Real-time Driver Drowsiness and Distraction Monitoring System

BACHELOR OF TECHNOLOGY IN BioEngineering

By

Dhruv Gupta (B21008)

Table of Contents

Abstract

The National Crime Records Bureau (NCRB) 2022 report states there were 464,674 roads-related traffic collisions in 2021, which caused 150,000+ traffic-related deaths in India.[1] According to the latest statistics (2023), the top cause (~50%) for accidents is reckless driving and the second leading cause (~25%) is that the drivers are getting distracted in some way (drowsy, talking on the phone, not attentive etc.). In addition to vehicle control, drivers regularly perform secondary tasks that hinder safe driving.

This project firstly briefly discusses different types of distractions and measures used for driver distraction detection. Then, it presents a technique for automatic detection and evaluation of driver distraction and drowsiness, based on visual information from a live video stream. This technique aims to locate, track, and analyze both the driver's face and eyes to detect distraction to prevent accidents.

In this project, a combination of algorithms was developed using OpenCV image processing methods and Facial Landmark Detection, which were refined to detect drivers' activity while driving in the real-time video stream.

Keywords – Driver behaviour, Driver distraction, Drowsiness detection, OpenCV,

Facial Landmark detection

Chapter 1: Introduction

1.1 Introduction

In one of the published reports on the leading causes of deaths in the world, the *World Health Organisation* cites that road accidents are one of the top 10 causes of early deaths[2]. One in five such accidents is caused due to distracted driver behaviour. These behaviours include but are not limited to sleeping or drowsing, talking on smartphone, texting on smartphone, operating the radio, drinking a beverage, reaching behind, talking to co-passenger and working on hair and makeup.

Three types of driver distractions are identified by *NHTSA (National Highway*

Traffic Safety Administration)[3]:

1. **Visual Tasks**, such as something as simple as checking a navigation system, which causes a driver to divert his or her attention from the road.
2. **Manual Task**, which requires a driver to remove one or both hands from the steering wheel, such as reaching for a drink or cell phone, operating radio, etc.
3. **Cognitive tasks**, causing a driver's mind and focus to wander to something besides the task of driving, such as sleeping, drinking a beverage, talking to a co-passenger etc.

Considering the causes mentioned above, the driver's drowsiness and distraction detection seem like a reasonable strategy in avoiding road accidents. The three main ways of real-time drowsiness detection in the research domain are the following:

1. **Vehicle-based measures** - These strategies attempt to identify distraction and drowsiness from vehicle circumstances and monitor the variations of steering wheel angle, acceleration, lateral position, etc. However, these approaches detect driver drowsiness slowly and are not efficient in real-time detection. [4,5]
2. **Physiological measures** - Physiological or Signal-based methods infer drowsiness from a combination of psychological and physiological parameters and are one of the most accurate methods. Some essential methods that can be used for investigation are *Electro-Encephalogram (EEG)*, activities of the autonomic nervous system from *Electro-Cardiogram (ECG)*, etc. But this approach needs to consider additional equipment attached to the driver, which can affect driving negatively and are also expensive and lacks practicality. [4,6]
3. **Behavioural measures** - In this strategy, the behaviour of the driver, including yawning, eye closure, eye blinking, head pose, etc., are monitored through a camera, and the driver is alerted if any of these drowsiness symptoms are detected. Facial feature-based strategies can evaluate the target in real-time without much equipment. Also, they are reasonable and more accessible than other strategies.[4]

These measures have been studied, and the advantages and disadvantages of each have been discussed in various research papers. However, to develop an efficient distraction detection system, we found that behavioural measures seem to be more effective and economical. One

such strategy based on Behavioral measures would be to develop a video/image processing based algorithm to identify drivers engaging in distracting activities by feeding it real-time camera video stream. This algorithm can be then set up in an embedded computer, such as RaspberryPi with an attached camera, to classify the driver's behaviour by checking if they are driving mindfully, wearing the seatbelt, and alerting them in case of a potential hazard. Thus, by continuously monitoring the driver one can detect the distracted state of the driver and a timely warning is issued.

1.2 Literature Review

Driver's distraction and drowsiness detection is an active area of research. Various government organizations, as well as private companies, have been working to solve the problem. There are commonly three types of measures, as briefly discussed in the previous section. Several techniques are proposed so far in each type of measure for monitoring the distraction of drivers. In this regard, the research is intended towards an effective, efficient and economical way to tackle the problem.

Arun Sahayadhas et al. presented a hybrid drowsiness detection system that is based on non-intrusive physiological measures with other measures to determine the drowsiness level of a driver accurately. The paper also discussed the various ways to determine the distracted state of drivers and provided a comparative study among them. The technique presented provides high accuracy but is complex as it uses measures like ECG. [4]

Duy Tran et al. proposed an algorithm for realistic validation of distraction detection. It consisted of four deep neural networks – VGG-16, AlexNet, GoogleNet, and a residual network implemented on an embedded graphic processing unit platform. They collected a dataset of images of the drivers in both normal and distracted driving conditions. However, due to the heavy processing needed to process the Neural networks in general, it isn't efficient enough to implement for the real-time task.[7]

Soukupová and Čech in their 2016 paper, *Real-Time Eye Blink Detection Using Facial Landmarks*, used facial landmarks to find the position of eyes to detect blinking in real-time. They showed that the landmarks were detected precisely enough to reliably estimate the level of the eyes opening in an image. Their algorithm exhibits excellent robustness against any head orientation

concerning the position of the camera, varying illumination and facial expressions. They also detailed the types of systems used for detecting drowsiness of drivers. [8]

1.3 Motivation

With the increase in the number of taxi driving online apps, the number of cabs functioning in India has increased many folds in the last ten years. Also, a high volume of goods and passengers in India are transported via roads through trucks and buses respectively. Sleeping due to fatigue and tiredness are principal driving hazards in such cases.

With an increase in the pressure to make fast deliveries and quickly pick and drop passengers, drivers are stretching their limits and are driving even in extreme tiredness without paying much attention to how dangerous this can be. In many cases, due to extended driving hours, feeling tired, drowsy, sleepy or not being attentive can cause severe incidents on highways and city streets. The risk of accidents increases many folds in hilly areas, especially. Citing one example, on 8th of July, 2019, Delhi-bound bus met with an accident Off Yamuna Expressway after driver dozed off leading to loss of 29 lives.[15] This is just one example in the list of hundreds of such accidents which happen every year. Though accidents due to these reasons are avoidable, nothing much has been done in this area due to which accidents are rising by every passing year.

Many automobile and transportation companies are searching for a solution to these problems in order to ensure the safety of the driver, goods, passengers and other people on the road. Finding a working, efficient and cost-effective solution to detect driver drowsiness and distraction can play a very effective role in reducing the number of accidents by a big factor.

The concerned project can be a breakthrough if it is implemented with adequate hardware and proper warning systems. With the implementation of this project, not only do we foresee a sharp decline in the number of accidents on the road, but we also expect that it will enforce and encourage safe driving habits in society.

1.4 Final Output Overview

In the completed project, we were able to take in a live video stream of a person (driver) from the webcam of the laptop and put it through a program written in python language. The program successfully detected the face and eyes in the video, which was simultaneously

displayed to the user too. If the eyes were closed partially or totally for a preset amount of time, then a loud alarm was sounded by the laptop until the person opened his/her eyes which would automatically stop the alarm. Detailed execution and methods used in building the project are explained step by step in further chapters. In the following chapters, the building of the project and algorithms used are explained from scratch in detail so that even with limited prior knowledge of the subject, one can understand the majority of the theory with ease.

Chapter 2: Problem Statement

2.1 Problem Definition

The alarming statistics presented in the Introduction section indicates a necessity to test new innovative methods to tackle the problem of Driver distraction detection. One such method would be to develop an algorithm to detect drivers sleeping or drowsing by feeding the algorithm camera video.

The problem is stated as below:

Given a video stream of a dashboard camera, an algorithm needs to be developed to classify each driver's behaviour and determine if they are driving attentively or drowsing due to fatigue.

But first, we need to define the steps involved in reaching the end goal.

2.2 Execution

We can broadly divide the whole project into five smaller parts:

1. How do we detect a face in an image?
2. How to find and extract landmark points from a detected face?
3. How to detect eyes, nose, lips and jawline in real-time?
4. How to detect drowsiness from observing eye landmarks?
5. What can be done to improve the project?

Chapter 3: Face Detection

3.1 HOG with SVM implementation

In order to detect faces/humans in any given image, multiple methods and algorithms exist in the world of AI. In our implementation, we have used a particular method known as Histogram of Oriented Gradients which is heavily used in image processing and gives very accurate results. [9]

Dalal and Triggs in their seminal work, "*Histogram of Oriented Gradients for Human Detection",* demonstrated that a highly accurate human detector can be trained using the HOG image descriptor and a Linear Support Vector Machine (SVM).[10]

Following are the steps used to detect faces in an image using HOG:

1. We take *P positive* samples from our training data of the objects we want to detect (in our case faces) and extract HOG descriptors from these samples.
2. Then we take *N* negative samples from a *negative training set* that does not contain any face and extract HOG descriptors from these samples as well. In practice *N >> P.*
3. Then we train a linear *Support Vector Machine* on our positive and negative samples.
4. For each image in our negative training set, we apply the sliding window technique and slide your fixed window across the image. For each frame, we compute our HOG descriptors and apply our classifier. If the classifier (incorrectly) classifies a given window as an object, feature vector associated with the false positive is recorded along with the probability of the classification.
5. We take the false-positive samples found in the earlier stage, sort them by their confidence (i.e. probability) and re-train our SVM classifier using these samples.
6. Our trained classifier can be applied to our test dataset. After we finish scanning the image, we apply a suppression technique (non-maximum suppression implementation in python) to remove redundant and overlapping bounding boxes created in the last step

(Figure 3.1). The suppression technique removes the smaller boxes and keeps the largest box intact.

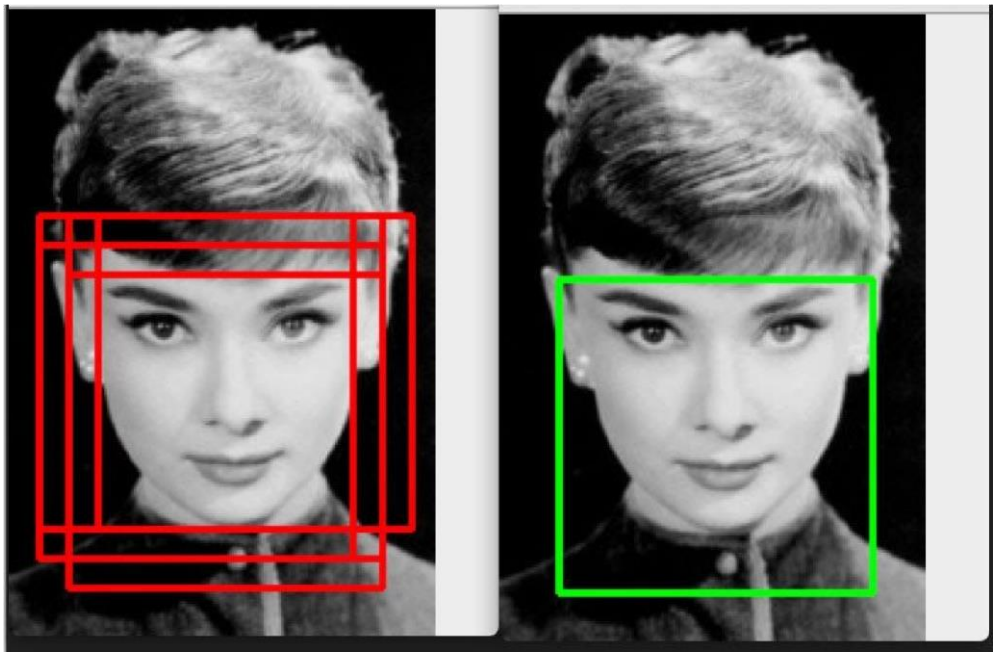

Figure 3.1: (Left) Multiple overlapping bounding boxes detected around the face. (Right) Applying NM suppression to remove the redundant boxes.[11]

Using the *Histogram of Oriented Gradients* technique combined with a *linear SVM classifier*, we have successfully detected a face in an image. Though in this chapter, we have explained the brief working of HOG with SVM to find faces, in our project, we have used a pre-trained python implementation of this algorithm rather than creating this classifier ourselves as the python library is faster and robust and can work with a wide variety of images pretty quickly.

Next question that is to be answered is - why do we need to find a face in an image?

The answer to that question is the foundation on which the next part is laid upon. One of the conditions of facial landmark detection is finding a face in an image, which we succeeded in doing in this part of the project.

Chapter 4: Facial Landmark Detection

4.1 Facial Landmarks

Facial landmarks are used to localise and identify key attributes of the face, including eyes, eyebrows, nose, mouth, jawline etc. Facial landmarks have been used successfully in many tasks such as face alignment, head pose estimation, blink detection and much more.

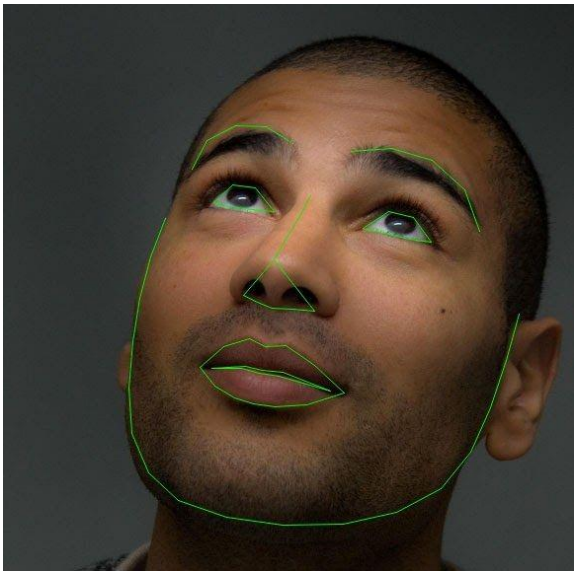

Figure 4.1: Facial landmarks are used to label and identify key facial attributes in an image[13]

Facial landmarks detection is a type of shape prediction problem in which, given an input image, a shape predictor attempts to detect important structural key points along the shape of the face.

It is usually considered as a two-step process:

- **Step 1:** Find the face in the image.
- **Step 2:** Detect the vital facial structures on the Region of Interest.

In the earlier chapter, we have explained already how we localise a face in a given image. From that step, we obtain the face bounding box and also the (x, y) coordinates of the face.

Next, given the face region, we then apply step 2 to detect key facial structures in the face region using an implementation of the 2014 paper by Kazemi and Sullivan.[12]

In brief, the implementation goes as follows:

1. In order to train an ML model, a training set consisting of labelled facial landmarks on an image is used. These images are manually labelled, consisting of specific (x, y)-coordinates of regions surrounding each facial structure.
2. The method also uses the probability (Priors) on the distance between pairs of input pixels.
3. Taking data both from 1 and 2 and using it as training data, an ensemble of regression trees (Machine learning model) are trained to estimate the facial landmark positions from the pixel intensities of the image.[12]

The result of this implementation is a facial landmark detector that can detect facial landmarks in real-time with high-accuracy.

We have used *Dlib python library* 's robust implementation of facial landmark detector in our project. Dlib's pre-trained detector is used to estimate the location of ***68 (x, y)-coordinates*** of crucial points on the face [Figure 4.2].

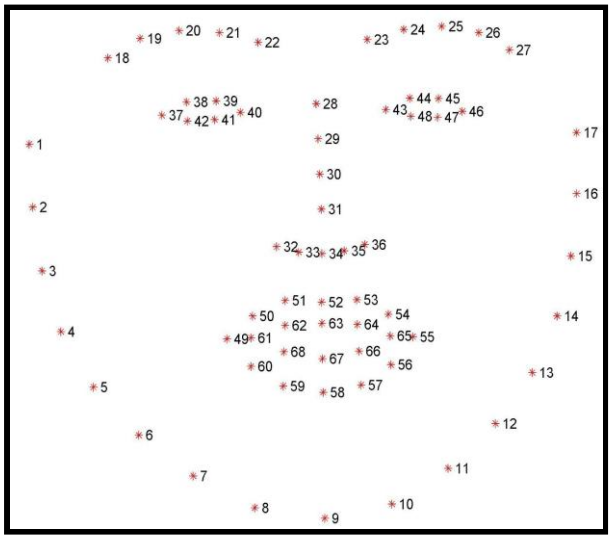

Figure 4.2. Visualizing the 68 facial landmark coordinates (htttps://arxiv.org/pdf/1805.05563.pdf)

Figure 4.3: Applying facial landmark detection using Dlib, OpenCV, and Python.

In Figure 4.3, we can see that the green box outlines the face and red dots point to specific facial features, such as jawline, mouth, nose, eyes, and eyebrows.

Finally, to summarise it, two different algorithms were used to get the image shown above. Histogram of Oriented gradients + Linear SVM was used for face detection (using OpenCV library in python) and Dlib's facial landmark detector to obtain the (x,y) coordinates.

In the next part, we answer the question - how to use facial landmarks to detect eyes and other features in real-time?

Chapter 5: Individual Feature Extraction

5.1 Key-feature extraction

Landmark points are like raw data with very high importance and informational value which can be used in a lot of scenarios to build useful and real-life applications such as drowsiness detection system. In this part, we use our detected facial landmarks to label and extract face regions and use them for our further application.

As discussed in the earlier chapter, the detector implemented inside Dlib produces 68 *(x, y)*-coordinates that map to specific facial structures. These 68 point mappings are all indexed and can be used individually to take note of a specific point or region. Looking back at Figure 4.2, we see that facial regions can be accessed easily in Python using indexing:

- *The jaw* - [0, 17].
- *The right eyebrow* - [17, 22].
- *The left eyebrow* - [22, 27].
- *The nose* - [27, 35].
- *The right eye* - [36, 42].
- *The left eye* - [42, 48].
- *The mouth* - [48, 68].

We created a dictionary of these indexed points so that we could extract the coordinates and indexes of various features from the landmark array with ease. We then visualized each of the regions by overlaying the segregation results on the input image.

We loop over each entry in the dictionary we created and extracted the indices of the current facial part to get the *(x, y)-coordinates* for each region. Then, we loop over the individual points of that single region to join them and draw a translucent hull.

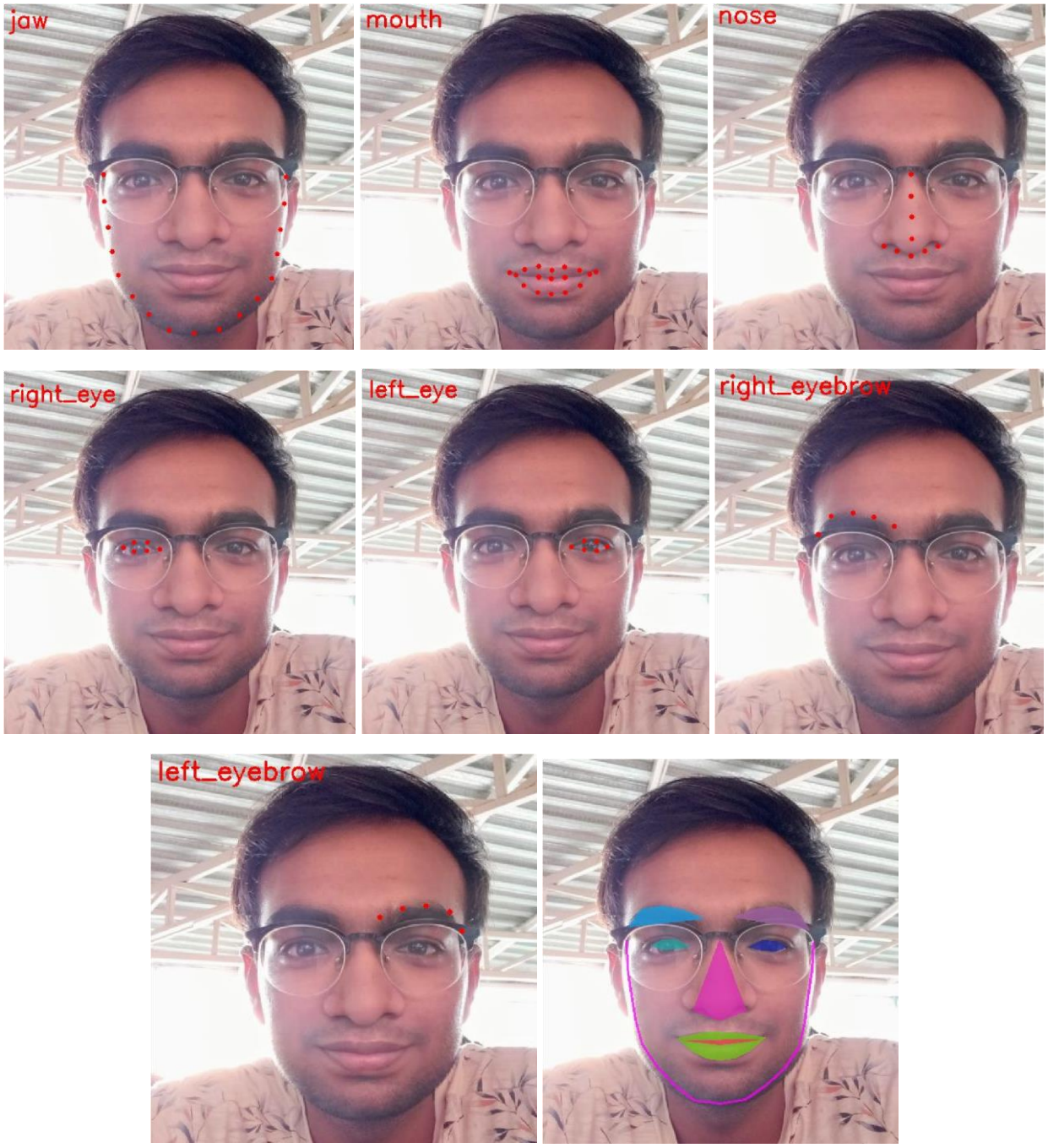

Figure 5.1 (1 -7) Automatically determining separate regions from landmarks.
(8)Individual facial regions overlayed with a translucent colour

Figure 5.1 shows the final output when we passed an image through the implemented algorithm. Hence, we saw how different face regions can be extracted using facial landmarks and how this knowledge can be applied in the further application.

5.2 Detection & extraction in real-time

In section 5.1, we have implemented various algorithms on still images. But we wish to implement the same algorithms in videos. With the help of OpenCV and Python, we were able to do that with very less effort.

We used a utility library *"Imutils"* in Python, which granted us access to any of our webcam/USB camera/Raspberry Pi camera module in a very efficient and fast manner. This library also allows us to conveniently implement the whole program on a RaspberryPi with just the change of a small parameter. The code loops over and gets each frame from our video stream, applies the algorithm discussed in the earlier section and then outputs it on the screen (if on laptop/PC).

With minimal changes in the underlying implementation, we could see that the entire process and working of the algorithm are the same. The main differences involved setting up our video with the help of *Imutils library* and then getting frames from the stream.

Now a new question arises - Why do we need to extract different regions of the face for our drowsiness detection?

Answer - We wish to detect if a person is sleeping or not, and for that, we need the eye region landmarks in real-time. In the next and final part, we use the techniques used in this chapter to create & complete our final drowsiness detection model.

Chapter 6: Algorithm to Detect Drowsiness

6.1 Methods to detect eye blink

The process of drowsiness detection is reasonably straightforward:

1. Setting up of camera and smooth video streaming.
2. Find a face in each frame and if that is found, apply facial landmark detection to extract both the eye regions.
3. Determine if the eyes are closed.[8]

4. If the eyes are closed for a certain threshold of time, sound the alarm until eyes are open again.

In general, there are two ways in which we can check if eyes are closed or not:

1. **Method 1** -Traditional way is to localise eyes, find the **whites of the eyes**, and then determine if that region disappears for a certain threshold of time.
2. **Method 2** - Find the **eye aspect ratio (EAR)** to check for closed eyes. This involves the calculation of the ratio of distances between the landmarks of the eyes.

The method of calculating EAR was discussed by Soukupová and Čech in their 2016 paper, *"Real-Time Eye Blink Detection Using Facial Landmarks"*.[8]

6.2 Using Eye Aspect Ratio Method

In our final project, we are only interested in two facial landmarks - the eyes. In our earlier chapter, we have seen that eyes are represented by 6 coordinates, marked in clockwise rotation on an eye image (Figure 6.1).

As observable in the image, a relation can be derived between the height and width of the coordinates known as the *eye aspect ratio* (EAR).[8]

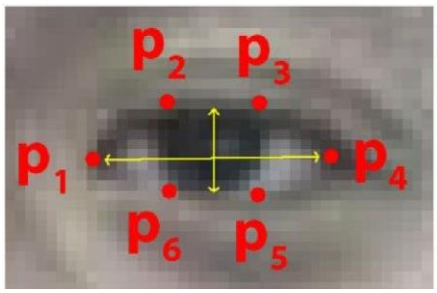

Figure 6.1. p1 to p6 are eye landmarks[8]

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

Here, p1 to p6 are the extracted eye landmark coordinate points. The vertical distance between the eye points contributes to the numerator, and horizontal distance contributes to being the

denominator averaged for two points. The EAR is constant when the eyes are open but drops nearly to zero when eyes are closed or blinks (Figure 6.2).

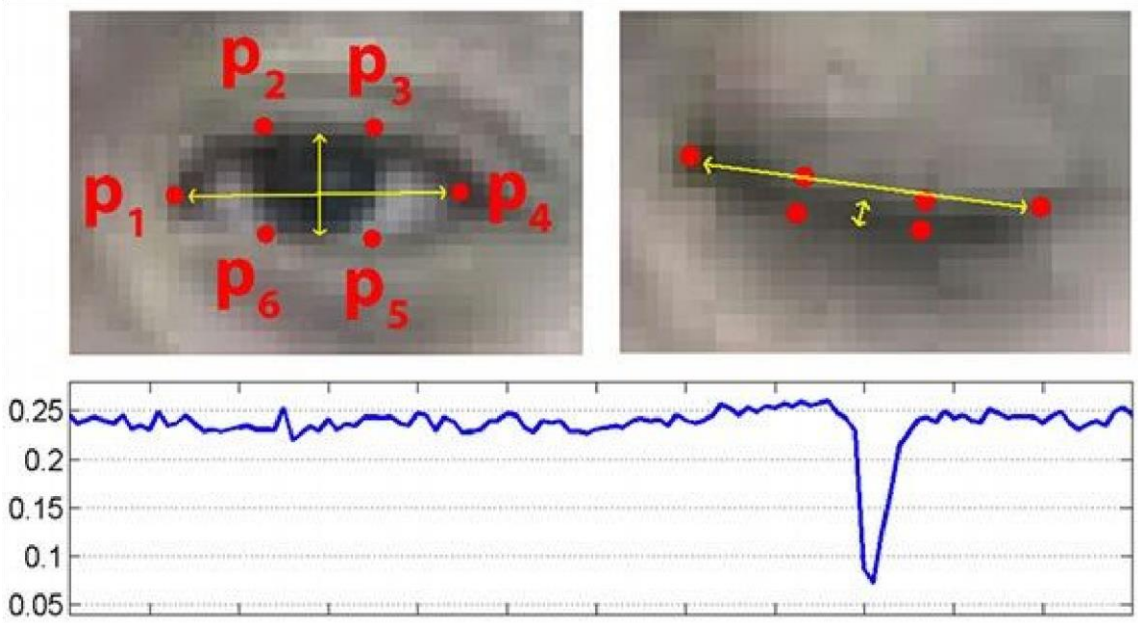

Figure 6.2. (Bottom) Plotting the EAR over several frames of a video sequence. The drop indicates a single blink where EAR drops to near zero.[8]

Hence, in our algorithm, we monitor the EAR to see if the value drops but doesn't increase again, thus inferring that the person has slept. In their paper, Soukupová and Čech also recommended that averaging both eye's EAR together can give us a better estimation.

We initialized two tunable parameters in our implementation namely *EAR Threshold* and *EAR Consecutive frame threshold.* The '*EAR threshold*' variable sets a lower limit on the eye threshold ratio, the value below which will be considered as closed eyes. '*EAR Consecutive frame threshold*' variable keeps a limit on the maximum number of consecutive frames after which the person will be considered asleep and the alarm will go off. We keep the '*EAR Consecutive frame threshold*' high enough so that it doesn't take blinking as a false positive but also, low enough to keep the algorithm sensitive to even shorter impulses of drowsiness or sleepiness. We can make the drowsiness detector more sensitive by decreasing the above variable further.

In our implementation, after experimentation, we found following values suitable for each variable:

1. ***EAR threshold*** - 0.24

2. ***EAR Consecutive frame threshold*** - 30

Following **pseudo code** explains the process better:

while the camera is on: loop through all the frames: detect face; find facial landmarks; extract eye features; calculate EAR; if EAR < EAR_Threshold: increment counter by 1; if counter > EAR_Consecutive_Frame_Threshold: start the ALARM;
Print "ALERT" on the screen
else: stop alarm; reset counter;

The program runs for as long as the camera is on and alerts the driver if the EAR goes below a threshold, indicating that the driver is starting dozing off. The alarm switches off as soon as the driver is awake and opens their eyes again. This whole process runs in an infinite loop and works while the car is running, and simultaneously the program keeps running.

In the next chapter, results and output from the program run are included along with the link for a demo video and actual code.

Chapter 7: Results and Conclusion

7.1 Result and Final output

After running the program for several instances, we could see that the setup was working correctly. Following are some captures from the program in progress:

Figure 7.1 Program work even when the person is wearing spectacles. (a) Open eye EAR, in this case, is 0.32. (b) An alarm is sounded and a warning message is displayed when EAR drops below the threshold

In Figure 7.1, we see that eyes are detected even when the person is wearing spectacles. Open eye EAR, in this case, is 0.32 while the closed eye EAR is similar to the previous example.

As the above output (also the demonstration video) show, the drowsiness detector can detect when a person is at risk of dozing off and then plays a loud alarm to wake them up.

7.2 Inferences and Conclusion

The simple coordinate distance calculation based method used in chapter 6 to calculate the Eye Aspect Ratio to check for open and closed eye saves us from heavy image processing algorithms which are slow and hefty. This method instead gives us a speedy and accurate working setup which can be easily applied and implemented on an embedded computer such as Raspberry Pi with minimal changes and efforts to make it practical and useful.

Due to the smart and minimalist execution of the whole program, it can run with a varied set of changing backgrounds with different lighting conditions, causing zero effect on the performance. Hence, "***Real-time driver distraction and drowsiness monitoring system***" was successfully realized.

Chapter 8: Future work & Scope for Improvements

There is always room for improvement in a particular work done by someone, and this project is no exception to this rule. We tried our best to include plenty of features while trying to improve the functioning of the program at each stage. There are still some ideas and improvements that can be implemented to improve and enhance the Drowsiness detection system to make it more flexible and usable in a variety of situations. Following are some of the suggested additions and improvements:

1. Try and test the whole program using a Raspberry Pi and compatible cameras to check and improve the practical functioning of the project.
2. As an added functionality, check if the *seat belt* is worn or not. Create a simple binary classification model which would take the image of the driver as input and check if the seat-belt is not worn and warn the driver based on the output.
3. Add a *binary classification* model to detect if the *driver is holding a phone* in his/her hand, indicating that the driver is either texting or calling on the phone. Warn the driver in such a case.
4. Using the existing landmark detection, check if a face is detected in the video but landmarks of only one side of the face are detected, indicating that the driver is looking in another direction and is distracted. Alongside, use *head-pose detection* to check if the driver is looking downwards or either side rather than the front. Sound the alarm if this situation continues for consecutive frames.
5. Integrate the system with the ignition of the car so that it works only when the car is moving and not when it is parked.

Some many other functions and improvements can be made to the project, depending on particular use-cases and acceptable cost.

References and Sources

1. Traffic accidents, NCRB Report, Chapter 1: Traffic Accidents, Government of India
2. Top 10 causes of death in the world, WHO (2018)

3. United States Department of Transportation, National Highway Traffic Safety Administration
4. Detecting Driver Drowsiness Based on Sensors: A Review- Arun Sahayadhas, Kenneth Sundaraj and Murugappan Murugappan
5. Liu, C.C.; Hosking, S.G.; Lenné, M.G. Predicting driver drowsiness using vehicle measures: Recent insights and future challenges. J. Saf. Res. 2009, 40, 239–245.
6. Akin, M.; Kurt, M.; Sezgin, N.; Bayram, M. Estimating vigilance level by using EEG and EMG signals. Neural Comput. Appl. 2008, 17, 227–236.
7. Duy Tran; Ha Manh Do; Weihua Sheng; He Bai; Girish Chowdhary - Real-time detection of distracted driving based on deep learning
8. Soukupová, Tereza and Jan Cech. "Real-Time Eye Blink Detection using Facial Landmarks." (2016).
9. A detailed explanation of Histogram of Oriented Gradients
10. N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, San Diego, CA, USA, 2005, pp. 886-893 vol. 1.
11. Histogram of Oriented Gradients tutorial-NMS box detection.
12. Kazemi, V., & Sullivan, J. (2014). One millisecond face alignment with an ensemble of regression trees. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 1867-1874.
13. Dlib Library implementation
14. i-Bug dataset used to train the pre-trained Dlib model
15. Qazi Faraz Ahmad | News18 | Bus accident in Delhi

Websites and other Sources for reference

1. Histogram Oriented Gradients Object Detection
2. Learn OpenCV
3. A convenience function package to work with images easily.