. It is possible to reduce the quantity of data in the database that is used to recognise the test photographs by using the eigenface approach. Vectors, also known as feature vectors, are used to store the images that have been captured in the database. They are decided by projecting a collection of Eigen faces onto the training image, which is then used to produce the collection of Eigen faces. In this case, PCA is generally used in conjunction with the eigenface approach to minimise dimensionality while maintaining data integrity.

4.3 Eigen face approach

Because of its speed of operation, simplicity of use, and ability to learn, the Eigenface approach for face recognition is very efficient and useful to both the user and the system. When it comes to computer vision, face recognition is performed via the usage of eigen faces, which are basically just a collection of eigenvectors. A purely appearance-based approach to face recognition, in which variation in a set of face images is collected and used for comparison and suitable encoding of each individual face, is described here. Eigen faces are the primary components of a distributed face that are represented in the form of a collection of faces' covariance matrix, and they are referred to as eigenfaces. In this way, a face image is represented as a one-dimensional matrix in computer graphics. A face in two dimensions may be represented as a N x N matrix in N2 dimension space, as we have seen before. These N x N matrices are turned into row matrices using a transformation matrix. Numerous research have been undertaken on this issue, but they have all failed to take into consideration the fact that face recognition requires the presence of a face stimulus, which assumes that the existing measurements of face recognition are substantial and acceptable. As a result, the coding and encoding of easily available face pictures are likely to include information on the most prominent features of the photographs' appearance. However, there is a risk that the qualities that are generated will be unconnected to the well-known and required facial characteristics such as the nose, eyes, lips, and hair, among others. As a consequence, it is necessary to extract information from a face photograph. Following the extraction, we encode it effectively and compare the result to a database of faces that are similar to the one we extracted. For this, we collect variations using a series of face images, which is a very basic way for extracting the information content from the photos themselves. This phase is followed by determining the Principal Component of the face distribution (or, alternately, determining the Eigenvectors of a collection of face pictures) using the covariance matrix that has been gathered. Each row of the image is considered as a vector that is stacked one after the other in a single row, which allows the Eigenvector to be represented as a sort of face in the picture. Each individual face image is represented by a unique collection of individual face photos. Our research reveals that the total number of given input photographs in the prepared set is responsible for determining the sum of all projected eigen faces. The Eigenface algorithm may be used to approximate faces, particularly those with large eigenvalues that set the highest variance in the provided collection of pictures. Eigen faces are used in smaller numbers in order to enhance computing performance.

4.3.1 Eigenvalues and Eigenvectors:

[17]A linear equation expressed in matrix form is denoted by Ax= D in linear algebra. A linear operator's eigenvectors are non-zero vectors that, when acted on by the operator, result in the operator's eigenvalues. As a consequence, a scaled multiple of them is produced. The resultant scaler is referred to as the eigenvalue () for the eigenvector X. An eigenvector is a vector that is paralleled by linear transformation. It is one of the matrix's properties. When we compute a matrix on it, the vector's magnitude is altered. The vector's direction stays unchanged. As a result, we define as Ax = x, where A is a vector function. After that, translate the RHS section and write it as (AI)x = 0, where I is the identity matrix. The above equation is a homogeneous equation, which is a basic concept in linear algebra. The existence of a non-trivial solution is determined by assuming that Det(AI) = 0, where Det denotes the determinant. When it is evaluated, we are dealing with the degree n polynomial. This is referred to as A's characteristic polynomial. When A's dimension is represented by NxN, the solutions result in n roots of the characteristic polynomial. As a result, it returns n Eigenvalues for A that meet the Axi = ixi constraint, where I = 1,2,3,. n. If all of the resulting eigenvalues are distinct, we get n related linearly independent eigenvectors with separate orientations.

4.3.2 Face Image Representation:

In this approach we represent a set of, let's say m images of each having size NxN. This is done by vectors of size N2. We represent each face Γ1, Γ2, Γ3… Γn. All those ⫛obtained feature vectors are stored in the matrix with size NxN. One example is shown below which describes the entire process.

For example:

$$\begin{bmatrix} 3 & 7 \\ 6 & 5 \end{bmatrix} = \begin{bmatrix} 3 \\ 7 \\ 6 \\ 5 \end{bmatrix}$$

4.3.3 Mean and mean centered Image and Covariance matrix:

The average face is calculated as follows:

$$\Psi = \sum_{n=1}^{\infty} , \Gamma n$$

Then we calculate the difference between each face and the average face.

$$\Phi_i = \Gamma_i - \Psi$$

As seen below, we may create a covariance.

$C = AA^T$, where $A = \left[\Phi_1, \phi_2, \dots \phi_m\right]$ of size $N^2 X N^2$. As we can see that the size of covariance matrix will be $N^2 X N^2$. We need to discover the Eigenvectors for the covariance matrix, which is rather large.. However, because of its vast size, it is time consuming and laborious. In order to solve this difficulty, we calculate $A^T A$. Now let's consider the eigenvectors Vi of $A^T A$ such that

$A^T A X_i = \lambda_i X_i$

The eigenvectors Vi

of $A^T A$

are $X_1 \dots X_n^{\ 2}$

Now, we multiply the previous equation by A on both sides to simplify it, and we obtain

$AA^T A X_i = A\lambda_i X_i$

$AA^{T(} AXX) = \lambda i(AXi)$

From above its evident that Eigenvectors responding to AA^T is now securely calculated as a result of the dimension reduction where AX_i is the Eigenvector and λ_i is the Eigenvalue.

4.3.4 Eigenface space:

Let's say we have a covariance matrix AA^T. So the eigenvectors corresponding to that matrix which is denoted by U_i where U_i represents facial images. Those eigenfaces have a ghostly appearance. Faces with eigenvalues of 0 are removed, and only those eigenvectors that correlate to the Eigenface in the face space are accepted. This strategy greatly aids in the reduction of the Eigen face. The effectiveness of Eigen faces in describing variance among pictures is used to determine their rank. Here we project the face image into the face the projection of each image as Ω1 for projection of image l and Ω2 for projection of image2 and likewise.

Chapter 5

Hardware Implementation

5.1 Introduction

Chapter 5 goes into great depth about the hardware used to create the fatigue detection system. The Raspberry Pi was used as the computing hardware. Brief descriptions of the hardware used, as well as the installation and setup methods, are given. At a more fundamental level of abstraction, the chapter's middle portion explained how drowsiness detection works as a whole. OpenCv libraries are used to do this. Use OpenCv's.xml files for a range of inputs to get the desired result Viola Jones' method is the primary source of the face and eye detection in the.xml files used for drowsiness detection. Some of the methods employed include Adaboost, Cascading, Haar features, and the creation of an integral picture. All of these features have theoretical foundations, which are briefly examined.

5.2 Implementation

5.2.1 About Raspberry Pi

An inexpensive, credit-card-sized computer, it is used for the implementation of tiny projects at a cheap cost. It requires an external monitor or television to be linked to it in order to view and run its operating system. We may supply input to it via the use of a keyboard and a mouse. Its operating system must be loaded from an external memory, which must be done through USB. It may be programmed in a variety of languages, including C++, Python, and others.

There are many components to it, including the following:

i. 700 MHz processor.

ii. 512 MB RAM.

iii. USB ports.

iv. Micro SD card slots.

v. Ethernet port.

vi. HDMI port.

vii. 40 GPIO pins.

viii. Camera interface.

ix. Display interface.

x. Power supply port.

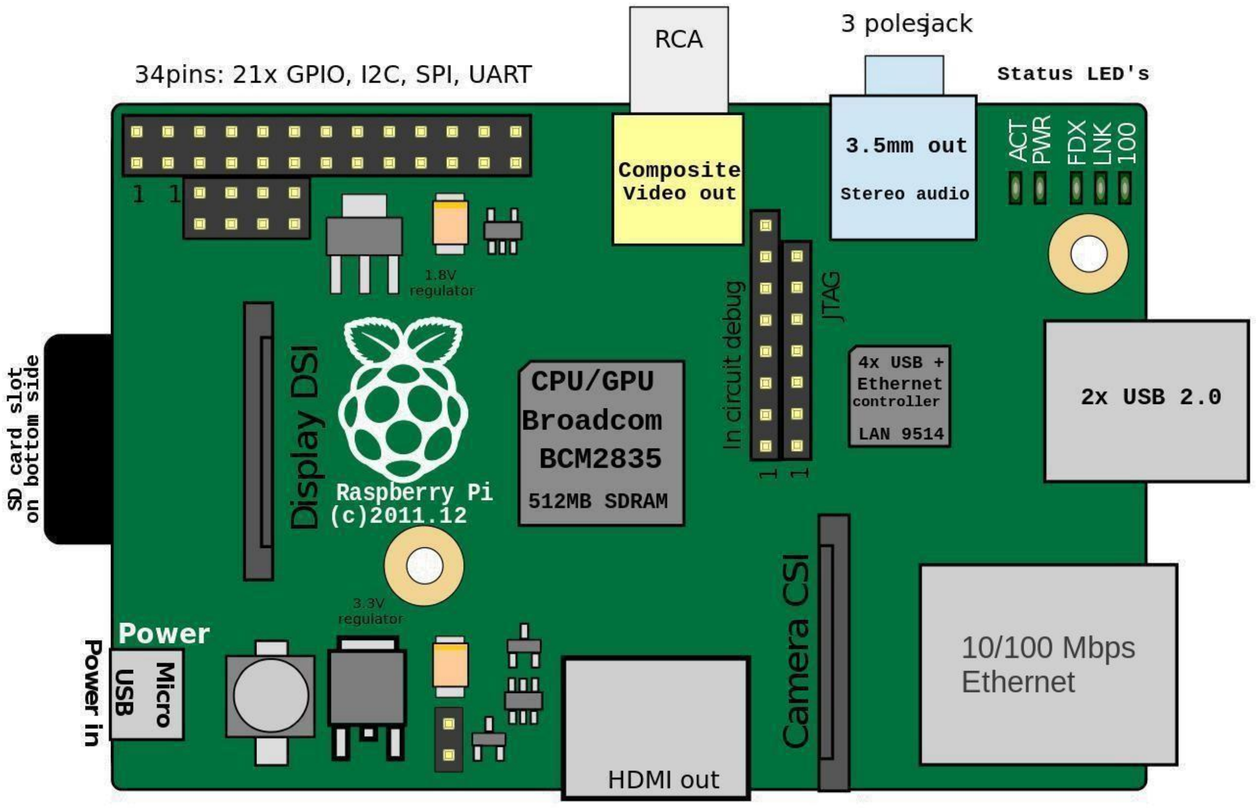


Figure5.1: figure shows different components of RaspberryPi

We have implemented Face detection with the help of:

i. Raspberry pi

ii. Web Cam

iii. Raspbian operating system

iv. Python IDLE

v. OpenCv (Open source Computer Vision) for python with Harr object detection trainer

vi. Program code for face detection written in Python Programming language

Paul Viola and Michael Jones created it in 2001, the face identification algorithm used in OpenCv is known as the Viola-Jones approach and is well-known in the computer vision community. Even though this approach may be used to identify a variety of objects, it is most often utilised for face and eye identification in real time in this application.

Viola-Jones algorithm has four stages:

i. Haar Feature Selection
ii. Creating an Integral Image
iii. Adaboost Training
iv. Cascading Classifiers

5.2.2 Haar features

Haar-like features are digital image characteristics that are employed in the detection of objects in digital images. [17] Alternatively, we might argue that they are rectangle-shaped dark and bright patches with traits that are comparable to those seen on our faces. So, in essence, we move those features across our face in order to determine the output of each feature.

For example:

All faces share some similar properties

1. While the upper-cheeks are lighter, the area around the eyes is darker
2. The area around the nasal bridge is brighter than the area around the eyes

As a result, these facial characteristics are used in the development of haar-like characteristics. Each characteristic is associated with a specific spot on the face.

Output of Rectangle features:

We will move the related kind of rectangle throughout the face to get different values.

1. Value = ∑ (pixels in black area) - ∑ (pixels in white area).
2. Three types: two-, three-, four-rectangles, Viola and Jones used two rectangle features.

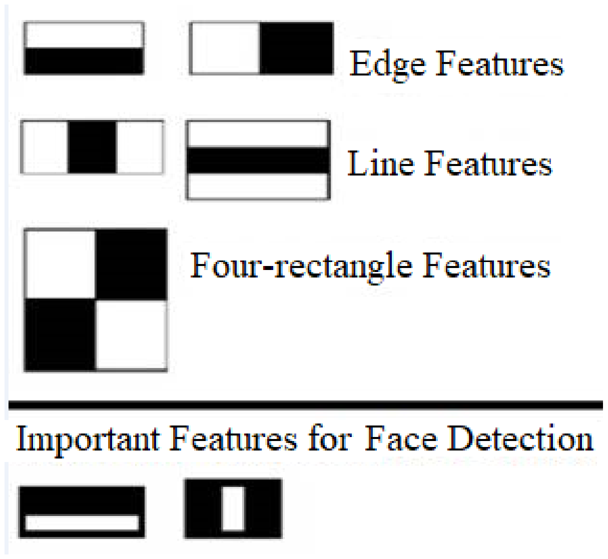


Figure 5.2: Different features used for Haar cascade

Reference: W. Zhao, R. Chellappa, P.J. Phillips, and A. Rosenfeld, "Face Recognition: A Literature Survey," ACM Computing Surveys, vol. 35, pp. 399-459 , 2003

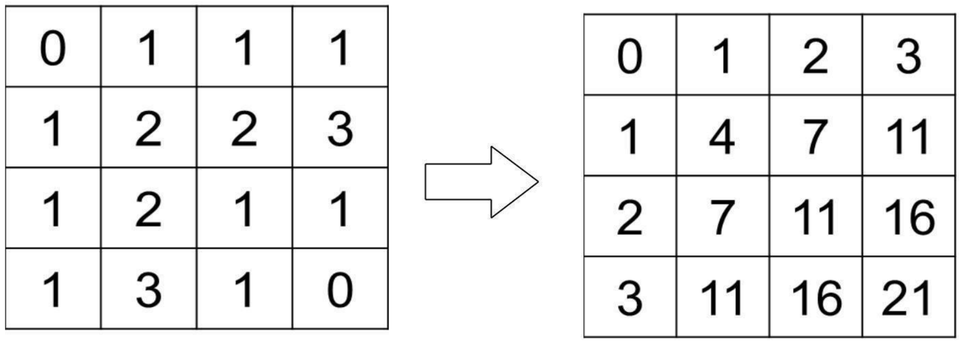
5.2.3 Integral Image

We are all aware that each point on an image is represented by a pixel value of some kind. [17] As a result, in order to determine the output of the applied Haar features, we must first obtain the total of the pixel values of all the areas in question and then solve the summation. However, this is a monumental undertaking. The notion of integral image is introduced in order to decrease the amount of calculations.

Definition of Integral Image:

Basically, the integral image is a matrix with the same dimensions as the window. The integral picture at the coordinates (x, y) is the total of the pixels above and to the left of the coordinates (x, y).

For example:



0	1	1	1
1	2	2	3
1	2	1	1
1	3	1	0

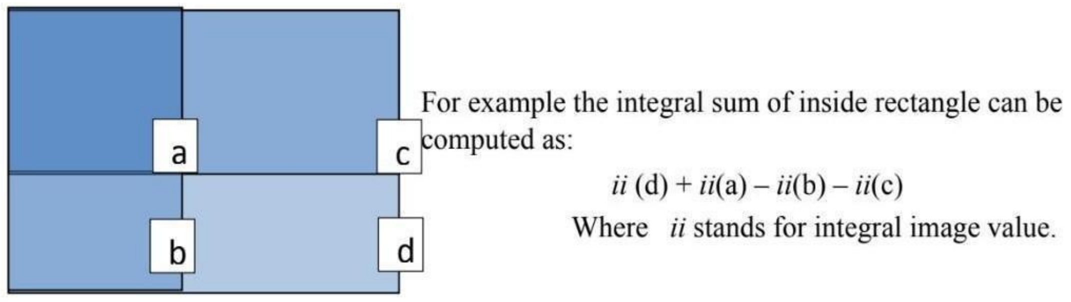
0	1	2	3
1	4	7	11
2	7	11	16
3	11	16	21

Figure 5.3: figure shows integral image formation

Reference: W. Zhao, R. Chellappa, P.J. Phillips, and A. Rosenfeld, "Face Recognition: A Literature Survey," ACM Computing Surveys, vol. 35, pp. 399-459 , 2003

Here, the pixel value of each box is adjusted by the total of all the boxes to the left and above it, allowing us to utilise the formula below to get the output of Haar features with far less computation, hence lowering the time required for calculation.

As a result, if we wish to determine the pixel value of a rectangle, we may do it by simply picking four points from the integral picture, as shown previously.



For example the integral sum of inside rectangle can be computed as:

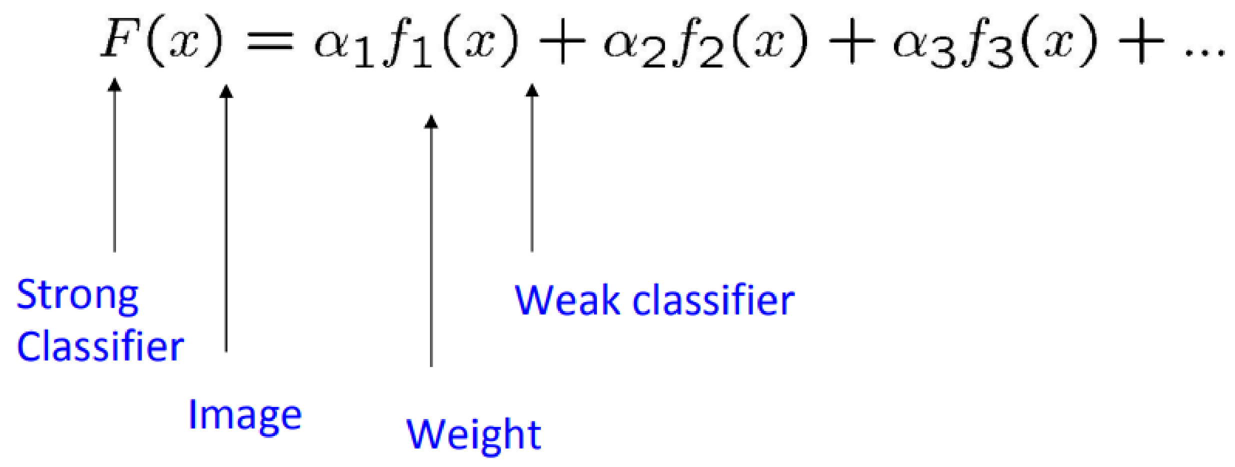
$ii\,(\mathrm{d}) + ii(\mathrm{a}) - ii(\mathrm{b}) - ii(\mathrm{c})$

Where ii stands for integral image value.

Figure 5.4: Integral image calculation

5.2.4 AdaBoost

[17] Adaboost is an abbreviation for "Adaptive" boost. We design a strong classifier as a linear combination of weak classifiers in this case since there are so many features that are completely ineffective in identifying face characteristics. It may be expressed in the following way:



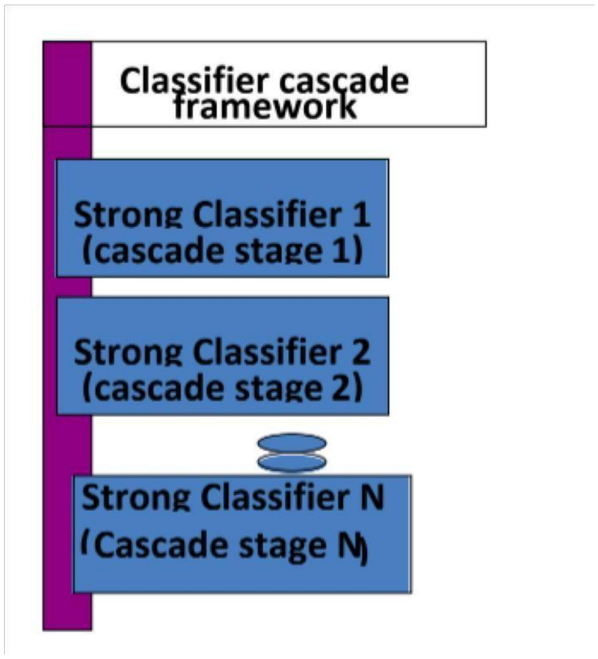
$$F(x) = \alpha_1 f_1(x) + \alpha_2 f_2(x) + \alpha_3 f_3(x) + \ldots$$

Features of weak classifiers:

Each rectangle is considered to be a simple and weak classifier. Each weak classifier is allocated a weight function based on the relevance of its place in the classification hierarchy. Finally, the linear combination of these classifiers results in the formation of the strong classifier.

5.2.5 Cascade

Let us suppose that we have 600 strong classifiers after going through the Adaboost step. In order to determine if a frame includes a face or not, the following criteria must be met: To save time, organise the 600 characteristics into distinct phases of classifiers and apply them one at a time, rather than applying them all at once on a window. If a window does not pass the first step, it should be discarded. We do not consider any of the remaining characteristics to be part of it. If it passes, add the second stage of features and proceed with the procedure as previously stated. A face area is a window that has passed through all of the phases.


Classifier cascade framework
Strong Classifier 1 (cascade stage 1)
Strong Classifier 2 (cascade stage 2)
Strong Classifier N (Cascade stage N)

Modification in the algorithm

Modification in face detection:

The loading of the cascade file is necessary for the detection of faces. The recorded frame must then be passed on to a function that performs edge detection in order to fulfill the requirement. Following this procedure, it is feasible to identify almost all of the potential types of objects corresponding to various sizes. As a result, the issue of reducing the amount of processing required arises. Instead of identifying all of the items in the frame, we will focus on detecting just the face, which fills almost the whole recorded frame in the majority of instances. As a result, we may alter the algorithm such that it only detects in this manner.

Modification in eye detection:

If we apply the characteristics to every part of the face, the amount of processing required will be prohibitively large. This is especially true for eye identification. As a result, in order to avoid this predicament, we should only be interested in those portions of the face where we are certain that an eye exists. By considering the following information, we may narrow down our search area for the eye.

1. Only upper parts of the face contain the maximum probability of finding an eye.
2. The place of occurrence of eyes are a few pixels below the forehead.

Modification in colour selection

Instead of utilising a coloured picture for sleepiness detection, the image is changed to grey scale in order to minimise the number of channel parameters, which aids in increasing the speed with which the classifiers can be calculated and evaluated.

Chapter 6
Conclusion & Limitations

6.1 Conclusion

Implementation of sleepiness detection has been completed, and it consists of the following procedures:

Successful capture of video using a camera during a runtime session. [17] The captured footage was separated into frames, and each frame was subjected to in-depth analysis. The successful identification of the face was followed by the successful detection of the eye. If the driver's eyes are closed for consecutive frames, the condition is classed as sleepy; otherwise, the situation is labelled as normal blink, and the loop of taking images and assessing the driver's status is repeated over and over again until the driver is awake. During the sleepy condition, the eye is not enclosed by a circle or it is not recognised, and the accompanying message is not shown in this implementation. For every successful detection of an open eye while driving when not sleepy, the driver's eye is marked with a circle, and the program writes 1.

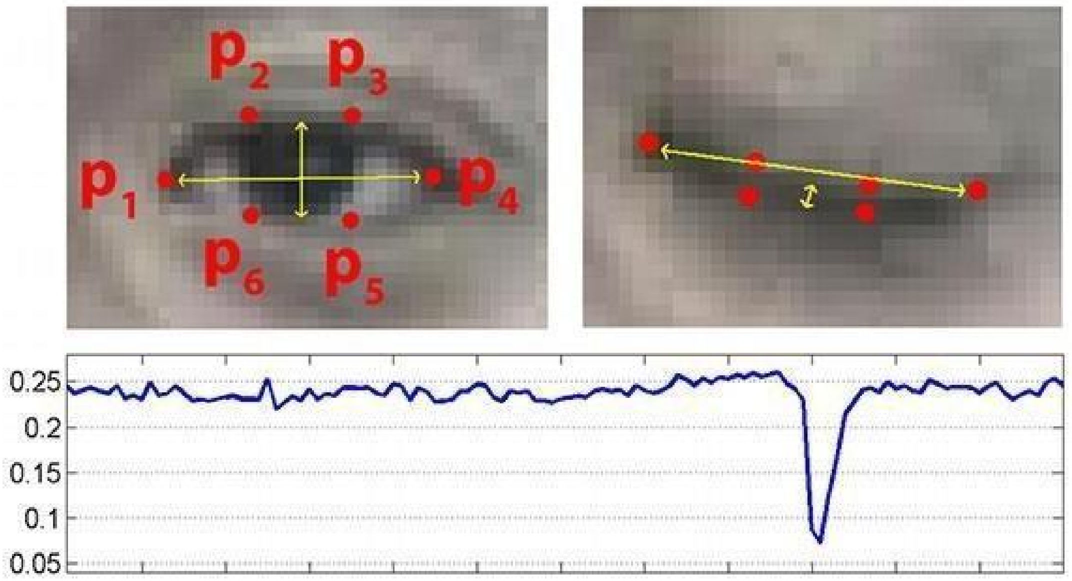


Figure6.1: Eye Aspect Ratio Calculation Reference: K. C. Yowand, R. Cipolla, "Feature- based human face detection, "Image Vision Comput., vol.15, no.9, 1997, pp.713–735.

On the top-left, we have an eye that is completely open, with the eye's facial landmarks mapped on the surface of the eye. Then there's an eye that's closed in the upper right corner. The eye aspect ratio is then shown against time at the bottom.

In this image, we can observe that the eye aspect ratio is constant (showing that the eye is open), then swiftly dips to zero, then fast raises again, suggesting that the eye has been blinking.

As an example, with our sleepiness detector, we'll be watching the eye aspect ratio to check whether it decreases but does not raise again, indicating that the individual has closed their eyes. We have determined that sleepiness is present if and only if the eye aspect ratio goes below 0.2 for more than 48 frames. This is due to the fact that a regular camera can only record at 24 frames per second, resulting in a 2 second window for the acquisition of the 48 frames.

Following conditions trigger the drowsiness alert:

1. Eyes closed for 45 frames
2. Yawn more than 2 times
3. Head goes vertically down

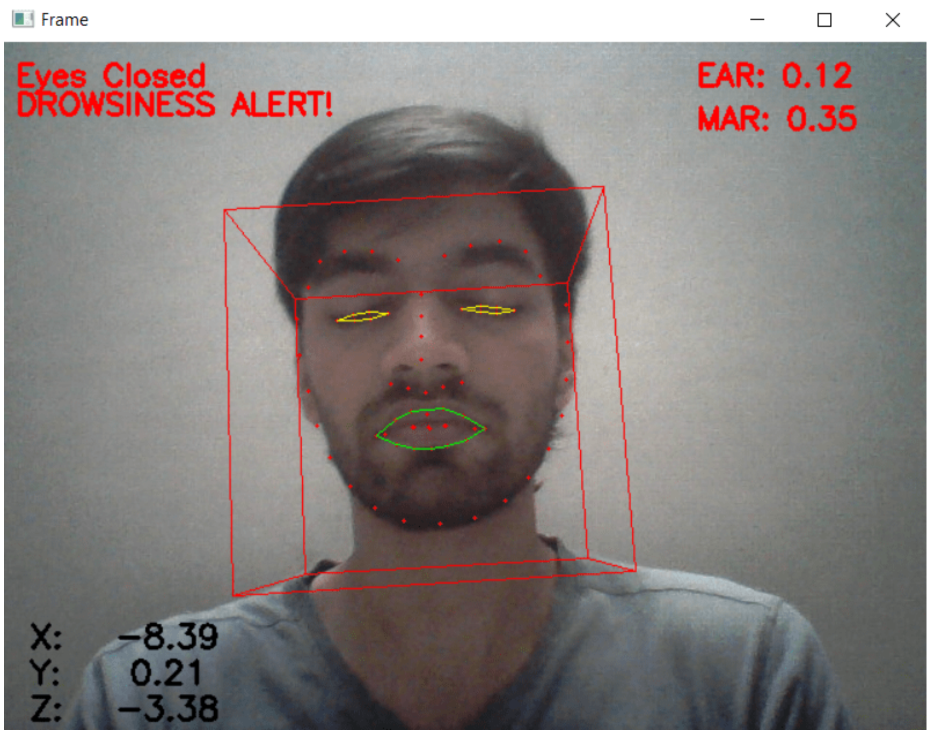


Figure6.2: Eyes getting closed for more than 24 consecutive frames

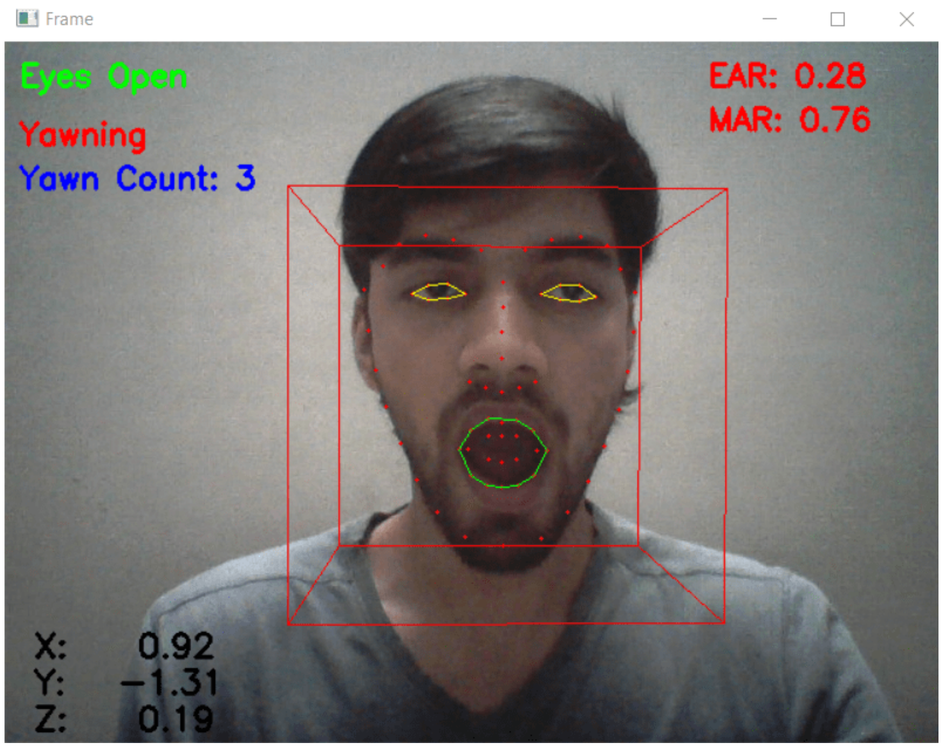


Figure6.3: More than 2 yawns detected

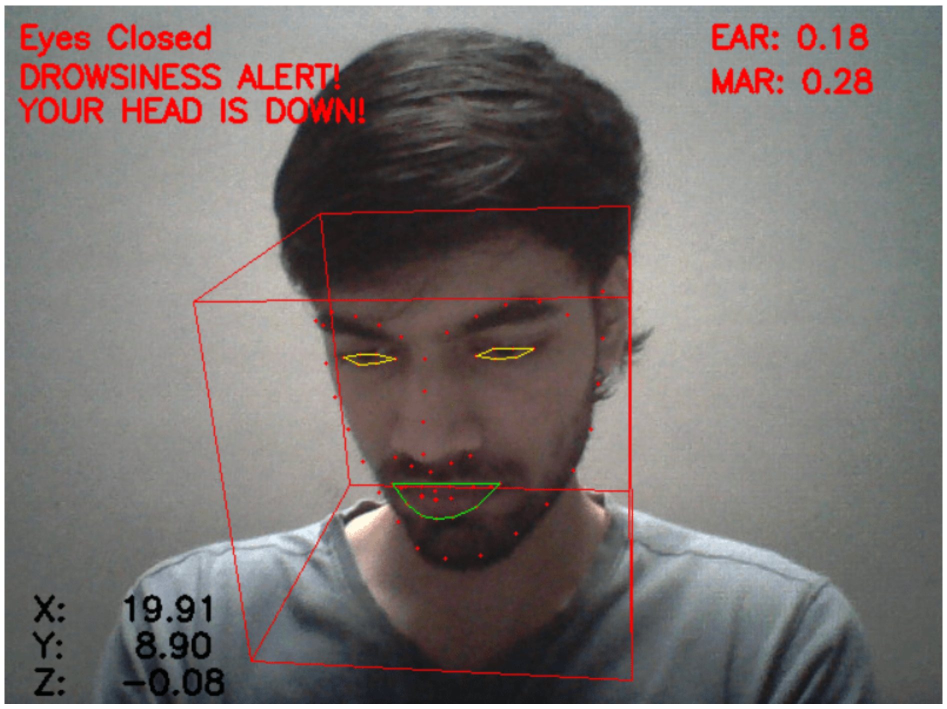


Figure6.4: Head gets tilted vertically down

Limitations

Dependence on ambient light: The model designed for this purpose is highly dependent on the lighting conditions in the room. Because our algorithm considers the eyesight to be a dark area when it is closed and a brighter region when it is open, if the ambient state changes in such a way that there is a potential of brighter and darker conditions depending on the light source, the result will be incorrect. Additionally, this model is dependent on a certain minimum degree of light situation, otherwise it becomes challenging to detect. In order to prevent this inaccuracy, we may either utilise LED lights for improved detection or an infrared camera to identify the object.

Distance of camera from driver face: The code was written with the assumption that the distance between the camera and the subject's face should be around 100 cm in order to get the best possible outcome. As a result, since various vehicles have different sorts of seat lengths, the output of the planned set up may range from vehicle to vehicle.

Processor speed of hardware: For the implementation, we made use of the RaspberryPi. The RaspberryPi has a CPU with a speed of 700 MHz. As a result, this processor's performance is insufficient for video processing tasks to be completed. As a result, a CPU with very high speed is required, which will eventually raise the cost of the device.

Use of spectacles:Detecting the condition of the eye becomes more difficult if the person wears spectacles, as seen in the image below. Because it is highly dependent on light, the output for a closed eye may be the same as for an opened eye if the glasses are reflected. It is thus necessary to keep the eye near to the camera in order to avoid light for this reason.

Multiple face problem: If more than one face appears in the window, the camera may identify an increased number of faces, resulting in an undesirable output. Because each person's face is in a distinct state of health. Consequently, we must ensure that only the driver's face is inside the camera's seeing field of view. In addition, because of the operation on many faces, the speed of detection is reduced.

REFERENCES

[1]. L. Mansinha, R. G. Stockwell, and R. P. Lowe, “Pattern analysis with two dimensional spectral localization: Applications of two-dimensional S transforms,” Physica A, vol. 239, pp.286–295, 1997.

[2]. J. MacQueen, “Some methods for classification and analysis of multivariate observations,” in Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability., pp. 281–297, University of California Press., 1967.

[3]. J. Liu, C. K. Wong, and K. K. Hui, “An adaptive user interface based on personalized learning,” Intelligent Systems, IEEE, vol. 18, no. 2, pp. 52– 57, 2003.

[4]. H. Wagner, C. MacDonald, and A. Manstead, “Communication of individual emotions by spontaneous facial expressions,” Journal of Personality and Social Psychology, vol. 50(4), pp. 737–743, 1986.

[5]. P. F. Alcantarilla and T. Solutions, “Fast explicit diffusion for accelerated features in nonlinear scale spaces,” IEEE Trans. Patt. Anal. Mach. Intell, vol. 34, no. 7, pp. 1281– 1298, 2011.

[6]. S. Leutenegger, M. Chli, and R. Y. Siegwart, “Brisk: Binary robust invariant scalable keypoints,” in Computer Vision (ICCV), 2011 IEEE International Conference on. IEEE, 2011, pp. 2548–2555.

[7]. H. Bay, T. Tuytelaars, and L. Van Gool, “Surf: Speeded up robust features,” in Computer vision–ECCV 2006. Springer, 2006, pp. 404– 417.

[8]. Viola P, Jones M, Snow D (2003) Detecting pedestrians using patterns of motion and appearance. 9th IEEE international conference on computer vision (ICCV 2003), Nice, France: 734- 741.

[9]. Meng – che chuang, Raja bala, Edgar A. Bernal, Peter Paul, Aaron Burry, “Estimating Gaze Direction of Vehicle Drivers using a Smartphone Camera” CVPR workshop paper 2014.

[10]. Boon – Giin Lee and Wan – Young Chung, “Driver alertness monitoring using fusion of facial features and bio signals” IEEE SENSORS JOURNAL Vol. 12 july 2012.

[11]. K. Murata, E. Fujita, S. Kojima, S. Maeda, Y. Ogura, T. Kamei, T. Tsuji, S. Kaneko, M. Yoshizumi, and N. Suzuki, “Noninvasive biological sensor system for detection of drunk driving,” IEEE Trans. Inf. Technol. Biomed., vol. 15, no. 1, pp. 19– 25, Jan. 2011.

[12]. LIU Yi,GONG Wei-guo,et al.Robust Classifier Based two-1ayer Adaboost for Precise Eye Location[J].Computer Applications, 2008,28(3):801-803.

[13]. JIANG Shui-lang,YANG Min. Eye Location Method for Driver Fatigue Detection in Dual Spaces[J]. Computer Engineering, 2008,34(24):180-182.

[14]. CAO Wei,WANG Feng,ZHOU Ping. Eyes Location Algorithm Based on Machine Vision[J].Traffic and Computer, 2007,25(6):34-36,40.

[15]. LI Pei-lin, HE Cui-qun,et al.Comparison and Analysis of Eye orientation Methods in Driver Fatigue Detection[J].Forest Engineering, 2008,24(5):35-38.

[16] W. Zhao, R. Chellappa, P.J. Phillips, and A. Rosenfeld, “Face Recognition: A Literature Survey,” ACM Computing Surveys, vol. 35, pp. 399-459, 2003.

[17] Implementation of Voila Jones Algorithm by Ole Helvig Jensen, university of Denmark, 2008, pp: 20-36.

[18] Christian Scharfenberger, Samarjit Chakraborty, John Zelek and David Clausi”, Anti-Trap Protection for an Intelligent Smart Car Door System”, 15th International IEEE

Conference on Intelligent Transportation System, Anchorage, Alaska, USA, September 16- 19, 2012.

[16] [19] Andrzej Majkowski, Marcin Kołodziej, Remigiusz J. Rak, Paweł Tarnowski, Karol Szczepanek, Detecting symptoms of driver fatigue using video analysis, 2016, pp: 20-36.

APPENDIX

Code:

```
from scipy.spatial import distance as dist
from imutils import face_utils
import numpy as np
import imutils
import dlib
import cv2
import threading

#for Alarm
import playsound
path = "beep2.wav"
def sound_alarm(path):
   playsound.playsound(path)

#head pose
K = [6.5308391993466671e+002, 0.0, 3.1950000000000000e+002,
    0.0, 6.5308391993466671e+002, 2.3950000000000000e+002,
    0.0, 0.0, 1.0]
D = [7.0834633684407095e-002, 6.9140193737175351e-002, 0.0, 0.0, -1.3073460323689292e+000]

cam_matrix = np.array(K).reshape(3, 3).astype(np.float32)
dist_coeffs = np.array(D).reshape(5, 1).astype(np.float32)

object_pts = np.float32([[6.825897, 6.760612, 4.402142],
                         [1.330353, 7.122144, 6.903745],
                         [-1.330353, 7.122144, 6.903745],
                         [-6.825897, 6.760612, 4.402142],
                         [5.311432, 5.485328, 3.987654],
                         [1.789930, 5.393625, 4.413414],
                         [-1.789930, 5.393625, 4.413414],
                         [-5.311432, 5.485328, 3.987654],
                         [2.005628, 1.409845, 6.165652],
                         [-2.005628, 1.409845, 6.165652],
                         [2.774015, -2.080775, 5.048531],
                         [-2.774015, -2.080775, 5.048531],
                         [0.000000, -3.116408, 6.097667],
                         [0.000000, -7.415691, 4.070434]])

reprojectsrc = np.float32([[10.0, 10.0, 10.0],
                     [10.0, 10.0, -10.0],
                     [10.0, -10.0, -10.0],
                     [10.0, -10.0, 10.0],
                     [-10.0, 10.0, 10.0],
                     [-10.0, 10.0, -10.0],
                     [-10.0, -10.0, -10.0],
                     [-10.0, -10.0, 10.0]])

line_pairs = [[0, 1], [1, 2], [2, 3], [3, 0],
          [4, 5], [5, 6], [6, 7], [7, 4],
          [0, 4], [1, 5], [2, 6], [3, 7]]

def get_head_pose(shape):
   image_pts = np.float32([shape[17], shape[21], shape[22], shape[26], shape[36],
                     shape[39], shape[42], shape[45], shape[31], shape[35],
                     shape[48], shape[54], shape[57], shape[8]])

   _, rotation_vec, translation_vec = cv2.solvePnP(object_pts, image_pts, cam_matrix, dist_coeffs)

   reprojectdst, _ = cv2.projectPoints(reprojectsrc, rotation_vec, translation_vec, cam_matrix,
                              dist_coeffs)

   reprojectdst = tuple(map(tuple, reprojectdst.reshape(8, 2)))

   # calc euler angle
```

```
    rotation_mat, _ = cv2.Rodrigues(rotation_vec)
    pose_mat = cv2.hconcat((rotation_mat, translation_vec))
    _, _, _, _, _, _, euler_angle = cv2.decomposeProjectionMatrix(pose_mat)

    return reprojectdst, euler_angle

#calculating eye aspect ratio
def eye_aspect_ratio(eye):
    # compute the euclidean distances between the vertical
    A = dist.euclidean(eye[1], eye[5])
    B = dist.euclidean(eye[2], eye[4])

    # compute the euclidean distance between the horizontal
    C = dist.euclidean(eye[0], eye[3])
    # compute the eye aspect ratio
    ear = (A + B) / (2.0 * C)
    return ear

#calculating mouth aspect ratio
def mouth_aspect_ratio(mou):
    # compute the euclidean distances between the horizontal
    X   = dist.euclidean(mou[0], mou[6])
    # compute the euclidean distances between the vertical
    Y1  = dist.euclidean(mou[2], mou[10])
    Y2  = dist.euclidean(mou[4], mou[8])
    # taking average
    Y   = (Y1+Y2)/2.0
    # compute mouth aspect ratio
    mar = Y/X
    return mar

cap = cv2.VideoCapture(0)
predictor_path = 'shape_predictor_68_face_landmarks.dat'
```

```
# define constants for aspect ratios
EYE_AR_THRESH = 0.25
EYE_AR_CONSEC_FRAMES = 45
MOU_AR_THRESH = 0.75

COUNTER = 0
yawnStatus = False
yawns = 0
# initialize dlib's face detector (HOG-based) and then create
# the facial landmark predictor
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor(predictor_path)

# grab the indexes of the facial landmarks for the left and right eye
# also for the mouth
(lStart, lEnd) = face_utils.FACIAL_LANDMARKS_IDXS["left_eye"]
(rStart, rEnd) = face_utils.FACIAL_LANDMARKS_IDXS["right_eye"]
(mStart, mEnd) = face_utils.FACIAL_LANDMARKS_IDXS["mouth"]

# loop over captuing video
while True:
   # grab the frame from the cap, resize
   # it, and convert it to grayscale
   # channels)
   ret, frame = cap.read()
   frame = imutils.resize(frame, width=640)

   gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
   prev_yawn_status = yawnStatus
   # detect faces in the grayscale frame
   rects = detector(gray, 0)

   # loop over the face detections
   for rect in rects:
        # determine the facial landmarks for the face region, then
        # convert the facial landmark (x, y)-coordinates to a NumPy
        # array
        shape = predictor(gray, rect)
        shape = face_utils.shape_to_np(shape)

        # extract the left and right eye coordinates, then use the
        # coordinates to compute the eye aspect ratio for both eyes
        leftEye = shape[lStart:lEnd]
        rightEye = shape[rStart:rEnd]
        mouth = shape[mStart:mEnd]
        leftEAR = eye_aspect_ratio(leftEye)
        rightEAR = eye_aspect_ratio(rightEye)
        mouEAR = mouth_aspect_ratio(mouth)
        # average the eye aspect ratio together for both eyes
        ear = (leftEAR + rightEAR) / 2.0

        #head pose detection
        reprojectdst, euler_angle = get_head_pose(shape)
        for (x, y) in shape:
                cv2.circle(frame, (x, y), 1, (0, 0, 255), -1)

        for start, end in line_pairs:
           print(reprojectdst[start])
           cv2.line(frame, (int(reprojectdst[start][0]), int(reprojectdst[start][1])),
(int(reprojectdst[end][0]), int(reprojectdst[end][1])), (0, 0, 255))
        xx=euler_angle[0, 0]
        cv2.putText(frame, "X: " + "{:7.2f}".format(xx), (20, 420), cv2.FONT_HERSHEY_SIMPLEX,
                0.75, (0, 0, 0), thickness=2)
        yy=euler_angle[1, 0]
        cv2.putText(frame, "Y: " + "{:7.2f}".format(yy), (20, 445), cv2.FONT_HERSHEY_SIMPLEX,
                0.75, (0, 0, 0), thickness=2)
        cv2.putText(frame, "Z: " + "{:7.2f}".format(euler_angle[2, 0]), (20, 470),
cv2.FONT_HERSHEY_SIMPLEX,
            0.75, (0, 0, 0), thickness=2)
    if xx>18:
        #print("x",xx)
        #print("y",yy)
        cv2.putText(frame, "YOUR HEAD IS DOWN! ", (10,
80),cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
        cv2.putText(frame, "DROWSINESS ALERT!", (10, 57),
            cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
        s=threading.Thread( name='sound_alarm',target=sound_alarm,args=(path,) )
        s.start()

    ###########################
    # compute the convex hull for the left and right eye, then
    # visualize each of the eyes
    leftEyeHull = cv2.convexHull(leftEye)
    rightEyeHull = cv2.convexHull(rightEye)
    mouthHull = cv2.convexHull(mouth)
    cv2.drawContours(frame, [leftEyeHull], -1, (0, 255, 255), 1)
    cv2.drawContours(frame, [rightEyeHull], -1, (0, 255, 255), 1)
    cv2.drawContours(frame, [mouthHull], -1, (0, 255, 0), 1)

    # check to see if the eye aspect ratio is below the blink
    # threshold, and if so, increment the blink frame counter
    if ear < EYE_AR_THRESH:
        COUNTER += 1
        cv2.putText(frame, "Eyes Closed ", (10, 30),cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0,
255), 2)

        # if the eyes were closed for a sufficient number of
        if COUNTER >= EYE_AR_CONSEC_FRAMES:
            # draw an alarm on the frame
            cv2.putText(frame, "DROWSINESS ALERT!", (10, 50),
                cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
            s=threading.Thread( name='sound_alarm',target=sound_alarm,args=(path,) )
            s.start()
            #print("eyes",COUNTER)

    # otherwise, the eye aspect ratio is not below the blink
    # threshold, so reset the counter and alarm
    else:
        COUNTER = 0
        cv2.putText(frame, "Eyes Open ", (10, 30),cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255,
0), 2)

    cv2.putText(frame, "EAR: {:.2f}".format(ear), (480, 30),
        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)

    # yawning detections

    if mouEAR > MOU_AR_THRESH:
        cv2.putText(frame, "Yawning ", (10, 70),cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255),
2)
        yawnStatus = True
        output_text = "Yawn Count: " + str(yawns + 1)
        cv2.putText(frame, output_text, (10,100),cv2.FONT_HERSHEY_SIMPLEX, 0.7,(255,0,0),2)
        #print(yawns)
    else:
        yawnStatus = False

    if prev_yawn_status == True and yawnStatus == False:
        yawns+=1
        if yawns>2:
            s=threading.Thread( name='sound_alarm',target=sound_alarm,args=(path,) )
            s.start()
            yawns=0
            #print(yawns)

    cv2.putText(frame, "MAR: {:.2f}".format(mouEAR), (480, 60),
        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
    #cv2.putText(frame,"Lusip Project @
Swarnim",(370,470),cv2.FONT_HERSHEY_COMPLEX,0.6,(153,51,102),1)
    # show the frame
    cv2.imshow("Frame", frame)
    key = cv2.waitKey(1) & 0xFF
    # if the `q` key was pressed, break from the loop
    if key == ord("q"):
    break
# do a bit of cleanup
cv2.destroyAllWindows()
cap.release()
```