ABSTRACT

Accidents due to Drowsiness has become the most common tragedy in the world. So we have come up with a device which will be installed in the front of the driver's seat which will be instantaneously capturing the face of the driver in order to detect drowsiness while driving. Since the camera is installed in-front of driver, if he is drowsy or sleepy, by the rate of movement or the blink of the eyes using few algorithms, the software detects that he is drowsy and alarms and becomes aware or take some rest.

The main idea behind this project is to develop a nonintrusive system which can detect fatigue of any human and can issue a timely warning. Drivers who do not take regular breaks when driving long distances run a high risk of becoming drowsy a state which they often fail to recognize early enough. According to the expert's studies show that around one quarter of all serious motorway accidents are attributable to sleepy drivers in need of a rest, meaning that drowsiness causes more road accidents than drink-driving. This system will monitor the driver eyes using a camera and by developing an algorithm we can detect symptoms of driver fatigue early enough to avoid the person from sleeping. So, this project will be helpful in detecting driver fatigue in advance and will give warning output in form of alarm and pop-ups.

CHAPTER 1:

INTRODUCTION

Real Time Drowsiness behaviours which are related to fatigue are in the form of eye closing, head nodding or the brain activity. Hence, we can either measure change in physiological signals, such as brain waves, heart rate and eye blinking to monitor drowsiness or consider physical changes such as sagging posture, leaning of driver's head and open/closed state of eyes.

Various experiments have been done earlier with regard to the drowsiness detection of driver. In the past few years, many countries became curious to pay high attention towards driver's safety problems. Researchers have been making various efforts to invent techniques for the detection of drowsy driver such as monitoring of road and physiological techniques which requires the contact of electrode with our body such as chest, face making it an implantable method. In this thesis, we described the direct method which can detect drowsiness without any help of electrode using various detection function.

As we have discussed there are many conditions to be considered to detect, but we are taking eye blinks into consideration because head nodding won't be a specifically applied since driver nods even if he is not sleepy and we can't take brain activity in the picture because it needs many hardware products to be installed over driver's head which is not comfortable and feasible and makes the user annoying.

Stages of Drowsiness

Drowsiness will not appear instantly. But, it will emerge symptoms which generally occurred in every driver.

- Unable to focus, eyes blinking fast, eyelids are so heavy.
- Day dreaming, hard to concentrate.
- Unable to remember road which has been through.
- Yawning and rubbing eyes.
- Head nodding.

- Out of lane.
- Temperamental and feeling restless.

Stage of Drowsiness	Behavior examples
1	Eye moving fast, many movements shown
2	Eye moving slower, mouth open little
3	Mouth mumbling, touching face, fixing the seating
4	Head shaking, yawning, blinking slower
5	Eye closed, head nodding

Table 1.1 Stages of Drowsiness

Before directly going to the drowsiness detection, we went with just blink detection and counting the blinks. We built upon this knowledge and develop a computer vision application that is capable of detecting and counting blinks in video streams using facial landmarks and OpenCV.

To build our blink detector, we'll be computing a metric called the **eye aspect ratio (EAR)**,

Real-Time Eye Blink Detection Using Facial Landmarks. Unlike traditional image processing methods for computing blinks which typically involve some combination of:

1. Eye localization.
2. Thresholding to find the whites of the eyes.
3. Determining if the "white" region of the eyes disappears for a period of time (indicating a blink).

4. The eye aspect ratio is instead a much more elegant solution that involves a very simple calculation based on the ratio of distances between facial landmarks of the eyes.
   This method for eye blink detection is fast, efficient, and easy to implement.

Eye blink detection with OpenCV, Python, and dlib

Our blink detection blog post is divided into four parts.

In the first part we'll discuss the eye aspect ratio and how it can be used to determine if a person is blinking or not in a given video frame.

From there, we'll write Python, OpenCV, and dlib code to (1) perform facial landmark detection and (2) detect blinks in video streams.

Based on this implementation we'll apply our method to detecting blinks in example webcam streams along with video file

Understanding the "Eye Aspect Ratio"

Each eye is represented by 6 (x, y)-coordinates, starting at the left-corner of the eye, and then working clockwise around the remainder of the region:

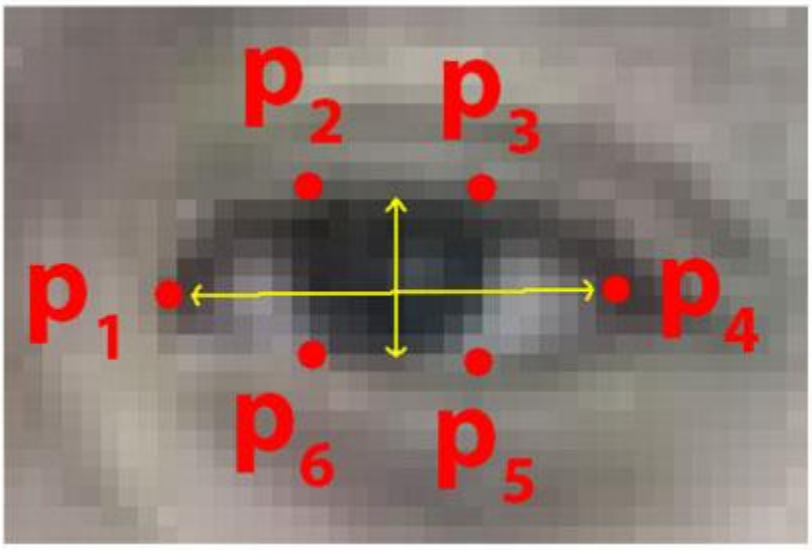


Figure 1.1 Eye aspect ratio

There is a relation between the **width** and the **height** of these coordinates.

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

Where p1, …, p6 are 2D facial landmark locations.

The numerator of this equation computes the distance between the vertical eye landmarks while the denominator computes the distance between horizontal eye landmarks, weighting the denominator appropriately since there is only one set of horizontal points but *two* sets of vertical points.

Well, as we'll find out, the eye aspect ratio is approximately constant while the eye is open, but will rapidly fall to zero when a blink is taking place.

Using this simple equation, we can avoid image processing techniques and simply rely on the ratio of eye landmark distances to determine if a person is blinking.

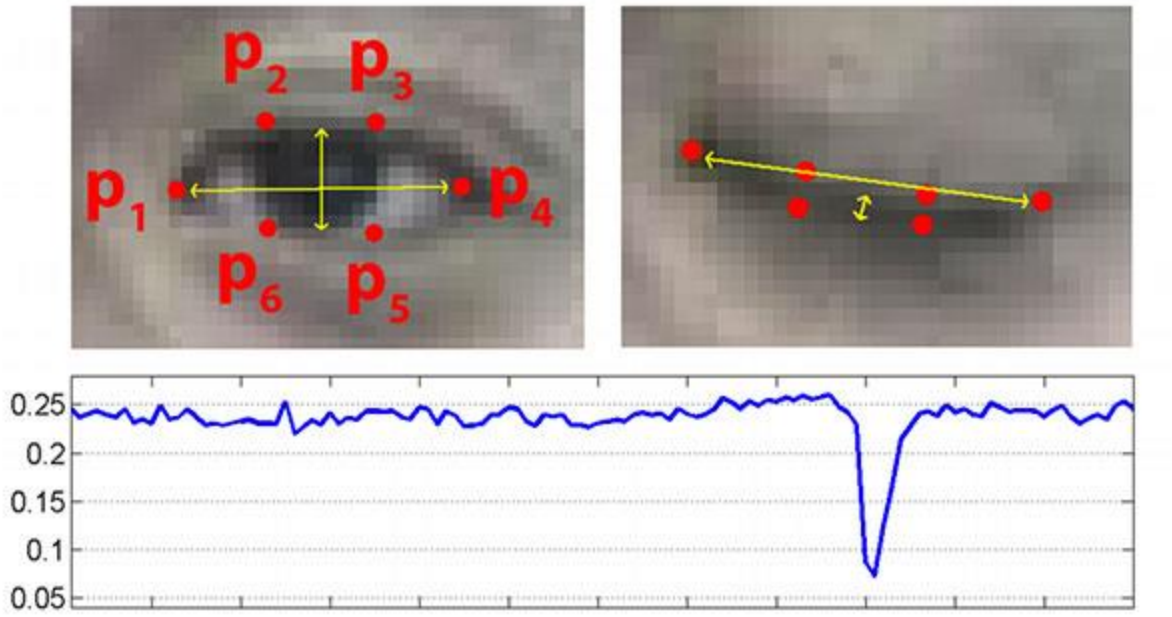


Figure 1.2 Top-left: A visualization of eye landmarks when then the eye is open. Top-right: Eye landmarks when the eye is closed. Bottom: Plotting the eye aspect ratio over time. The dip in the eye aspect ratio indicates a blink.

On the top-left we have an eye that is fully open — the eye aspect ratio here would be large(r) and relatively constant over time.However, once the person blinks (top-right) the eye aspect ratio decreases dramatically, approaching zero.The bottom figure plots a graph of the eye aspect ratio over time for a video clip. As we can see, the eye aspect ratio is constant, then rapidly drops close to zero, then increases again, indicating a single blink has taken place.

So using this we detected and counted the eye blink and next moving with the main part i.e. detecting the drowsiness. So what was the upgradation to detect drowsiness was changing with the time frame.

So what time frame does is when the eye is closed for a particular or given specific time it detects as drowsiness. Whenever the requirements used to meet it is given such a way that the driver should be alerted with a alarm.

And later we added a feature which also included mouth which tends to Yawn during drowsiness. If the driver speaks it won't detect because we have used the same concept of time frame, when a person yawns, he tends to open his mouth wide for a while which is way longer than while he speaks so adding to one the feature project

Figure 1.3 Demonstration of Eye blink detection

CHAPTER 2:

STATISTICS

- According to the National Highway Traffic Safety Administration (NHTSA) in 2015 there were 824 fatalities (2.3 percent of all fatalities) that were drowsy-driving-related, matching the annual 2011 to 2015 average. In 2015, there were 736 crashes that were attributed to drowsy driving, slightly higher than the prior five-year average of 732 crashes.

- A December 2016 study by the AAA Foundation for Traffic and Safety found that drivers who usually sleep for less than 5 hours daily, drivers who have slept for less than 7 hours in the past 24 hours, and drivers who have slept for 1 or more hours less than their usual amount of sleep in the past 24 hours have significantly elevated crash rates. The estimated rate ratio for crash involvement associated with driving after only 4-5 hours of sleep compared with 7 hours or more is similar to the U.S. government's estimates of the risk associated with driving with a blood alcohol concentration equal to or slightly above the legal limit for alcohol in the U.S.

- The Governors Highway Safety Association issued a report in August 2016 concluding that the estimated annual societal cost of fatigue-related fatal and injury crashes was $109 billion. This figure does not include property damage.

- A 2014 AAA Traffic Safety Foundation study found that 37 percent of drivers report having fallen asleep behind the wheel at some point in their lives. An estimated 21 percent of fatal crashes, 13 percent of crashes resulting in severe injury and 6 percent of all crashes, involve a drowsy driver.

- A 2013 study by the Federal Rail Administration found that fatigue greatly increases the chances of an accident in which human factors play a role, with the risk of such an accident rising from 11 percent to 65 percent.

- Although sleepiness can affect all types of crashes during the entire day and night, drowsy-driving crashes most frequently occur between midnight and 6 a.m., or in the late-afternoon, according to NHTSA.

- NHTSA found that many drowsy-driving crashes involve a single vehicle, with no passengers besides the driver, running off the road at a high rate of speed with no evidence of braking.

- The beginning of daylight savings is linked to an increase in auto accidents, according to an analysis by the University of British Columbiaand a study by researchers at John Hopkins and Stanford University.

- In 2016, drowsiness or sleepiness was a factor for 2.5 percent of drivers and motorcycle operators involved in fatal crashes, as shown in the chart below.

Researchers estimate that more than 70 million Americans suffer from a sleep disorder (Institute of Medicine, 2005).

Did You Know?

- An estimated 1 in 25 adult drivers (aged 18 years or older) report having fallen asleep while driving in the previous 30 days.[1,2]
- The National Highway Traffic Safety Administration estimates that drowsy driving was responsible for 72,000 crashes, 44,000 injuries, and 800 deaths in 2013.[3] However, these numbers are underestimated and up to 6,000 fatal crashes each year may be caused by drowsy drivers.[4-5]

Who's more likely to drive drowsy?

- Drivers who do not get enough sleep.
- Commercial drivers who operate vehicles such as tow trucks, tractor trailers, and buses.
- Shift workers (work the night shift or long shifts).

- Drivers with untreated sleep disorders such as one where breathing repeatedly stops and starts (sleep apnea).
- Drivers who use medications that make them sleepy.

Learn the warning signs of drowsy driving—

- Yawning or blinking frequently.
- Difficulty remembering the past few miles driven.
- Missing your exit.
- Drifting from your lane.
- Hitting a rumble strip on the side of the road.

CHAPTER 3:

LITERATURE REVIEW

In this section, we have discussed various methodologies that have been proposed by researchers for drowsiness detection and blink detection during the recent years.

In 2015, Sayani Ghosh, Tanaya Nandy and Nilotpal Manna proposed a scheme that to detect and track eye images with complex background, distinctive features of user eye are used. Generally, an eye-tracking and detection system can be divided into four steps: (i) Face detection, (ii) Eye region detection, (iii) Pupil detection and (iv) Eye tracking.

Image processing technique is incorporated for detection of these. Camera is incorporated in the dashboard of vehicle which takes the images of the driver regularly at certain interval. From the images first the face portion is recognized from the complex background. It is followed by eye region detection and thereafter the pupil or eyelid detection. The detection algorithm finally detects the eyelid movement or closeness and openness of eyes. In the proposed method, eye detection and tracking are applied on testing sets, gathered from different images of face data with complex backgrounds. This method combines the location and detection algorithm with the grey prediction for eye tracking. The accuracy and robustness of the system depends on consistency of image acquisition of the driver face in real time under variable and complex background. For this purpose the driver's face is illuminated using a near-infrared (NIR)illuminator.

In 2018, Sarmad Al-gawwam and Mohammed Benaissa proposed a scheme that Blinking is a natural eye motion defined as the rapid closing and opening of the eyelid of a human eye. The proposed technique is composed of four main steps. These steps are applied to each frame of an input video. This method uses Zface for automatic tracking of facial landmarks to localise the eyes and eyelid contours. The robustness of this method for 3D registration and reconstruction from the 2D video has been validated in a series of experiments. Using ZFace, no pre-training is required to perform 3D registration from the 2D video. A combined 3D supervised descent method is employed to define the shape model by a 3D mesh. ZFace registers a dense parameterised shape model to an image such that its landmarks correspond to consistent locations on the face. ZFace is used to track 49 facial landmarks from videos, where eye features

are detected for each video frame, and the eye-opening state is estimated using the vertical distance (d) between eyelids.

$$d = \sqrt{(P2.x - P1.x)^2 + (P2.y - P1.y)^2}$$

where P1 and P2 are the eye landmark points. It is assumed that the obtained signal from the distance between the upper and lower eyelids is mostly fixed when an eye is open and approaches zero when the eye is closing. This is relatively insensitive to body and head positions. The resulting signal is affected by interference primarily caused by saccadic eye movements and facial expressions. These interferences are filtered while the shape of the signal is maintained. Lastly, the analysis of the filtered signal can be implemented to detect eye blinks represented as peaks representing the distance change between eyelids.

Tanmay Rajpathaka, Ratnesh Kumarb and Eric Schwartzb proposed a scheme of Eye Detection Using Morphological and Color Image Processing such that they discussed commonly used approaches for passive eye detection include the template matching method, eigenspace method, and Hough transform-based method .

In the template matching method, segments of an input image are compared to previously stored images, to evaluate the similarity of the counterpart using correlation values. The problem with simple template matching is that it cannot deal with eye variations in scale, expression, rotation and illumination. Use of multiscale templates was somewhat helpful in solving the previous problem in template matching.

A method of using deformable templates is proposed by Yuille et al. This provides the advantage of finding some extra features of an eye like its shape and size at the same time. But the rate of success of this approach depends on initial position of the template. Pentland et al. Proposed an eigenspace method for eye and face detection. If the training database is variable with respect to appearance, orientation, and illumination, then this method provides better performance than simple template matching. But the performance of this method is closely related to the training set used and this method also requires normalized sets of training and test images with respect to size and orientation.

Another popular eye detection method is obtained by using the Hough transform. This method is based on the shape feature of an iris and is often used for binary valley or edge maps. The drawback of this approach is that the performance depends on threshold values used for binary conversion of the valleys.

Apart from these three classical approaches, recently many other image-based eye detection techniques have been reported. Feng and Yuen used intensity, the direction of the line joining the centers of the eyes, the response of convolving an eye variance filter with the face image, and the variance projection function (VPF) technique to detect eyes. Zhou and Geng extended the idea of VPF to the generalized projection function (GPF) and showed with experimental results that the hybrid projection function (HPF), a special case of GPF, is better than VPF and integral projection function (IPF) for eye detection.

Kawaguchi and Rizon located the iris using intensity and edge information. They used a feature template, a separability filter, the Hough transform, and template matching in their algorithm. Sirohey and Rosenfeld proposed an eye detection algorithm based on linear and nonlinear filters. Huang and Wechsler's method , used genetic algorithms and built decision trees to detect eyes. For the purpose of face detection, Wu and Zhou employed size and intensity information to find eye-analog segments from a gray scale image, and exploited the special geometrical relationship to filter out the possible eye-analog pairs. Han et al. applied such techniques as morphological closing, conditional dilation and a labeling process to detect eye-analog segments. Hsu et al. used color information for eye detection.

In 2015, Keval Lakhani, Aunsh Chaudhari, Kena Kothari, Harish Narula proposed a scheme for Image Capturing using Blink Detection. There are several ways that can be employed to perform eye blink detection using Image Processing. One of them is using the Haar Cascade Classifier. This technique has the following major steps: 1) Frame Capturing 2) Face Detection 3) Eye Detection 4) Eye Tracking 5) Eye Blink Detection.

The first step in this process is the intialization where a video of the individual's face is taken and correspondingly, a process Frame method is used to create frames from this captured video. The resultant colored frames are converted to gray scale by eliminating the component of luminance. Next, for face detection we use the Haar classifier that detects an object on the basis of a facial feature. The feature is detected if the classifier is regionalizing a particular area that

has the highest probability of containing the sensed feature. Moving forward, the classifier detects the face and marks it with a colored rectangle that is later useful to approximate an axis for eye detection. The detection of the eyes involves training the Haar classifiers. Once the face is detected, the AdaBoost and Haar feature algorithms train the classifier with the help of two sets of images. The first one contains the image scene, whereas the second does not contain the object at all. Consequently, having trained the Cascade classifer, the eyes are detected along the axis of the face recognition rectangle and another colored rectangle is formed bordering the eyes, showing that the eyes have been detected successfully.

Eye tracking relates to extracting features, parts of the eye in order to determine their movement. The two parts that are most important in this method are – the corneal reflection and pupil-center. With the backing of an accurate location of these features and the mathematical trigonometric calculations involved, the point of regard for a pair of eyes can be found. The data obtained at the end of this procedure must be used in a sensible way because eye movements can either be voluntary or involuntary and experimental results must be evaluated accordingly. Finally, coming to the crux of the matter at hand, eye blink detection is performed using the frames that have been detected earlier. With the help of these frames, the status of the eye can be determined– whether it is open or closed. Applying binarization to frames, thresholding is performed. In binary frames, 0 represents the black color and 1 represents the white color for each pixel. To check if the eye is blinking, the length and width of the portion below the eyebrows is determined.

Keeping a count on the number of gray and black points, if the number black points detected are greater than a predetermined number, the eye is closed else it is open. The eye blink detection mechanism in this technique is subject to lighting conditions as well as the distance between the detector and the eye. If the distance is long, the process of recognition is extremely difficult. The accuracy differs depending on the lighting of the environment – natural or artificial. A significant takeaway from the IP technique is that the detection efficiency can be improved by applying the Medium Blur Filter on the binary frames. The filter aids in noise detection that is a typical pre-processing method to improve accuracy of blink detection.

CHAPTER 4:

DATASET CREATION AND TRAINING FOR FACIAL LANDMARK

Dlib is a pretty famous and awesome machine learning library written in C++. It implements a wide range of algorithms that can be used either on the desktop and mobile platforms.

4.1 Face Landmark Localization

The process that is able to extrapolate a set of key points from a given face image, is called Face Landmark Localization (or Face Alignment).

The landmarks (key points) that we are interested in, are the one that describes the shape of the face attributes like: eyes, eyebrows, nose, mouth, and chin. These points gave a great insight about the analyzed face structure, that can be very useful for a wide range of applications, including: face recognition, face animation, emotion recognition, blink detection, and photography.

There are a lot of methods that are able to detect these points: some of them achieve superior accuracy and robustness by analysing a 3D face model extracted from a 2D image, others rely on the power of CNNs (Convolutional Neural Networks) or RNNs (Recurrent Neural Networks), and the other one utilize simple (but fast) features to estimate the location of the points.

The Face Landmark Detection algorithm offered by Dlib is an implementation of the Ensemble of Regression Trees (ERT) presented in 2014 by Kazemi and Sullivan. This technique utilize simple and fast feature (pixel intensities differences) to directly estimate the landmark positions. These estimated positions are subsequently refined with an iterative process

done by a cascade of regressors. The regressors produces a new estimate from the previous one, trying to reduce the alignment error of the estimated points at each iteration. The algorithm is blazing fast, in fact it takes about 1–3ms to detect a set of 68 landmarks on a given face.

4.2 One Millisecond Face Alignment With An Ensemble Of Regression Trees

Here ia an algorithm that performs face alignment in milliseconds and achieves accuracy superior or comparable to state-of-the-art methods on standard datasets. The speed gains over previous methods is a consequence of identifying the essential components of prior face alignment algorithms and then incorporating them in a streamlined formulation into a cascade of high capacity regression functions learnt via gradient boosting. In our case each regression function in the cascade efficiently estimates the shape from an initial estimate and the intensities of a sparse set of pixels indexed relative to this initial estimate. In particular, we incorporate into our learnt regression functions two key elements that are present in several of the successful algorithms cited and we detail these elements now. The first revolves around the indexing of pixel intensities relative to the current estimate of the shape. The extracted features in the vector representation of a face image can greatly vary due to both shape deformation and nuisance factors such as changes in illumination conditions. This makes accurate shape estimation using these features difficult. The dilemma is that we need reliable features to accurately predict the shape, and on the other hand we need an accurate estimate of the shape to extract reliable features. Instead of regressing the shape parameters based on features extracted in the global coordinate system of the image, the image is transformed to a normalized coordinate system based on a current estimate of the shape, and then the features are extracted to predict an update vector for the shape parameters. This process is usually repeated several times until convergence.

The second considers how to combat the difficulty of the inference/prediction problem. At test time, an alignment algorithm has to estimate the shape, a high dimensional vector, that best agrees with the image data and our model of shape. The problem is non-convex with many local optima. Successful algorithms handle this problem by assuming the estimated shape must lie in a linear subspace, which can be discovered, for example, by finding the principal components of the training shapes. This assumption greatly reduces the number of potential shapes considered during inference and can help to avoid local optima. Recent work uses the fact that a certain class of regressors are guaranteed to produce predictions that lie in a linear

subspace defined by the training shapes and there is no need for additional constraints. Crucially, our regression functions have these two elements. Allied to these two factors is our efficient regression function learning. We optimize an appropriate loss function and perform feature selection in a data-driven manner. In particular, we learn each regressor via gradient boosting with a squared error loss function, the same loss function we want to minimize at test time. The sparse pixel set, used as the regressor's input, is selected via a combination of the gradient boosting algorithm and a prior probability on the distance between pairs of input pixels. The prior distribution allows the boosting algorithm to efficiently explore a large number of relevant features. The result is a cascade of regressors that can localize the facial landmarks when initialized with the mean face pose. This algorithm is used to precisely estimate the position of facial landmarks in a computationally efficient way. This method utilizes a cascade of regressors. Here we will describe the details of the form of the individual components of the cascade and how we perform training.

4.2.1 The cascade of regressors

To begin we introduce some notation. Let xi ∈ R be the x, y-coordinates of the ith facial landmark in an image I. Then the vector S = (x T 1 , x T 2 , . . . , x T p) T ∈ R 2p denotes the coordinates of all the p facial landmarks in I. Frequently, we refer to the vector S as the shape. We use Sˆ(t) to denote our current estimate of S. Each regressor, rt, in the cascade predicts an update vector from the image and Sˆ(t) that is added to the current shape estimate Sˆ(t) to improve the estimate:

$$\hat{S}(t+1) = \hat{S}(t) + r_t(I, \hat{S}(t))$$

The critical point of the cascade is that the regressor rt makes its predictions based on features, such as pixel intensity values, computed from I and indexed relative to the current shape estimate Sˆ(t) . This introduces some form of geometric invariance into the process and as the cascade proceeds one can be more certain that a precise semantic location on the face is being indexed. Later we describe how this indexing is performed. Note that the range of outputs expanded by the ensemble is ensured to lie in a linear subspace of training data if the initial estimate Sˆ(0) belongs to this space. We therefore do not need to enforce additional constraints on the predictions which greatly simplifies our method. The initial shape can simply be chosen as the mean shape of the training data centered and scaled according to the bounding box output of

a generic face detector. To train each rt we use the gradient tree boosting algorithm with a sum of square error loss . We now give the explicit details of this process.

4.2.2 Learning each regressor in the cascade

Assume we have training data (I1, S1), . . . ,(In, Sn) where each Ii is a face image and Si its shape vector. To learn the first regression function r0 in the cascade we create from our training data triplets of a face image, an initial shape estimate and the target update step, that is,

$(I_{\pi i}, \hat{S}^{(0)}_i, \Delta S^{(0)}_i)$

where $\pi i \in \{1, \ldots, n\}$ (2) $\hat{S}^{(0)}_i \in \{S1, \ldots, Sn\} \backslash S_{\pi i}$ and (3) $\Delta S^{(0)}_i = S_{\pi i} - \hat{S}^{(0)}_i$ (4) for i = 1, . . . , N. We set the total number of these triplets to N = nR where R is the number of initializations used per image Ii . Each initial shape estimate for an image is sampled uniformly from {S1, . . . , Sn} without replacement. From this data we learn the regression function r0 (see algorithm 1), using gradient tree boosting with a sum of square error loss. The set of training triplets is then updated to provide the training data, $(I_{\pi i}, \hat{S}^{(1)}_i, \Delta S^{(1)}_i)$, for the next regressor r1 in the cascade by setting (with t = 0) $\hat{S}^{(t+1)}_i = \hat{S}^{(t)}_i + r_t(I_{\pi i}, \hat{S}^{(t)}_i)$ (5) $\Delta S^{(t+1)}_i = S_{\pi i} - \hat{S}^{(t+1)}_i$.

This process is iterated until a cascade of T regressors r0, r1, . . . , rT −1 are learnt which when combined give a sufficient level of accuracy. As stated each regressor rt is learned using the gradient boosting tree algorithm. It should be remembered that a square error loss is used and the residuals computed in the innermost loop correspond to the gradient of this loss function evaluated at each training sample. Included in the statement of the algorithm is a learning rate parameter $0 < \nu \leq 1$ also known as the shrinkage factor. Setting $\nu < 1$ helps combat over-fitting and usually results in regressors which generalize much better than those learnt with $\nu = 1$.

4.2.3 Algorithm Learning rt in the cascade

Have training data $\{(I_{\pi i}, \hat{S}^{(t)}_i, \Delta S^{(t)}_i)\}_{i=1}^{N}$ and the learning rate (shrinkage factor) $0 < \nu < 1$.

1. Initialise f0(I, Sˆ(t)) = arg min γ∈R2p X N i=1 kΔS (t) i − γk 2

2. for k = 1, . . . , K:

(a) Set for i = 1, . . . , N rik = ΔS (t) i − fk−1(Iπi , Sˆ (t) i)

(b) Fit a regression tree to the targets rik giving a weak regression function gk(I, Sˆ(t)).

(c) Update fk(I, Sˆ(t)) = fk−1(I, Sˆ(t)) + ν gk(I, Sˆ(t))

3. Output rt(I, Sˆ(t)) = fK(I, Sˆ(t))

4.3 Dlib pre-trained Models

The author of the Dlib library (Davis King) has trained two shape predictor models on the iBug 300-W dataset, that respectively localize 68 and 5 landmark points within a face image.

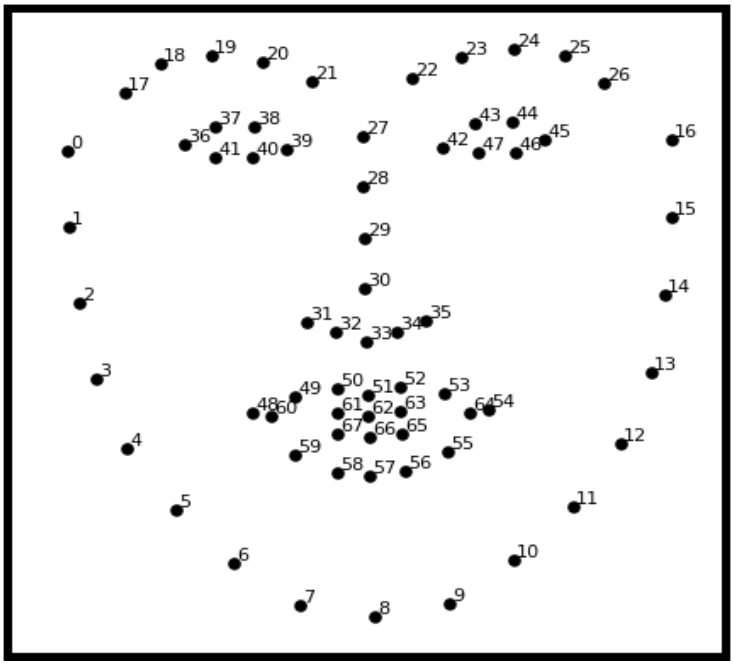


Fig 4.1 Facial landmark

The set of 68-points detected by the pre-trained Dlib shape_predictor_68

In this scenario we will consider only the shape_predictor_68 model. Basically, a shape predictor can be generated from a set of images, annotations and training options. A single annotation consists of the face region, and the labelled points that we want to localize. The face region can be easily obtained by any face detection algorithm (like OpenCV HaarCascade, Dlib HOG Detector, CNN detectors, …), instead the points have to be manually labelled or detected by already-available landmark detectors and models (e.g. ERT with SP68). Lastly, the training options are a set of parameters that defines the characteristics of the trained model. These parameters can be properly fine-tuned in order to get the desired behaviour of the generated model, more or less.

4.3.1 Understanding the Training Options

The training process of a model is governed by a set of parameters. These parameters affect the size, accuracy and speed of the generated model.

The most important parameters are:

- **Tree Depth**—Specifies the depth of the trees used in each cascade. This parameter represent the **"**capacity**"** of the model. An optimal value (in terms of accuracy) is 4, instead a value of 3 is a good tradeoff between accuracy and model-size.
- **Nu**—Is the regularization parameter. It determines the ability of the model to generalize and learn patterns instead of fixed-data. Value close to 1 will emphasize the learning of fixed-data instead of patterns, thus raising the chances for over-fitting to occur. Instead, an optimal nu value of 0.1 will make the model to recognize patterns instead of fixed-situations, totally eliminating the over-fitting problem. The amount of training samples can be a problem here, in fact with lower nu values the model needs a lot (thousands) of training samples in order to perform well.
- **Cascade Depth**—Is the number of cascades used to train the model. This parameter affect either the size and accuracy of a model. A good value is about 10-12, instead a value of 15 is a perfect balance of maximum accuracy and a reasonable model-size.

- **Feature Pool Size**—Denotes the number of pixels used to generate the features for the random trees at each cascade. Larger amount of pixels will lead the algorithm to be more robust and accurate but to execute slower. A value of 400 achieves a great accuracy with a good runtime speed. Instead, if speed is not a problem, setting the parameter value to 800 (or even 1000) will lead to superior precision. Interestingly, with a value between 100 and 150 is still possible to obtain a quite good accuracy but with an impressing runtime speed. This last value is particularly suitable for mobile and embedded devices applications.

- **Num Test Splits**—Is the number of split features sampled at each node. This parameter is responsible for selecting the best features at each cascade during the training process. The parameter affects the training speed and the model accuracy. The default value of the parameter is 20. This parameter can be very useful, for example, when we want to train a model with a good accuracy and keep its size small. This can be done by increasing the amount of num split test to 100 or even 300, in order to increase the model accuracy and not its size.

- **Oversampling Amount**—Specifies the number of randomly selected deformations applied to the training samples. Applying random deformations to the training images is a simple technique that effectively increase the size of the training dataset. Increasing the value of the parameter to 20 or even 40 is only required in the case of small datasets, also it will increase the training time considerably (so be careful). In the latest releases of the Dlib library, there is a new training parameter: the oversampling jittering amount that apply some translation deformation to the given bounding boxes in order to make the model more robust against eventually misplaced face regions.

4.3.2 Getting the Data Set ready

In order to replicate the Dlib results, we have to utilize the images and annotations inside the iBug 300W dataset (. The dataset consists of the combination of four major datasets: afw, helen, ibug, and lfpw.

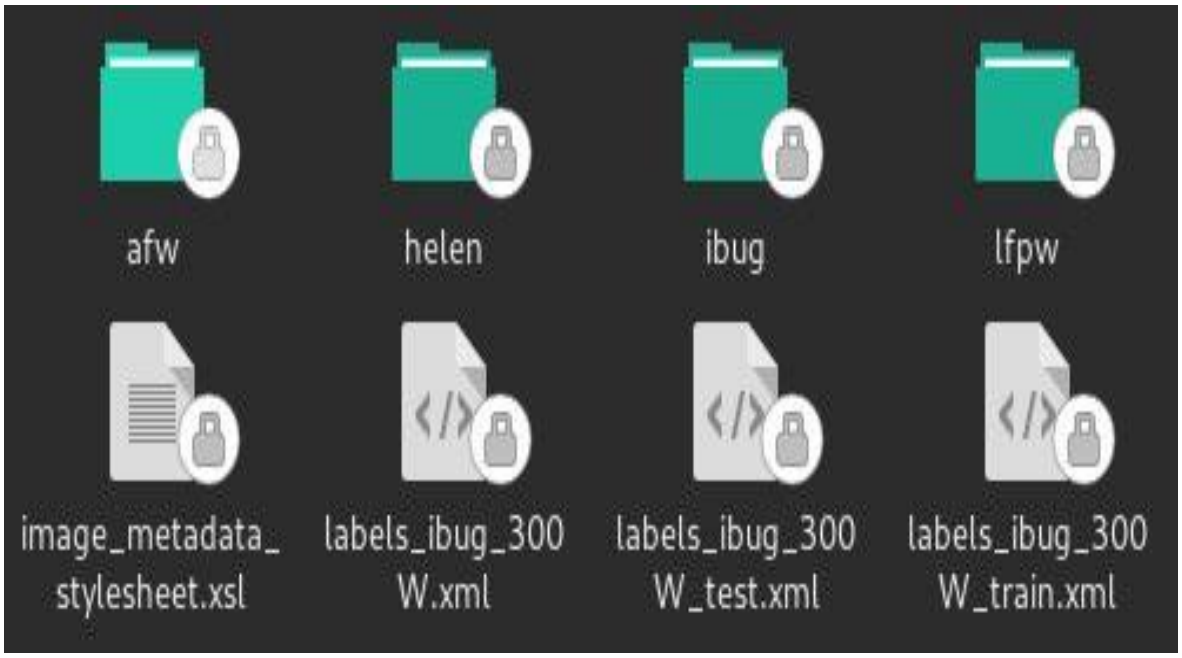


Fig 4.2 iBug 300W Dataset

The files (annotations) that we need to train and test the models are the: labels_ibug_300W_train.xml and labels_ibug_300W_test.xml.

Here the files corresponds to datasets prepared using ibug 300W images and an annotation tool. Img Lab tool can be used to annotate the images and create the dataset.The steps involved in the creation of the datset are as follows:

1) creating an outline of the face
2) Annotating the required parts on the face like eyes, mouth etc.
3) Drawing a bounding box around the face.
4) Downloading the xml file.

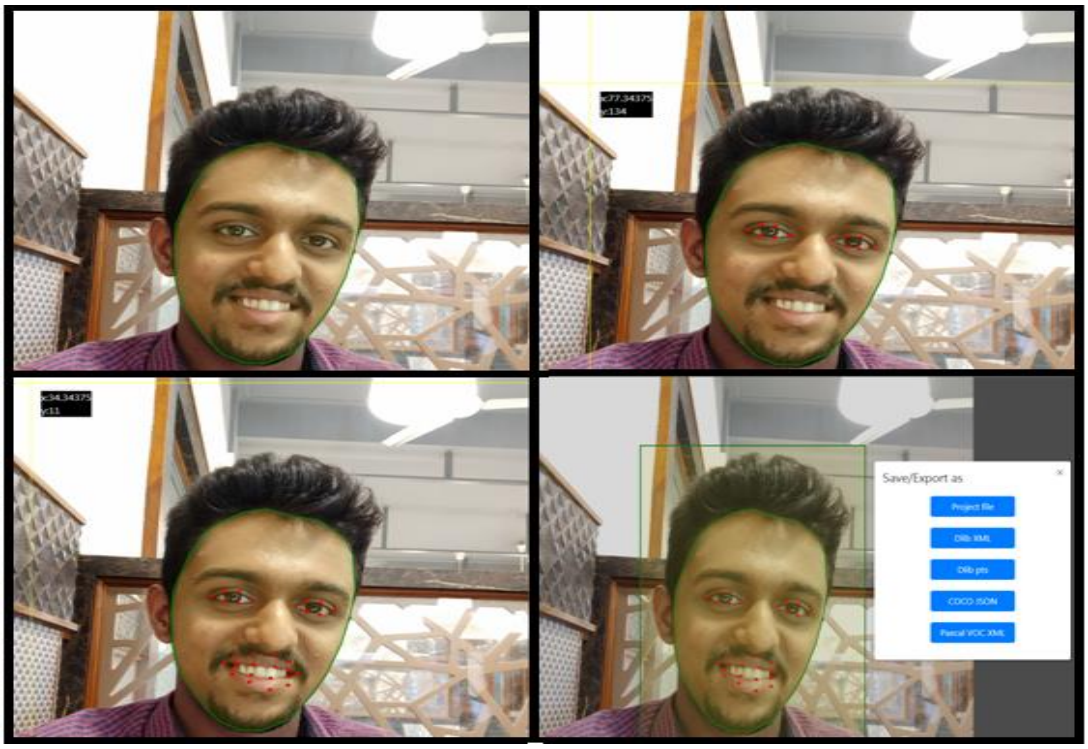


Fig 4.3 Steps involved in creating the dataset

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<?xml-stylesheet type='text/xsl'
href='image_metadata_stylesheet.xsl'?>
<dataset>
<name>dlib face detection dataset generated by ImgLab</name>
<comment>
    This dataset is manually crafted or adjusted using ImgLab web
tool
    Check more detail on
https://github.com/NaturalIntelligence/imglab
</comment>
<images>    <image file='sanjay.jpg'>
            <box top='1219.999999999999' left='1214.999999999999'
width='1369.999999999999' height='2049.9999999999986'>
                <label>unlabelled</label>
                <part name='0' x='1521' y='2024'/>
                <part name='1' x='1621' y='2064'/>
                <part name='2' x='2126' y='2114'/>
                <part name='3' x='2421' y='2164'/>
                <part name='4' x='2221' y='2079'/>
                <part name='5' x='2326' y='2089'/>
                <part name='6' x='2221' y='2169'/>
                <part name='7' x='2316' y='2194'/>
                <part name='8' x='1606' y='2069'/>
                <part name='9' x='1711' y='2089'/>
                <part name='10' x='1616' y='1974'/>
                <part name='11' x='1716' y='1984'/>
                <part name='12' x='1606' y='2644'/>
                <part name='13' x='1701' y='2644'/>
                <part name='14' x='1826' y='2654'/>
                <part name='15' x='1986' y='2679'/>
                <part name='16' x='2086' y='2709'/>
                <part name='17' x='2151' y='2739'/>
                <part name='18' x='1671' y='2699'/>
                <part name='19' x='1806' y='2754'/>
                <part name='20' x='1951' y='2784'/>
                <part name='21' x='2076' y='2769'/>
                <part name='22' x='1681' y='2574'/>
                <part name='23' x='1826' y='2554'/>
                <part name='24' x='1981' y='2574'/>
                <part name='25' x='2096' y='2629'/>
                <part name='26' x='1666' y='2749'/>
                <part name='27' x='1796' y='2819'/>
                <part name='28' x='1946' y='2889'/>
                <part name='29' x='2081' y='2849'/>
            </box>
            <box top='634.9999999999995' left='919.9999999999993'
width='1999.9999999999986' height='2694.999999999998'>
                <label>unlabelled</label>
            </box>
        </image>
</images></dataset>
```

Fig 4.4 Dataset xml file format

4.3.3 Training the Models

Imagine we are interested into training a model that is able to localize only the landmarks of the left and right eye. To do this, we have to edit the iBug training annotations by selecting only the relevant points:

Editing an xml file by selecting only the desired parts (landmarks).This can be done by calling the slice_xml() function, that creates a new xml-file with only the selected landmarks.

4.4 Conclusions

By properly fine-tuning the training options is possible to customize the training process in such a way that satisfy the constraints of the system we are developing. Such constrains can be about executive speed, memory and storage consumption, and overall accuracy and robustness. That characterize the platform we are developing with (desktop, mobile, and embedded—In general, different platforms have different sets of quality requirements).Moreover, by selecting only the relevant landmarks is possible to create specific models that localize a particular subset of landmarks, thus eliminating unnecessary point.

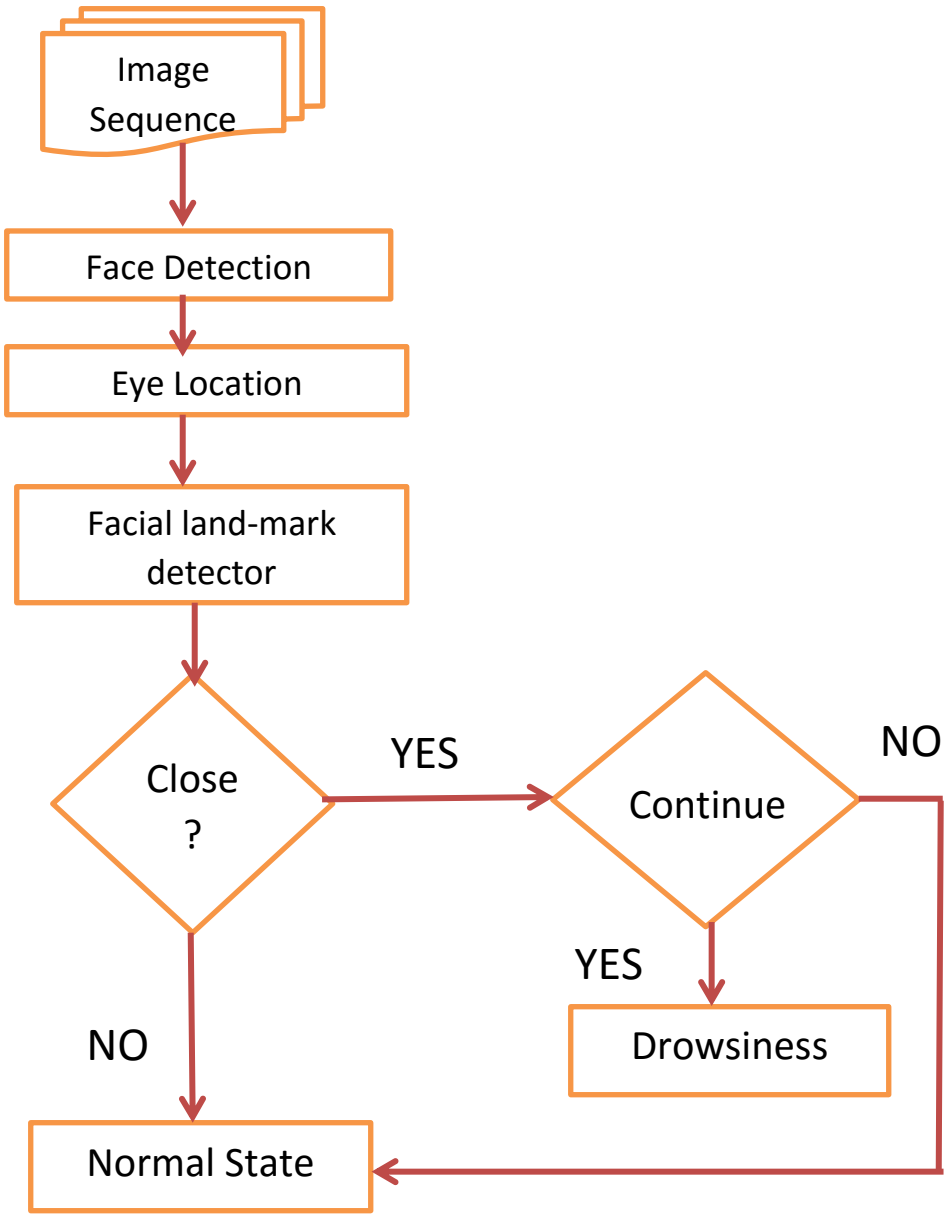
Example of alternative shape predictors. The model in the first quarter below , detects the eyes and eyebrows. The model on its left, localize the entire set of landmarks with a faster execution speed and a reduced size (compared to SP68). The last two models detect respectively: the face contour, nose and mouth.

If at all we have created a dataset consisting of all the landmarks,we can extract the particular parts of the face by slicing the xml file.

Fig 4.5 Facial landmark detection

CHAPTER 5:

WORK FLOW



5.1 Step by step procedure

- **Load face detector**: All facial landmark detection algorithms take as input a cropped facial image. Therefore, our first step is to detect all faces in the image, and pass those face rectangles to the landmark detector. We load OpenCV's and trained face detector
- **Create Facemark Instance** : Next we create an instance of the Facemark class. It is wrapped inside OpenCV smart pointer (PTR) so you do not have to worry about memory management.
- **Load landmark detector**: Next, we load the landmark detector. This landmark detector was trained on a few thousand images of facial images and corresponding landmarks. Public datasets of facial images with annotated landmarks are done during training process.
- **Capture frames from webcam**: The next step is to grab a video frame and process it. sets up the video capture from the webcam connected to your machine. We then continuously grab frames from the video until q button is pressed.
- **Detect faces**: We run the face detector on every frame of the video. The output of a face detector is a vector of rectangles that contain one or more faces in the image.
- **Run facial landmark detector**: We pass the original image and the detected face rectangles to the facial landmark detector. For every face, we get 68 landmarks which are stored in a vector of points using numpy libraries. Because there can be multiple faces in a frame, we have to pass a vector of vector of points to store the landmarks
- **Draw landmarks**: Once we have obtained the landmarks, we can draw a closed line over the 68 facial landamark by using opencv's convex hull method on each frame for displaying purpose.
- **Eye detection:Since we use** facial landmark prediction for eye detection Facial landmarks are used to localize and represent salient regions of the face, such as: Eyes , Eyebrows ,Nose ,Mouth ,Jawline in which we focus only on eye.
- **Eye State Determination:** after detecting eye we calculate eye aspect ration using Euclidean formula for further process**.**

- **Drowsiness Detection:** finally by EAR(eye aspect ratio) we compare it with standard threshold eye aspect ratio for drowsiness against standard consecutive frames. To detect drowsiness.

5.2 Face detection

The landmarks that we are interested in, are the one that describes the shape of the face attributes like: *eyes*, eyebrows, nose, mouth, and chin. These points gave a great insight about the analyzed face structure, that can be very useful for a wide range of applications, but we specifically use this for face detection

we used the more accurate HOG + Linear SVM face detector for the laptop/desktop implementation, but required a less accurate but faster Haar cascade to achieve real-time .

In general, you'll find the following guidelines to be a good starting point when choosing a face detection model:

Haar cascades: Fast, but less accurate. Can be a pain to tune parameters.

HOG + Linear SVM: Typically more accurate than Haar cascades with less false positives. Normally less parameters to tune at test time. Can be slow compared to Haar cascades.

Deep learning-based detectors: Significantly more accurate and robust than Haar cascades and HOG + Linear SVM when trained correctly. Can be very slow depending on depth and complexity of model. Can be sped up by performing inference on GPU (you can see an OpenCV deep learning face detector in this post).

Initially, the faces are detected using a Haar Cascade Classifier on an image in conjunction with the cropping of the cardinal section of the face.A geometric face model is formed with the detection of eyes performed using the Haar Cascade Classifier, while nose detection has been used as a reaffirmation mechanism along with the eyes.

Later, HOG (Histogram of Oriented Gradients) features are extracted from large numbers of facial images to be used as part of the recognition mechanism. These HOG features are then labeled together for a face/user and a Support Vector Machine (SVM) model is trained to predict faces that are fed into the system.

Possible use cases and environmental conditions are as follows:

- Access management using face recognition.
- Near-range scenarios involving distances of 5 to 6 feet between the camera and the face of the person to be recognized.
- Successful recognition is restricted to 15 degrees of freedom in pitch, yaw, and roll for the face.

5.2.1 Harr classifier method

In order to train a classifier to detect faces, two large sets of images are formed, with one set containing images with faces, and the other set without. These images are then used to generate classifier models. The classifier is generated by extracting Haar features from the positive and negative images. A single classifier is trained using each feature shown in the illustration below. However, a single classifier alone does not produce high accuracy, and so multiple such classifiers are cascaded. The final classifier formed is a weighted sum of weak classifiers. Using this method, the classifier provides classification accuracy of more than 95%.

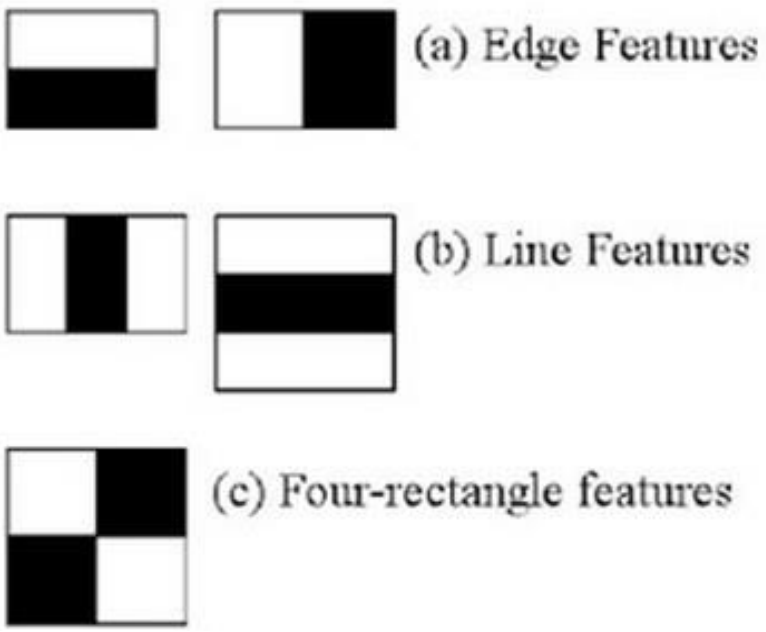


Fig 5.1 Haar Features

In the example below, images are used as convolutional kernels to extract features, where each feature is a value obtained by subtracting the summation of pixels under the black rectangle from the summation of pixels under the white rectangle.

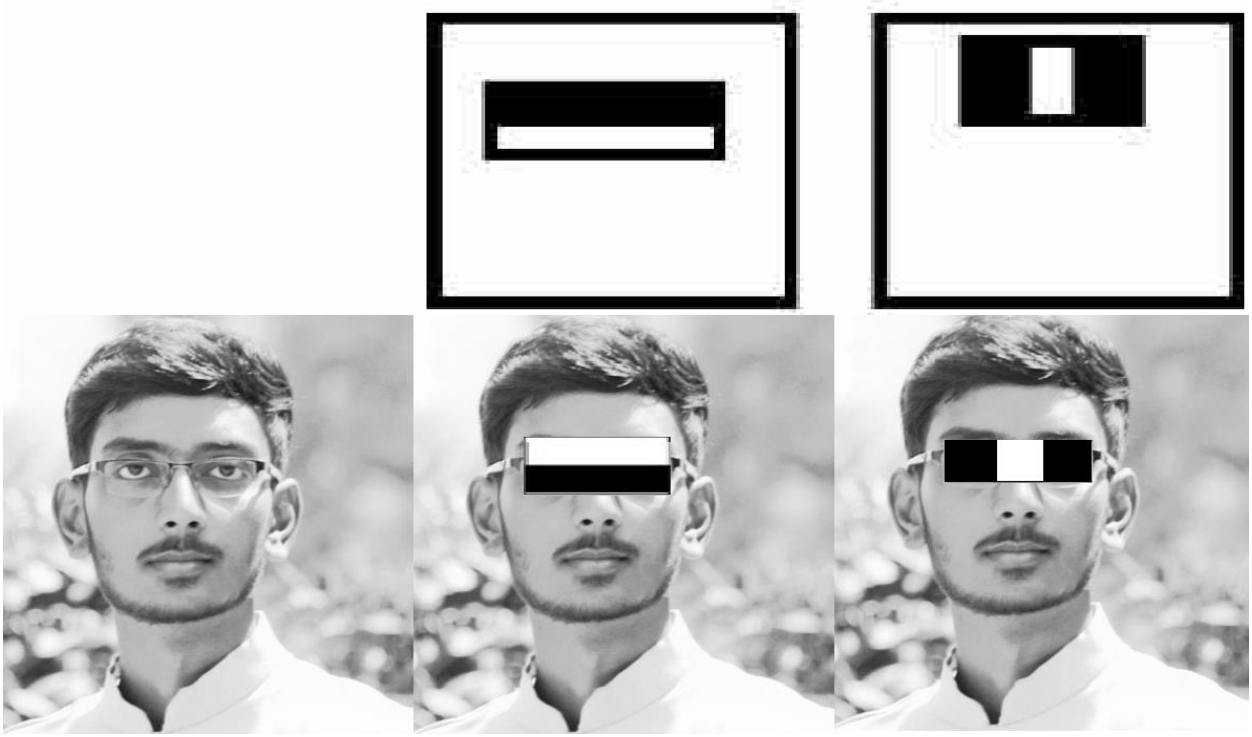


Fig 5.2 Example Haar Features

The image above provides examples of two features: edge and line. Edge features efficaciously map the face attribute; i.e., the eye region is darker than any other part of the face. The line feature maps the nose attribute; i.e., the vertical nose line on the face is lighter than the sides. Since, individually, any of these features cannot classify the pattern accurately, they are used in a cascade; hence the name of Haar Feature-based Cascade Classifiers.

The detection of the face is achieved using Haar Feature-based Cascade Classifiers, as discussed in the previous section. Typically, the accuracy of face recognition is highly dependent on the quality and variety of the sample images. The variety of sample images can be obtained by capturing multiple images with multiple facial expressions for the same face.

Fig 5.3 Sample face capture

Once the face is detected, it can be cropped and stored as a sample image for analysis

5.2.2 Geometric face model

To form a geometric face model, a pair of eyes are typically considered as the first feature to be located within the image. Ideally, any of the features can be used as a starting point to form a face model, but starting with the location of eyes produces a face model with higher accuracy. In some cases, the location of the nose is used to determine the face model. However, the eyes are typically considered as a primary starting feature, while the nose is considered as a secondary starting feature for situations when the eyes are not located or are partially occluded.

- **Face Model Using the Eyes:** From the coordinates for the center of both the eyes, the necessary section (features) of the face is obtained using the equations shown below:

$$h_{face} = K_f\, d_{eye} \quad \text{(eq. 1)}$$

$$h_{eye} = K_e\, h_{face} \quad \text{(eq. 2)}$$

$$w_{eye} = K_{we}\, h_{face} \quad \text{(eq. 3)}$$

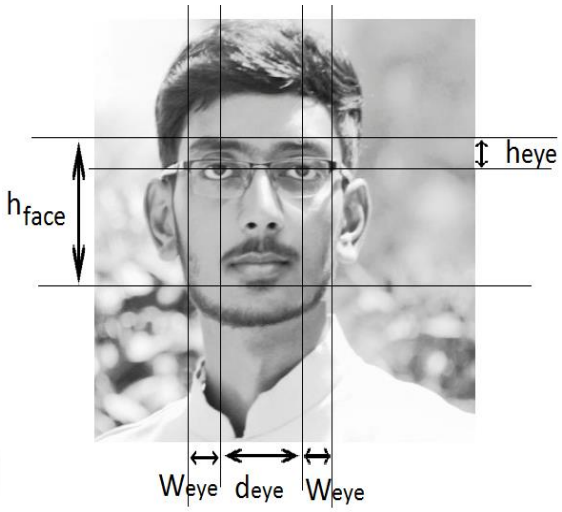


Fig 5.4 Geometric face model using the eyes

- **Face Model Using the Nose:** Using the coordinates for the center of the nose, the coordinates for the center of both the eyes are obtained using the equations shown below. Further, cropping of a necessary section of the face is obtained using equations (eq.) 1, 2, and 3 (mentioned above).

$$d_{nose\text{-}eye} = Nx - K_{wne}Nx \quad (eq.\ 4)$$

$$h_{nose\text{-}eye} = Ny/K_{hne} \quad (eq.\ 5)$$

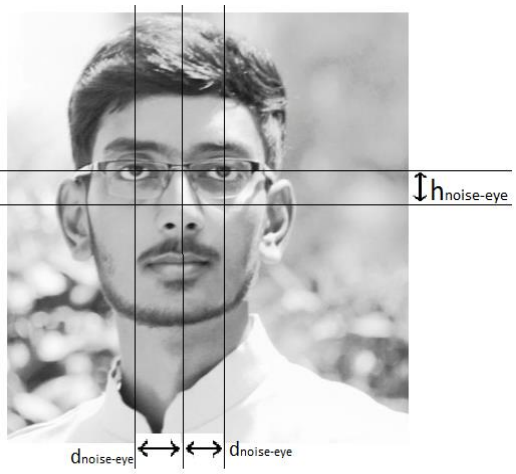


Fig 5.5 Geometric face model using the nose

To improve the recognition accuracy, faces cropped with dimensions less than 256 x 256 are discarded during the face capture process. Additionally, face regions deviate significantly with respect to the direction of the source light. To mitigate this, histogram equalization is performed on the cropped face image. This reduces asymmetries formed within the face due to uneven lighting.

5.2.3 Face Train

In this stage, features from images associated with each person are gathered. Later, a complete set of information from all of the stored images, isolated per person as a single SVM label, is trained to generate an SVM model.

Support vector machines (SVMs) are supervised machine learning models that divide and classify data. SVMs are widely used for applications such as face detection, classification of images, handwriting recognition, etc. An SVM model can be considered as a point space wherein multiple classes are isolated using hyperplanes.

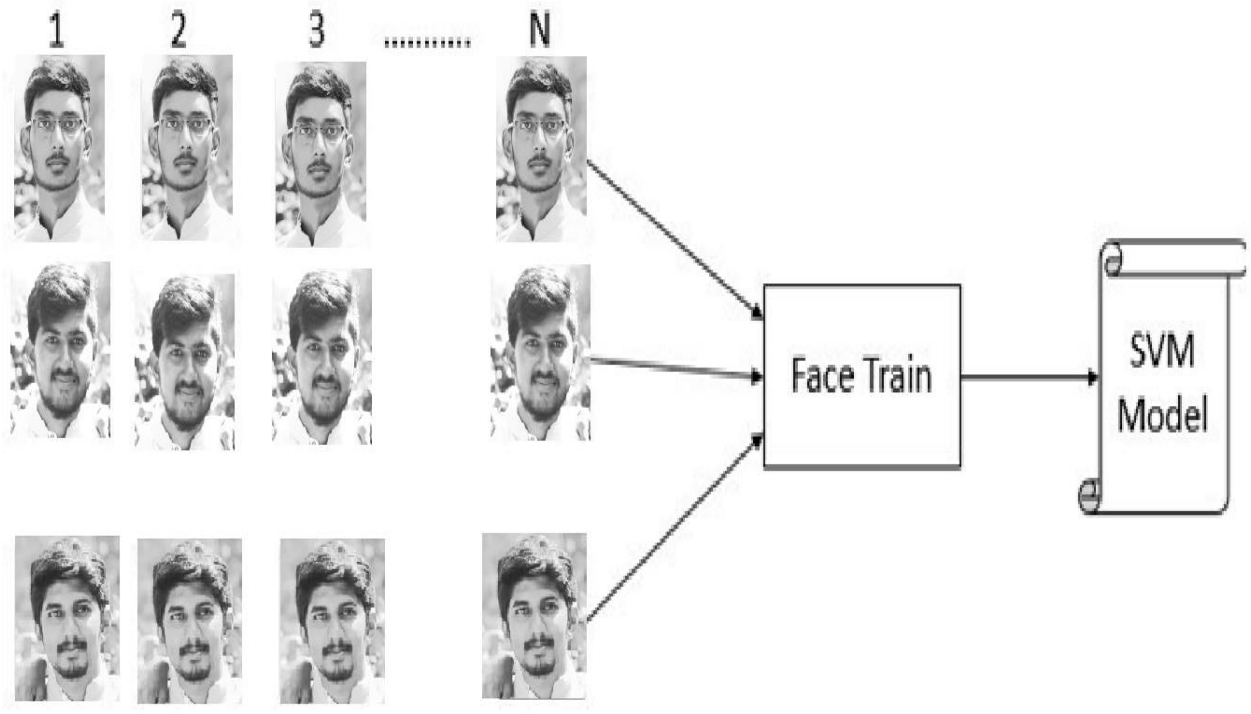


Fig 5.6 Block diagram of face training

5.2.1 Histogram of Oriented Gradients (HOG)

Calculate Histogram of Oriented Gradients

Step 1 : Preprocessing

HOG feature descriptor used for pedestrian detection is calculated on a 64×128 patch of an image. We have selected a patch of size 100×200 for calculating our HOG feature descriptor. This patch is cropped out of an image and resized to 64×128. Now we are ready to calculate the HOG descriptor for this image patch.

Fig 5.7 Resized image

Step 2 : Calculate the Gradient Images

To calculate a HOG descriptor, we need to first calculate the horizontal and vertical gradients; after all, we want to calculate the histogram of gradients. This is easily achieved by filtering the image with the following kernels.

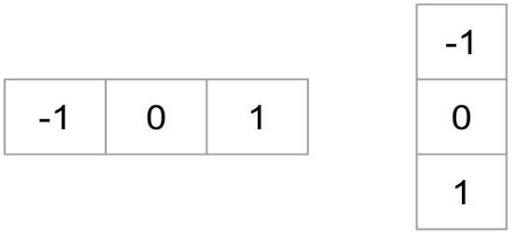


Fig 5.8 Horizontal and Vertical kernels

We can also achieve the same results, by using **Sobel** operator in OpenCV with kernel size 1.

Step 3 : Calculate Histogram of Gradients in 8×8 cells

In this step, the image is divided into 8×8 cells and a histogram of gradients is calculated for each 8×8 cells.Not only is the representation more compact, calculating a histogram over a patch makes this represenation more robust to noise. Individual graidents may have noise, but a histogram over 8×8 patch makes the representation much less sensitive to noise.

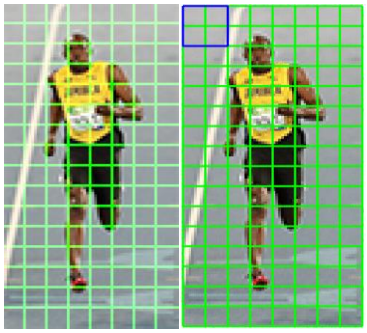


Fig 5.9 8×8 cells of HOG. And 16x16 cells

Step 4 : 16×16 Block Normalization

In the previous step, we created a histogram based on the gradient of the image. Gradients of an image are sensitive to overall lighting. If you make the image darker by dividing all pixel values by 2, the gradient magnitude will change by half, and therefore the histogram values will change by half. Ideally, we want our descriptor to be independent of lighting variations. In other words, we would like to "normalize" the histogram so they are not affected by lighting variations.

Step 5 : Calculate the HOG feature vector

To calculate the final feature vector for the entire image patch, the 36×1 vectors are concatenated into one giant vector. What is the size of this vector ? Let us calculate

1. How many positions of the 16×16 blocks do we have ? There are 7 horizontal and 15 vertical positions making a total of 7 x 15 = 105 positions.
2. Each 16×16 block is represented by a 36×1 vector. So when we concatenate them all into one gaint vector we obtain a 36×105 = **3780**dimensional vector.

Visualizing Histogram of Oriented Gradients

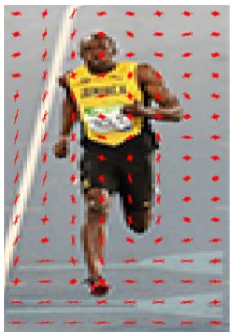


Fig 5.10 Visualizing Histogram of Oriented Gradients

The HOG descriptor of an image patch is usually visualized by plotting the 9×1 normalized histograms in the 8×8 cells. See image on the side. You will notice that dominant direction of the histogram captures the shape of the person, especially around the torso and legs.

A HOG is a feature descriptor generally used for object detection. HOGs are widely known for their use in pedestrian detection. A HOG relies on the property of objects within an image to possess the distribution of intensity gradients or edge directions. Gradients are calculated within an image per block. A block is considered as a pixel grid in which gradients are constituted from the magnitude and direction of change in the intensities of the pixel within the block.

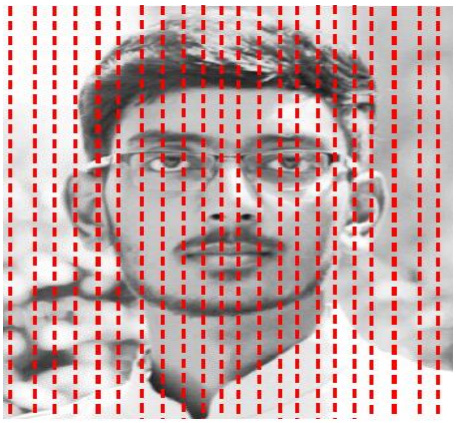


Fig 5.11 HOG features sample face

In the current example, all the face sample images of a person are fed to the feature descriptor extraction algorithm; i.e., a HOG. The descriptors are gradient vectors generated per pixel of the image. The gradient for each pixel consists of magnitude and direction, calculated using the following formulae:

$$g = \sqrt{g_x^2 + g_y^2}$$

$$\theta = \arctan \frac{g_y}{g_x}$$

In the current example, Gx and Gy are respectively the horizontal and vertical components of the change in the pixel intensity. A window size of 128 x 144 is used for face images since it matches the general aspect ratio of human faces. The descriptors are calculated over blocks of pixels with 8 x 8 dimensions. These descriptor values for each pixel over 8 x 8 block are quantized into 9 bins, where each bin represents a directional angle of gradient and value in that bin, which is the summation of the magnitudes of all pixels with the same angle. Further, the histogram is then normalized over a 16 x 16 block size, which means four blocks of

8 x 8 are normalized together to minimize light conditions. This mechanism mitigates the accuracy drop due to a change in light. The SVM model is trained using a number of HOG vectors for multiple faces. Hence after traning we get localized 68 point using which can detect face in any condition. This is because of wide verities of training data applied to process.

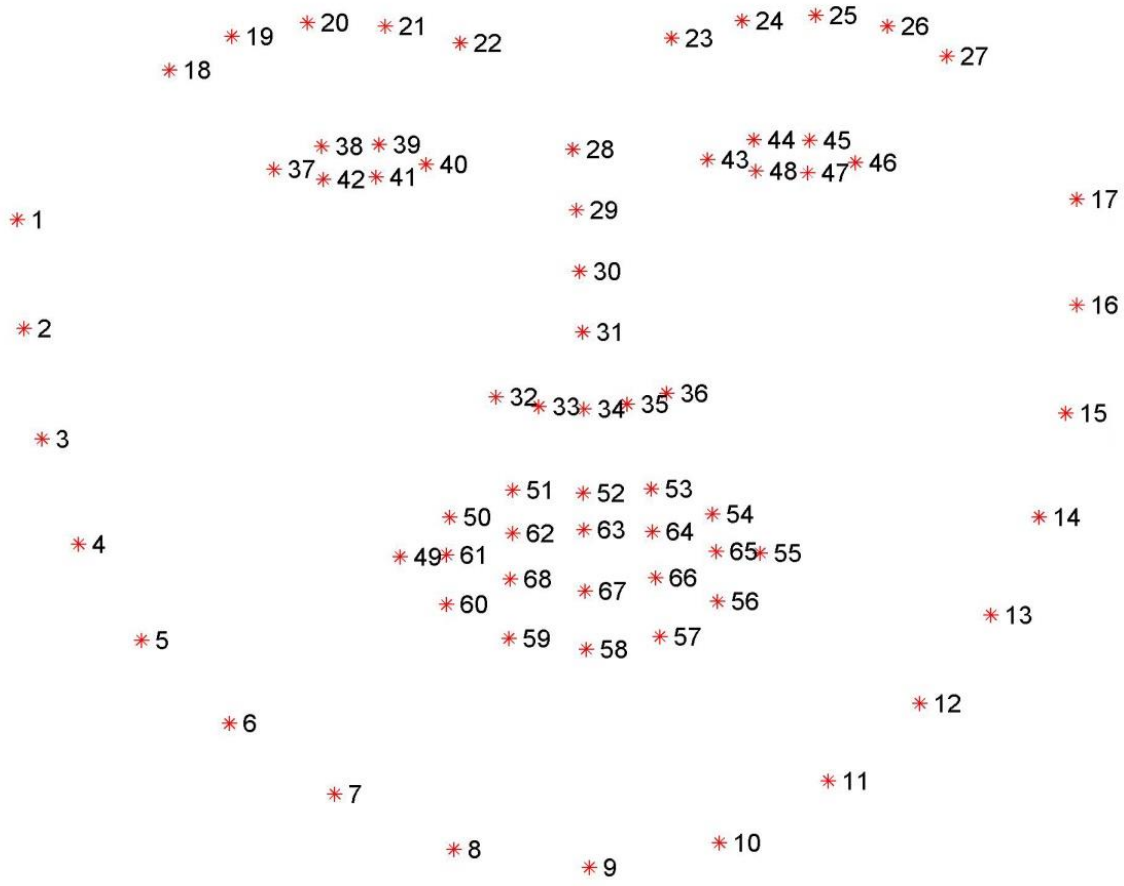


Fig 5.12 The full set of facial landmarks that can be detected via dlib

5.3 Eye Detection and Drowsiness Detection

From the face, the eyes are detected for further processing to detect whether the driver is in drowsiness or not. In this, we are specially characterizing the eyes from the face. A real time algorithm to detect eye blinks in a video sequence from a camera is used in this proposed system. Recent landmarks detectors exhibit excellent robustness against a head orientation with respect to a camera, varying illumination and facial expressions. In this project, the landmarks are detected precisely enough to estimate the level of the eye opening. The proposed algorithm

therefore estimates the landmark positions, extracts a quantity which is known as the eye aspect ratio (EAR) for characterizing the eye opening in each frame.

5.3.1 Eye detection

Out of 68 facial landmark there are 12 landmark related to eye, Each eye is represented by 6 (x, y)-coordinates, starting at the left-corner of the eye (as if you were looking at the person), and then working clockwise around the remainder of the region:

Left eye [37 to 42]

Right eye [43 to 48]

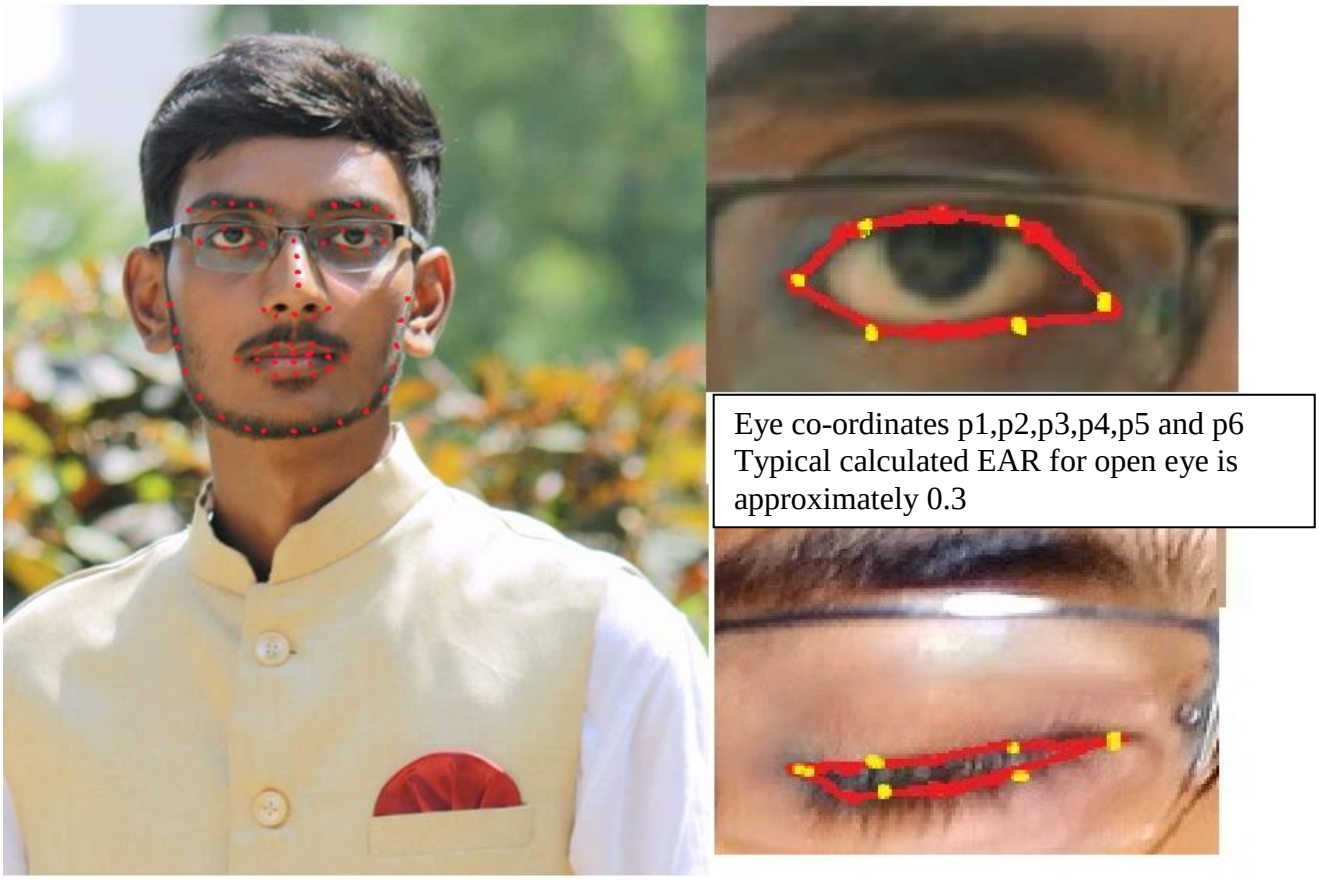


Fig 5.13 facial landmark alignment on face

Eye Aspect Ratio (EAR) technique

Each eye is represented by 6 (x, y)-coordinates in landmarks retuned Dlib predictor function , starting at the left-corner of the eye (as if you were looking at the person), and then working clockwise around the remainder of the region. There is a relation between the width and the height of these coordinates. Author then derive an equation that reflects this relation called the eye aspect ratio (EAR):

In this technique, we are using different landmarks to detect the opening and closing of eye. This landmark detectors that capture most of the characteristic points on a human face image. The eye blink is a fast closing and reopening of a human eye. Each individual person has a little bit different pattern of blinks. The pattern differs in the speed of closing and opening of the eye, a degree of squeezing the eye and in a blink duration. The eye blink lasts approximately 100-400ms. From the landmarks detected in the image, we derive the eye aspect ratio (EAR) that is used as an estimate of the eye opening state. For every video frame, the eye landmarks are detected. The eye aspect ratio between height and width of the eye is computed. From the fig. 2 P1,P2,…,P6 are the landmarks on the eye. where P1,…..,P6 are the 2D landmark locations on the eye.

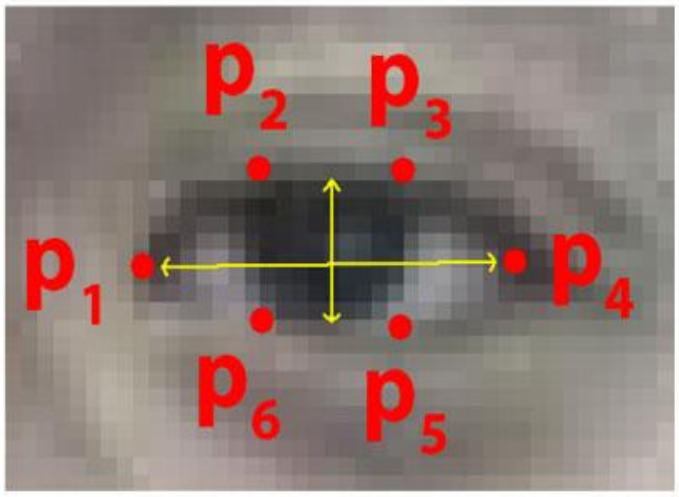


Fig 5.14: The 6 facial landmarks associated with the eye.

Based on this image, we should take away on key point:

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

Figure 5.15 The eye aspect ratio equation.

Where p1,.…, p6 are 2D facial landmark locations.

There is only one set of horizontal points but two sets of vertical points. the eye aspect ratio is approximately The numerator of this equation computes the distance between the vertical eye landmarks while the denominator computes the distance between horizontal eye landmarks, weighting the denominator appropriately since constant while the eye is open, but will rapidly fall to zero when a blink is taking place. When the person blinks the eye aspect ratio decreases dramatically, approaching zero. As shown in Figure-5.16, eye aspect ratio is constant, then rapidly drops close to zero, then increases again, indicating a single blink has taken place. This equation so interesting because the eye aspect ratio is approximately constant while the eye is open, but will rapidly fall to zero when a blink is taking place.

Using this simple equation, we can avoid image processing techniques and simply rely on the ratio of eye landmark distances to determine if a person is blinking.

To make this more clear, consider the following figure

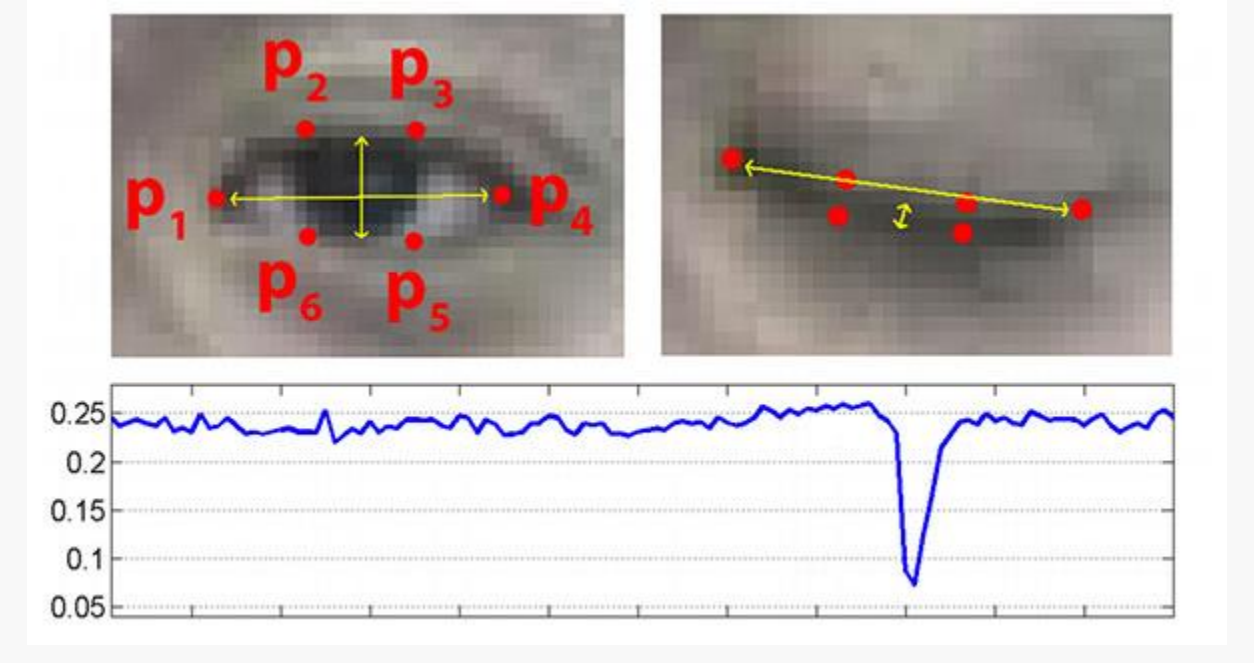


Fig 5.16: ***Top-left:*** **A visualization of eye landmarks when then the eye is open.** ***Top-right:*** **Eye landmarks when the eye is closed.** ***Bottom:*** **Plotting the eye aspect ratio over time. The dip in the eye aspect ratio indicates a blink**

On the top-left we have an eye that is fully open — the eye aspect ratio here would be large(r) and relatively constant over time.

However, once the person blinks (top-right) the eye aspect ratio decreases dramatically, approaching zero.

The bottom figure plots a graph of the eye aspect ratio over time for a video clip. As we can see, the eye aspect ratio is constant, then rapidly drops close to zero, then increases again, indicating a single blink has taken place.

Python Function for calculating EAR

```
def eye_aspect_ratio(eye):
        # compute the euclidean distances between the   vertical eye landmarks
        A = dist.euclidean(eye[1], eye[5])
        B = dist.euclidean(eye[2], eye[4])
        # compute the euclidean distance between the horizontal eye landmark
        C = dist.euclidean(eye[0], eye[3])
        # compute the eye aspect ratio
          ear = (A + B) / (2.0 * C)
        # return the eye aspect ratio
        return ear
```

Algorithm for detection of Blinks

```
if ear < EYE_AR_THRESH:
                    COUNTER += 1
            # otherwise, the eye aspect ratio is not below the blink threshold


else:
                    # if the eyes were closed for a sufficient number of
                    # then increment the total number of blinks
                    if COUNTER >= EYE_AR_CONSEC_FRAMES:
                        TOTAL += 1
                    # reset the eye frame counter
                    COUNTER = 0
```

5.3.2 Eye State Determination:

Finally, the decision for the eye state is made based on EAR calculated in the previous step. If the distance is zero or is close to zero, the eye state is classified as “closed” otherwise the eye state is identified as “open”.

Drowsiness Detection:

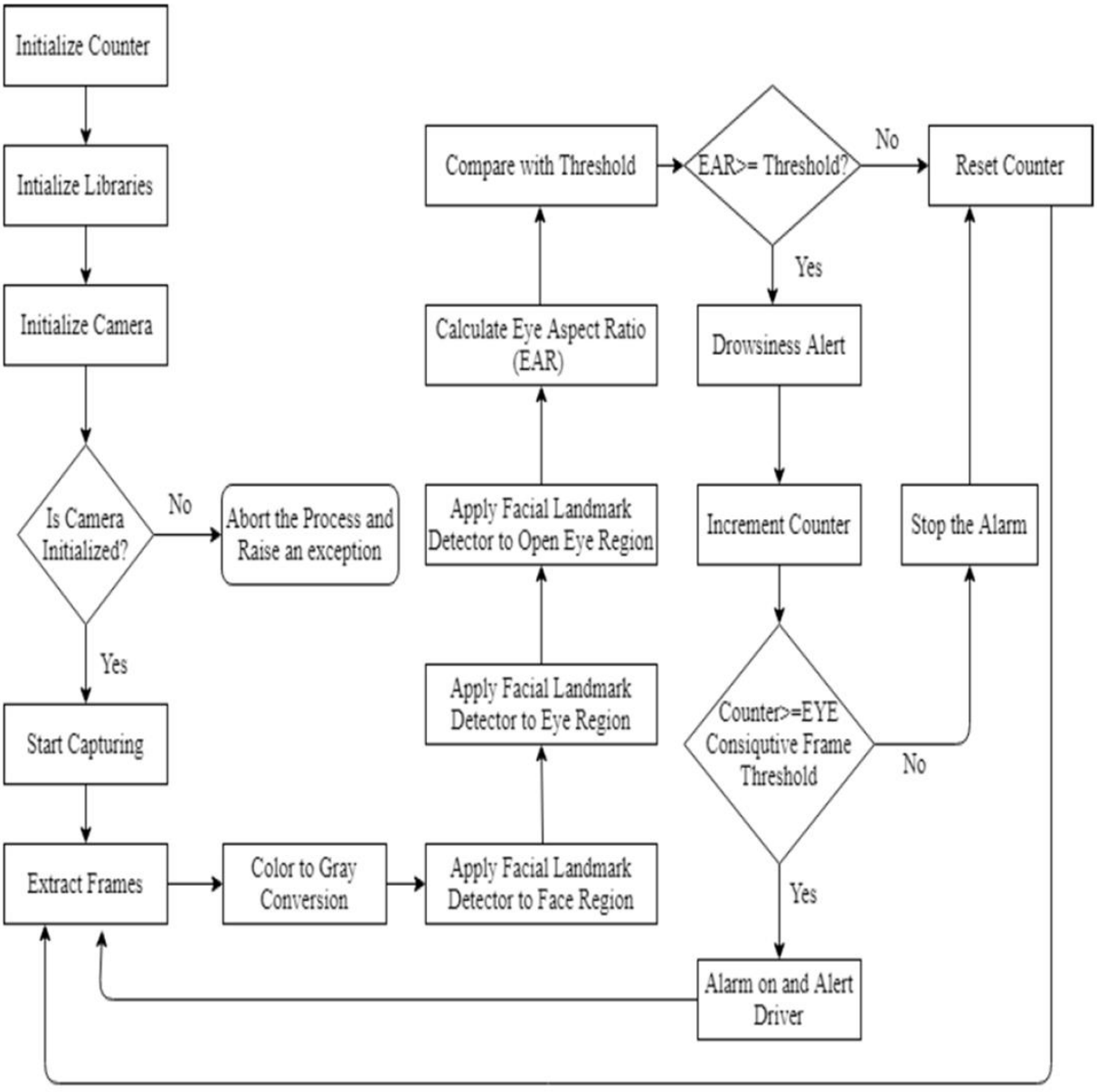
The last step of the algorithm is to determine the person’s condition based on a pre-set condition for drowsiness. The average blink duration of a person is 100-400 milliseconds (i.e. 0.1-0.4 of a second). Hence if a person is drowsy his eye closure must be beyond this interval. We set a time frame of 5 seconds. If the eyes remain closed for five or more seconds, drowsiness is detected and alert pop regarding this is triggered.

The EAR is mostly constant when an eye is open and is getting close to zero while closing an eye. Since eye blinking is performed by both eyes synchronously, the EAR of both eyes are taken and it is averaged. After getting the EAR value, if the value is less than the limit for 2 or 3 seconds the driver is said to be drowsy. The buzzer connected to the system performs actions to correct the driver abnormal behavior.

Fig 5.17 drowsiness detection

5.4 RESULTS

We measure the EAR with respect to time . Each dip in EAR curve corresponds to eye blink and width of EAR dip represents duration of eye closed andformed basis of Microsleep detection. Algorithm performs well in repeated experiments on different type of faces. EAR threshold adjustment should be useful for fine tuning of algorithm on different type of faces. No false detection and missed detection of blinks and microsleep was observed.



RESULTS

Implementation of drowsiness detection and eye blink detection with Python, OpenCV, dlib was done which includes the following steps: Successful runtime capturing of video with camera in the video stream if aspect ratio is satisfied for specified number of frames, then only the drowsy or the blink was detected. Here we have captuted some of the images pertaining to drowsiness detection and blink detection. Here for drowsiness detection we have considered two factors

1) Drowsiness alert due to eye

2) Drowsiness alert due to mouth

Fig 6.1 Drowsiness alert due to eye

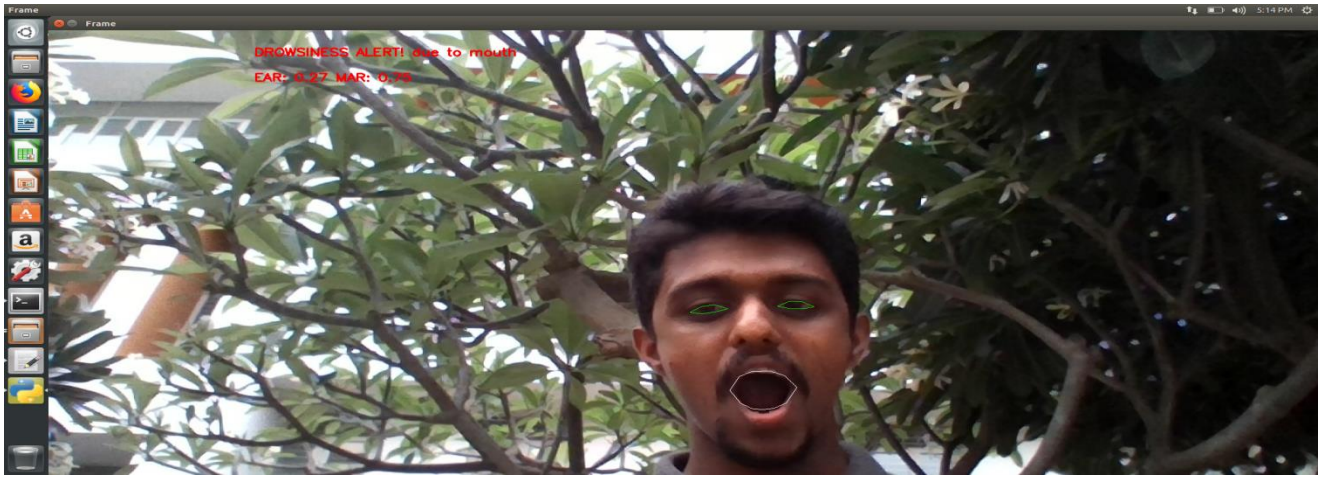


Fig 6.2 Drowsiness alert due to mouth

The above images depicts the drowsiness detected.The drowsiness detected are based on the threshold for aspect ratio and the number of frames taken into consideration.EAR and MAR are the eye and mouth aspect ratios respectively which are continouslly being calculated .

In case of blink detection ,we have two cases namely

1) Blink detection for 1 eye

2) Blink detection for 2 eyes

Working principle is same as that of the drowsiness detection but the only difference being number frames considered for detection.

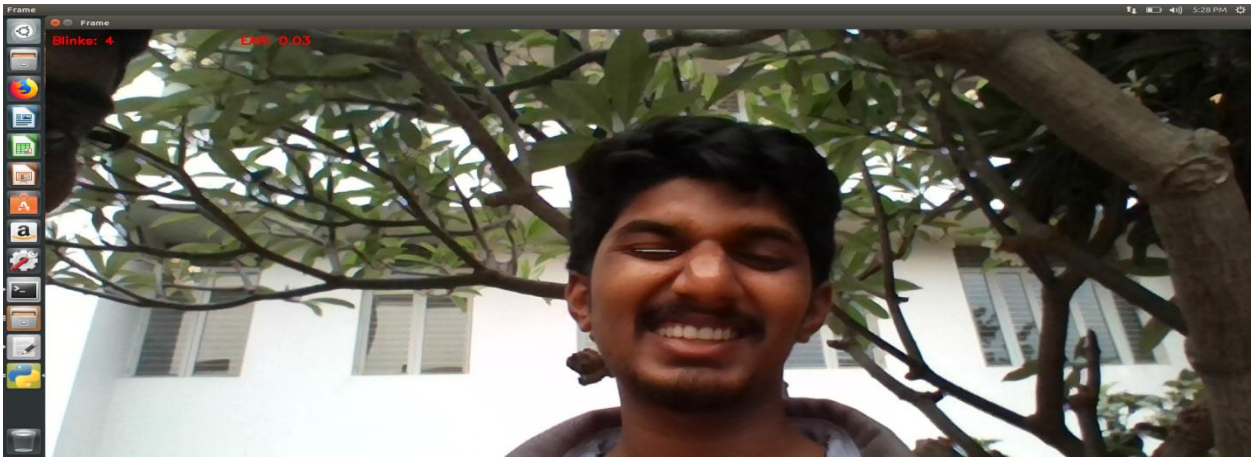


Fig 6.3 blink detection for 1 eye

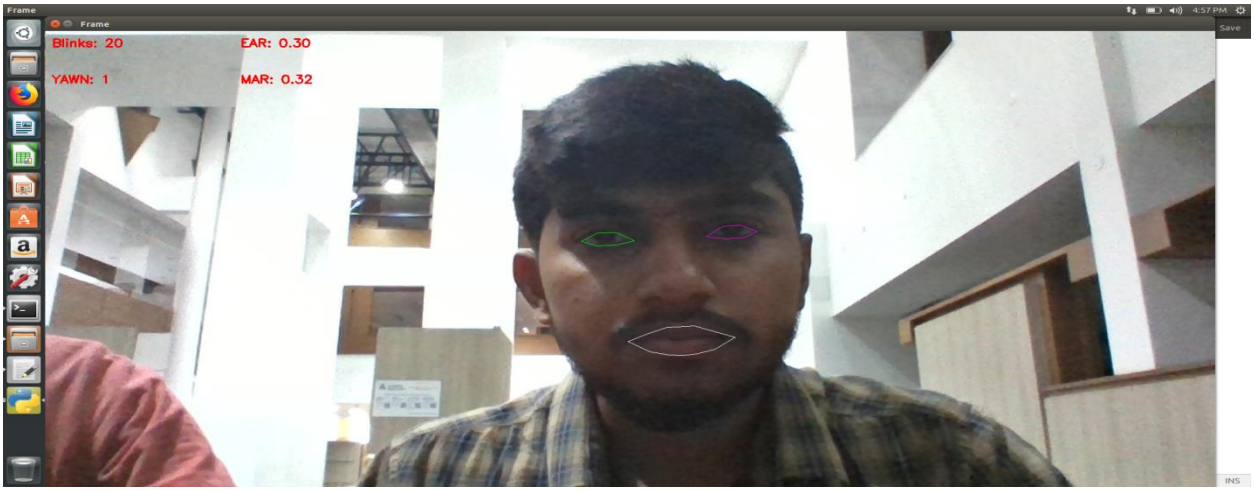


Fig 6.4 Blink detection for 2 eyes

These blink detectors can be used to capture images in case of spy applications.

REFERENCES

[1] Real Time Eye Detection and Tracking Method for Driver Assistance System., Sayani Ghosh, Tanaya Nandy and Nilotpal Manna.

[2] An Application of Detection Function for the Eye Blinking Detection., Tomasz Pander, Tomasz Przybyła, and Robert Czaba´nski, Krakow, Poland, May 25-27, 2008.

[3] Image Capturing using Blink Detection Keval Lakhani, Aunsh Chaudhari, Kena Kothari, Harish Narula., Keval Lakhani et al, / (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 6 (6) , 2015, 4965-4968.

[4] Robust Eye Blink Detection Based on Eye Landmarks.,Sarmad Al-gawwam and Mohammed Benaissa.Department of Electronic and Electrical Engineering, The University of Sheffield, Sheffield S1 3JD, UK.

[5] Eye detection in a face image using linear and nonlinear filters Saad A. Sirohey, Azriel Rosenfeld., Computer Vision Laboratory, Center for Automation Research, University of Maryland, College Park, MD 20742-3275, USA.

[6] Eye Detection Using Morphological and Color Image Processing .,Tanmay Rajpathaka, Ratnesh Kumarb and Eric Schwartzb.Machine Intelligence Laboratory, Department of Electrical and Computer Engineering, University of Florida, Gainesville, Florida.

[7] Eye-blinks in choice response tasks uncover hidden aspects of information processing., Edmund Wascher1, Holger Heppner1, Tina Möckel, Sven Oliver Kobald, Stephan Getzmann. Leibniz Research Centre For Working Environment and Human Factors, Ardeystr. 67, 44139 Dortmund, Germany.