B. TECH. PROJECT REPORT

On

DRIVER DROWSINESS DETECTION

BY

Achanta Vishnu Vardhan

DISCIPLINE OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY INDORE

December 2018

DRIVER DROWSINESS DETECTION

A PROJECT REPORT

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

of
BACHELOR OF TECHNOLOGY
in
ELECTRICAL ENGINEERING

Submitted by:
Achanta Vishnu Vardhan

Guided by:
Dr. Vivek Kanhangad,
Associate Professor,
Electrical Engineering,
Indian Institute of Technology Indore

INDIAN INSTITUTE OF TECHNOLOGY INDORE
December 2018

CANDIDATE'S DECLARATION

I hereby declare that the project entitled **"Driver Drowsiness Detection"** submitted in partial fulfilment for the award of the degree of Bachelor of Technology in **Electrical Engineering** completed under the supervision of **Dr. Vivek Kanhangad, Associate Professor, Electrical Engineering, IIT Indore** is an authentic work.

Further, I declare that I have not submitted this work for the award of any other degree elsewhere.

Achanta Vishnu Vardhan
1500002001
Discipline of Electrical Engineering
Indian Institute of Technology Indore

CERTIFICATE by BTP Guide

It is certified that the above statement made by the student is correct to the best of my knowledge and belief.

Dr. Vivek Kanhangad,
Associate Professor,
Discipline of Electrical Engineering,
Indian Institute of Technology Indore.

Preface

This report on “Driver Drowsiness Detection" is prepared under the guidance of Dr. Vivek Kanhangad.

This report mainly focuses on face detection (HOG, sliding windows) and feature extraction on the detected face to detect the status of eyes of the driver. Thus giving us an idea whether or not the driver is in a drowsy state.

I have tried to the best of my abilities and knowledge to explain the content of my project in a lucid manner.

Achanta Vishnu Vardhan
B.Tech. IV Year
Discipline of Electrical Engineering
IIT Indore

Acknowledgements

I have taken efforts in this project. However, it would not have been possible without the kind support and help of Dr. Vivek Kanhangad and IIT Indore. I am highly indebted to Dr. Vivek Kanhangad for his guidance and constant supervision as well as for providing necessary information regarding the project & also for his support in completing the project.

I would like to express my gratitude towards my parents for their kind co-operation and encouragement which helped me in completion of this project. I would like to express my special gratitude and thanks to Electrical engineering department for giving me such attention and time.

Achanta Vishnu Vardhan
B.Tech. IV Year
Discipline of Electrical Engineering
IIT Indore

Abstract

Driver fatigue is one of the major causes of accidents in the world. Detecting the drowsiness of the driver is one of the surest ways of measuring driver fatigue. In this project I aim to develop a prototype drowsiness detection system. This system works by monitoring the eyes of the driver and sending an alert when he/she is drowsy.

The system so designed is a non-intrusive real-time monitoring system. The priority is on improving the safety of the driver without being obtrusive. In this project the eye blink of the driver is detected. If the drivers eyes remain closed for more than a certain period of time, the driver is said to be drowsy and an alert is sent. The programming for this is done in OpenCV using the dlib library for the detection of facial features.

Table of Contents

List of Figures

Chapter 1
INTRODUCTION

1.1 Background:

The development of technology allows introducing more advanced solutions in everyday life. This makes work less exhausting for employees, and also increases the work safety. Vision-based systems are becoming more popular and are more widely used in different applications. These systems can be used in industry (e.g. sorting systems), transportation (e.g. traffic monitoring), airport security (e.g. suspect detection systems), and in the end-user complex products such as cars (car parking camera). Such complex systems could also be used to detect vehicle operator fatigue using vision-based solutions. Fatigue is such a psychophysical condition of a man, which does not allow for a full concentration. It influences the human response time, because the tired person reacts much slower, compared to the rested one. Appearance of the first signs of a fatigue can become very dangerous, especially for such professions like drivers. Nowadays, more and more professions require long-term concentration. People, who work for transportation business (car and truck drivers, steersmen, airplane pilots), must keep a close eye on the road, so they can react to sudden events (e.g. road accidents, animals on the road, etc.) immediately. Long hours of driving causes the driver fatigue and, consequently, reduces her/him response time.

Driver drowsiness detection is a car safety technology which helps prevent accidents caused by the driver getting drowsy. Various studies have suggested that around 20% of all road accidents are fatigue-related, up to 50% on certain roads. During long journeys, it's possible that the driver may lose attention because of drowsiness, which may be a potential reason for fatal accidents. With technologies like Driver Drowsiness Detection getting it is possible to detect driver's driving behaviour that may prove fatal to the vehicle as well as the people boarding it.

Having such sleep detection system in cars embedded in vehicles could protect precious lives and property worth billion dollars. The outcome would be positive – it would be suitable for fleet owners as well as individual vehicle users. In either case, the objective is identical by sleep detection while driving.

This system is based on driver visual analysis using image processing techniques. Computer vision can be a natural and non-intrusive technique for monitoring driver's sleepiness from the images taken by some cameras placed in front of the user. These approaches are effective because of the occurrence of sleepiness is reflected through the driver's face appearance and eyes activity.

The driver drowsiness detection system uses Image Processing to analyse the driver's eye blink pattern by sitting on the vehicle's dashboard. If the eye lid movements are abnormal than usual then the detection system triggers the alarm thus alerting the driver about the condition.

1.2 Literature Overview

A driver who falls asleep at the wheel loses control of the vehicle, an action which often results in a crash with either another vehicle or stationary objects. In order to prevent these devastating accidents, the state of drowsiness of the driver should be monitored. The following measures have been used widely for monitoring drowsiness:

(1) Vehicle-based measures— A number of metrics, including deviations from lane position, movement of the steering wheel, pressure on the acceleration pedal, etc., are constantly monitored and any change in these that crosses a specified threshold indicates a significantly increased probability that the driver is drowsy.

(2) Behavioural measures — The behaviour of the driver, including yawning, head pose, etc., is monitored through a camera and the driver is alerted if any of these drowsiness symptoms are detected.

(3) Physiological measures — The correlation between physiological signals (electrocardiogram (ECG), electromyogram (EMG), electrooculogram (EoG) and electroencephalogram (EEG)) and driver drowsiness has been studied by many researchers.

Other than these three, researchers have also used subjective measures where drivers are asked to rate their level of drowsiness either verbally or through a questionnaire. The intensity of drowsiness is determined based on the rating [13]. Such tasks could be complicated take a lot of effort and data collection.

1.3 Motivation

Life is precious and no number of words suffice to evaluate it. It's, therefore, imperative to protect it from fatal consequences while driving a vehicle. Driving a vehicle involves coordination of the locomotor system along with the healthy function of the brain. When the driver feels drowsy, it may unsettle the balance and may lead to erratic driving causing potential accidents. While driving, you may feel drowsy when you're under driving fatigue because of continuous driving for several hours. It's here that the driver drowsiness detection plays a significant role in preventing accidents that could otherwise cause massive loss of life and property.

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).

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[1]. This method for eye blink detection is fast, efficient, and easy to implement. Also we wouldn't be needing to collect any additional data, like in case of steering wheel pattern systems where the data collection could is challenging and dangerous if necessary precautions are not taken.

Chapter 2
Face Detection

In this chapter we shall discuss briefly the underlying techniques of face detection. The method could be divided in 3 sub-divisions. They histogram of oriented gradients, construction of an image pyramid and a sliding window mechanism.

2.1 Histogram of Oriented Gradients

The histogram of oriented gradients (HOG) is a feature descriptor used in computer vision and image processing for the purpose of object detection. The technique counts occurrences of gradient orientation in localized portions of an image. This method is similar to that of edge orientation histograms, scale-invariant feature transform descriptors, and shape contexts, but differs in that it is computed on a dense grid of uniformly spaced cells and uses overlapping local contrast normalization for improved accuracy.

The essential thought behind the histogram of oriented gradients descriptor is that local object appearance and shape within an image can be described by the distribution of intensity gradients or edge directions. The image is divided into small connected regions called cells, and for the pixels within each cell, a histogram of gradient directions is compiled. The descriptor is the concatenation of these histograms. For improved accuracy, the local histograms can be contrast-normalized by calculating a measure of the intensity across a larger region of the image, called a block, and then using this value to normalize all cells within the block. This normalization results in better invariance to changes in illumination and shadowing.

The HOG descriptor has a few key advantages over other descriptors. Since it operates on local cells, it is invariant to geometric and photometric transformations, except for object orientation. Such changes would only appear in larger spatial regions. Moreover, as Dalal and Triggs discovered, coarse spatial sampling, fine orientation sampling, and strong local photometric normalization permits the individual body movement of pedestrians to be ignored so long as they maintain a roughly upright position. The HOG descriptor is thus particularly suited for human detection in images.

Let’s now look into different stages of HOG.

2.1.1 Gradient Computation

The first step of calculation in many feature detectors in image pre-processing is to ensure normalized color and gamma values. As Dalal and Triggs point out, however, this step can be omitted in HOG descriptor computation, as the ensuing descriptor normalization essentially achieves the same result. Image pre-processing thus provides little impact on performance. Instead, the first step of calculation is the computation of the gradient values. The most common method is to apply the 1-D centered, point discrete derivative mask in one or both of the horizontal and vertical directions. Specifically, this method requires filtering the color or intensity data of the image with the following filter kernels:

$[-1, 0, 1]$ and $[-1, 0, 1]^T$

Dalal and Triggs tested other, more complex masks, such as the 3x3 Sobel mask or diagonal masks, but these masks generally performed more poorly in detecting humans in images. They also experimented with Gaussian smoothing before applying the derivative mask, but similarly found that omission of any smoothing performed better in practice.

2.1.2 Orientation binning

The second step of calculation is creating the cell histograms. Each pixel within the cell casts a weighted vote for an orientation-based histogram channel based on the values found in the gradient computation. The cells themselves can either be rectangular or radial in shape, and the histogram channels are evenly spread over 0 to 180 degrees or 0 to 360 degrees, depending on whether the gradient is "unsigned" or "signed". Dalal and Triggs found that unsigned gradients used in conjunction with 9 histogram channels performed best in their human detection experiments. As for the vote weight, pixel contribution can either be the gradient magnitude itself, or some function of the magnitude. In tests, the gradient magnitude itself generally produces the best results. Other options for the vote weight could include the square root or square of the gradient magnitude, or some clipped version of the magnitude.

2.1.3 Descriptor Blocks

To account for changes in illumination and contrast, the gradient strengths must be locally normalized, which requires grouping the cells together into larger, spatially connected blocks. The HOG descriptor is then the concatenated vector of the components of the normalized cell histograms from all of the block regions. These blocks typically overlap, meaning that each cell contributes more than once to the final descriptor. Two main block geometries exist: rectangular R-HOG blocks and circular C-HOG blocks. R-HOG blocks are generally square grids, represented by three parameters: the number of cells per block, the number of pixels per cell, and the number of channels per cell histogram. In the Dalal and Triggs human detection experiment, the optimal parameters were found to be four 8x8 pixels cells per block (16x16 pixels per block) with 9 histogram channels. Moreover, they found that some minor improvement in performance could be gained by applying a Gaussian spatial window within each block before tabulating histogram votes in order to weight pixels around

the edge of the blocks less. The R-HOG blocks appear quite similar to the scale-invariant feature transform (SIFT) descriptors; however, despite their similar formation, R-HOG blocks are computed in dense grids at some single scale without orientation alignment, whereas SIFT descriptors are usually computed at sparse, scale-invariant key image points and are rotated to align orientation. In addition, the R-HOG blocks are used in conjunction to encode spatial form information, while SIFT descriptors are used singly.

Circular HOG blocks (C-HOG) can be found in two variants: those with a single, central cell and those with an angularly divided central cell. In addition, these C-HOG blocks can be described with four parameters: the number of angular and radial bins, the radius of the center bin, and the expansion factor for the radius of additional radial bins. Dalal and Triggs found that the two main variants provided equal performance, and that two radial bins with four angular bins, a center radius of 4 pixels, and an expansion factor of 2 provided the best performance in their experimentation(to achieve a good performance, at last use this configure). Also, Gaussian weighting provided no benefit when used in conjunction with the C-HOG blocks. C-HOG blocks appear similar to shape context descriptors, but differ strongly in that C-HOG blocks contain cells with several orientation channels, while shape contexts only make use of a single edge presence count in their formulation.

2.1.4 Block Normalization

Dalal and Triggs explored four different methods for block normalization. Let "v" be the non-normalized vector containing all histograms in a given block, $\|v\|_k$ be its *k*-norm for k = 1, 2 and *e* be some small constant (the exact value, hopefully, is unimportant). Then the normalization factor can be one of the following:

L2 – norm: $$\boldsymbol{f} = \frac{\boldsymbol{v}}{\sqrt{\|\boldsymbol{v}\|_2^2 + \boldsymbol{e^2}}} \tag{2.1}$$

L2-hys: L2-norm followed by clipping (limiting the maximum values of v to 0.2) and renormalizing, as in

L1 – norm: $$\boldsymbol{f} = \frac{\boldsymbol{v}}{\|\boldsymbol{v}\|_1 + \boldsymbol{e}} \tag{2.2}$$

L1 – sqrt: $$\boldsymbol{f} = \sqrt{\frac{\boldsymbol{v}}{\|\boldsymbol{v}\|_1 + \boldsymbol{e}}} \tag{2.3}$$

In addition, the scheme L2-hys can be computed by first taking the L2-norm, clipping the result, and then renormalizing. In their experiments, Dalal and Triggs found the L2-hys, L2-norm, and L1-sqrt schemes provide similar performance, while the L1-norm provides slightly less reliable performance; however, all four methods showed very significant improvement over the non-normalized data.

2.1.5 Examples of HOG:

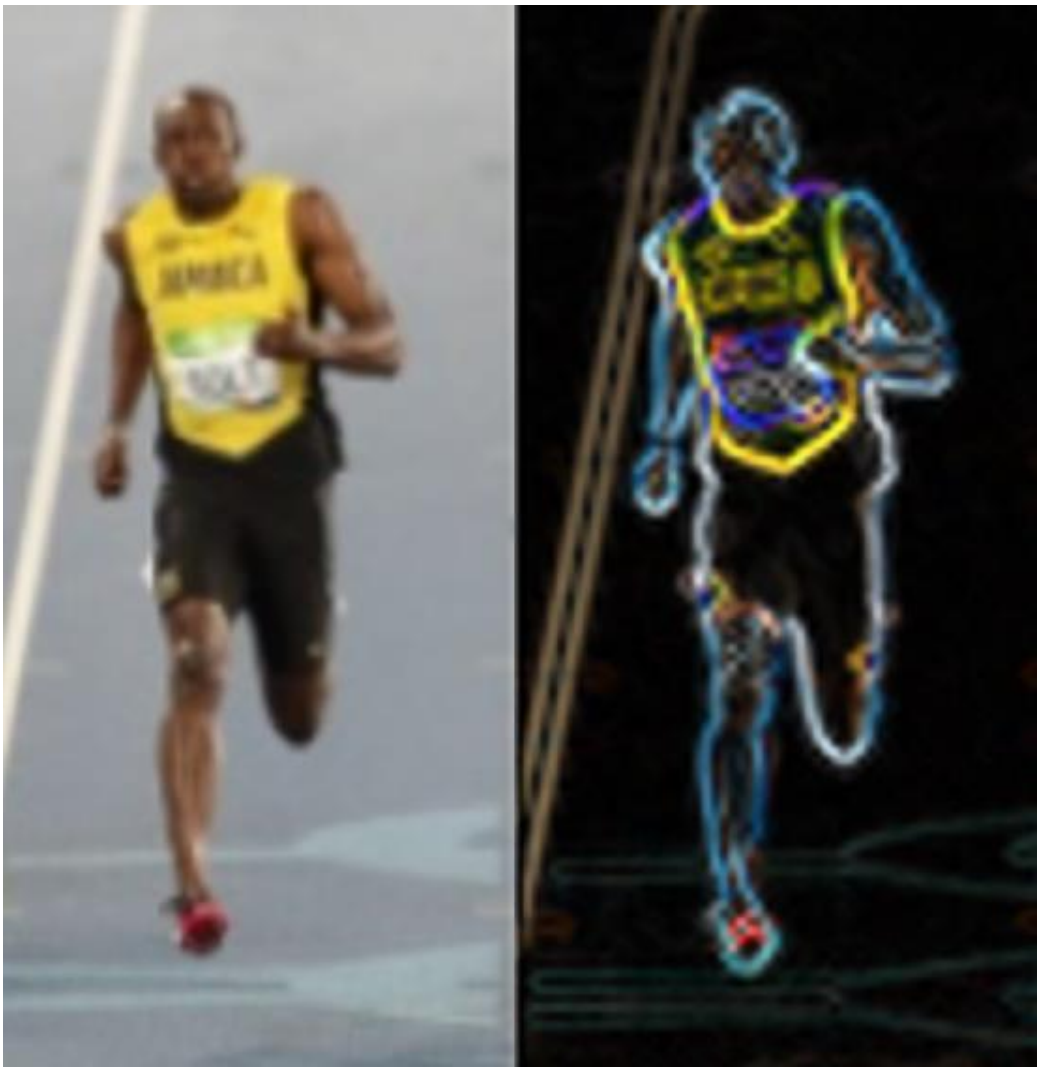

Figure 2.1: Sample image and generated histogram of oriented gradients of the image [26].

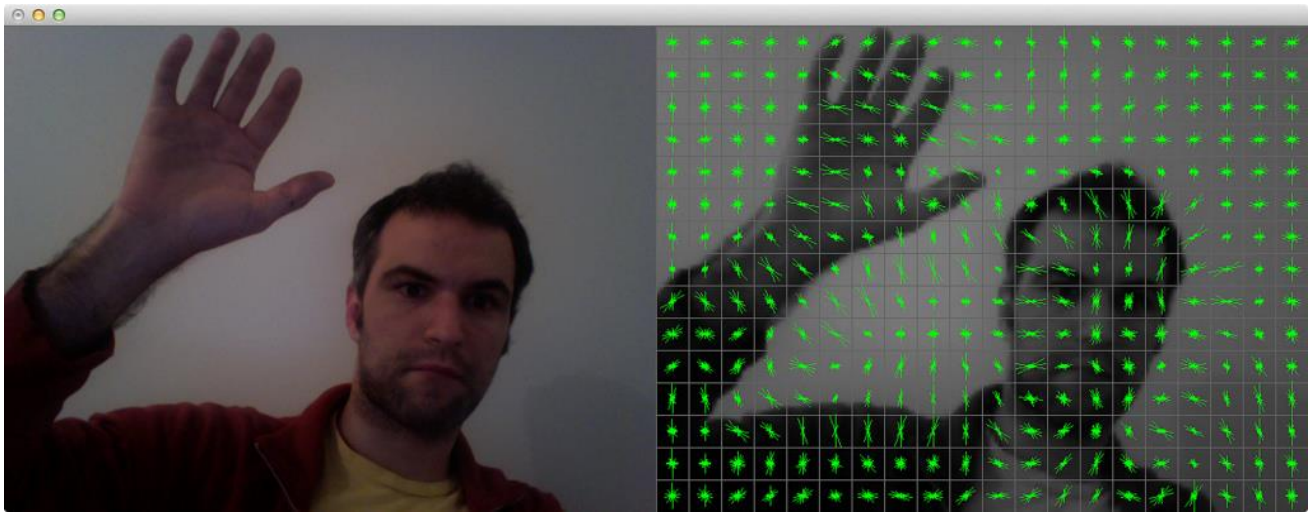

Figure 2.2: Sample image and generated histogram of oriented gradients of the image [25].

2.2 Image Pyramid

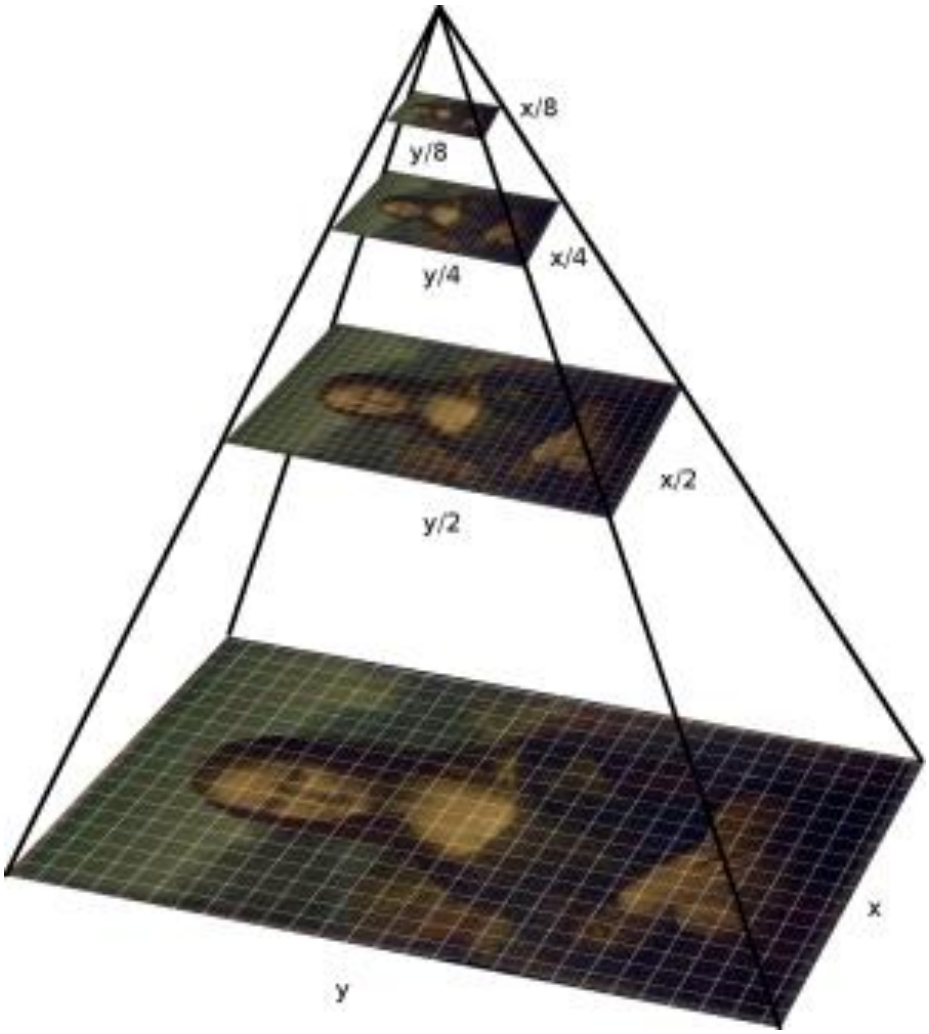

Figure 3: An example of an image pyramid. At each layer of the pyramid the image is downsized and (optionally) smoothed [22].

An “image pyramid” is a multi-scale representation of an image.

Utilizing an image pyramid allows us to find objects in images at different scales of an image. And when combined with a sliding window we can find objects in images in various locations. At the bottom of the pyramid we have the original image at its original size (in terms of width and height). And at each subsequent layer, the image is resized (subsampled) and optionally smoothed (usually via Gaussian blurring). The image is progressively subsampled until some stopping criterion is met, which is normally a minimum size has been reached and no further subsampling needs to take place.

2.3 Sliding Window

Sliding windows play an integral role in object classification, as they allow us to localize exactly “where” in an image an object resides. Utilizing both a sliding window and an image pyramid we are able to detect objects in images at various scales and locations. In the context of computer vision (and as the name suggests), a sliding window is a rectangular region of fixed width and height that “slides” across an image, such as in the following figures:

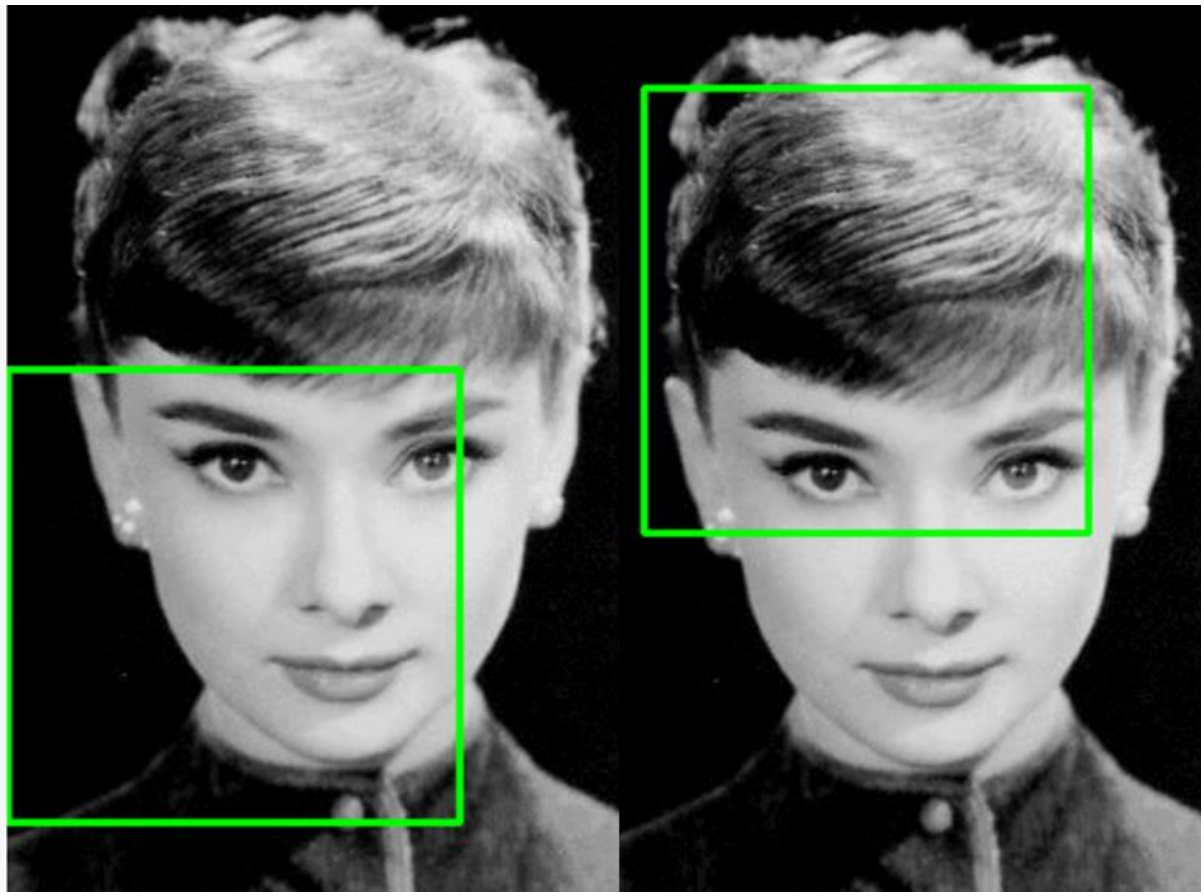

Figure 4 : Showing different positions of the sliding window [24].

For each of these windows, we would normally take the window region and apply an image classifier to determine if the window has an object that interests us — in this case, a face. Combined with image pyramids we can create image classifiers that can recognize objects at varying scales and locations in the image. These techniques, while simple, play an absolutely critical role in object detection and image classification.

Chapter 3
Drowsiness Detection based on EAR

In this chapter we shall see how an in depth analysis of the facial landmarks can help us determine the state of the driver. The state of driver can be interpreted based the eyes of the driver. If the driver's eyes are closed for a minimum amount of amount, we can say the driver is drowsy. To exploit this idea we can use the feature extraction and shape predictor methods. Once we get the data on eyes of the driver we can proceed to calculate the extent to which the eyes are open or closed. This extent is based on EAR (eye aspect ratio) which discussed in the later stages of this chapter.

3.1 Feature Extraction

In machine learning, pattern recognition and in image processing, feature extraction starts from an initial set of measured data and builds derived values (features) intended to be informative and non-redundant, facilitating the subsequent learning and generalization steps. Feature extraction is a dimensionality reduction process, where an initial set of raw variables is reduced to more manageable groups (features) for processing, while still accurately and completely describing the original data set.

When the input data to an algorithm is too large to be processed and it is suspected to be redundant (e.g. the same measurement in both feet and meters, or the repetitiveness of images presented as pixels), then it can be transformed into a reduced set of features (also named a feature vector). Determining a subset of the initial features is called feature selection. The selected features are expected to contain the relevant information from the input data, so that the desired task can be performed by using this reduced representation instead of the complete initial data.

3.2 Shape Predictor

Detecting facial landmarks is a subset of the shape prediction problem. Given an input image (and normally an ROI that specifies the object of interest), a shape predictor attempts to localize key points of interest along the shape. In the context of facial landmarks, our goal is detect important facial structures on the face using shape prediction methods.

Detecting facial landmarks is therefore a two-step process:

Step 1: Localize the face in the image.

Step 2: Detect the key facial structures on the face ROI.

The programs first loads the image and then detects the face of the driver using the dlib.frontal_face_detector() function provided by the dlib library. Then we proceed to predict the locations of facial landmarks by using the predictor() function a model which needs the location of the predictor being used as its input. Once the processing is done we get set of variables containing the data points (vector locations of the facial landmarks) for each of the recognized face.

From here we can make use of necessary data points to calculate the EAR of both the eyes (discussed in the next chapter).

The model is given below:

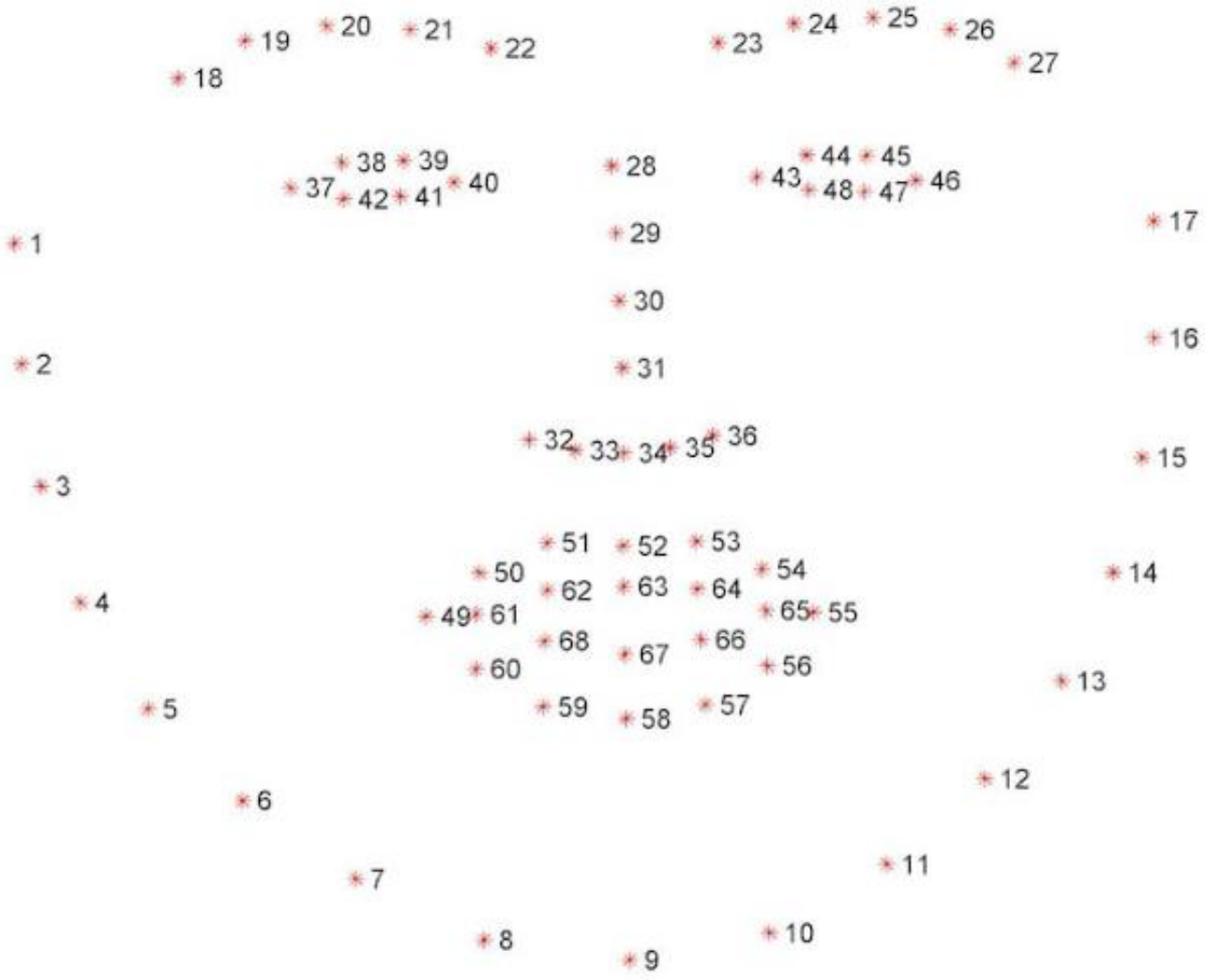

Figure 3.1: Facial Landmarks Predictor Model [22]

3.3 Eye Aspect Ratio

In this approach we only use two sets of facial structures “the eyes”. Each eye is represented by 6 (x, y)-coordinates (as seen in the shape predictor), 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:

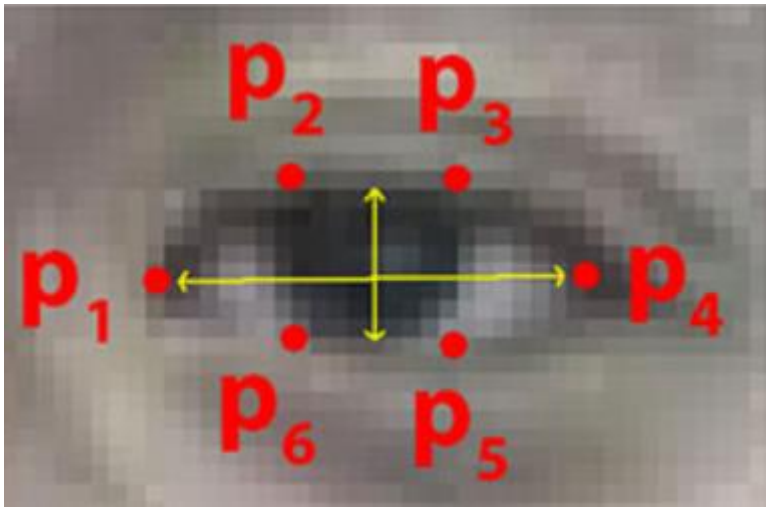

Figure 3.2: Sample positioning vectors determining the positions of the eyes[1].

Based on this image, we should take away one key point, “There is a relation between the width and the height of these coordinates”.

Based on the work by Soukupová and Čech in their 2016 paper, *Real-Time Eye Blink Detection using Facial Landmarks*, we can then derive an equation that reflects this relation called the eye aspect ratio (EAR):

$$\mathbf{EAR} = \frac{\|\boldsymbol{p_2} - \boldsymbol{p_6}\| + \|\boldsymbol{p_3} - \boldsymbol{p_5}\|}{2\|\boldsymbol{p_1} - \boldsymbol{p_4}\|} \tag{4.1}$$

Where p1… p6 are 2D facial landmark locations as depicted in the above figure. 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.

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 from Soukupová and Čech [1]:

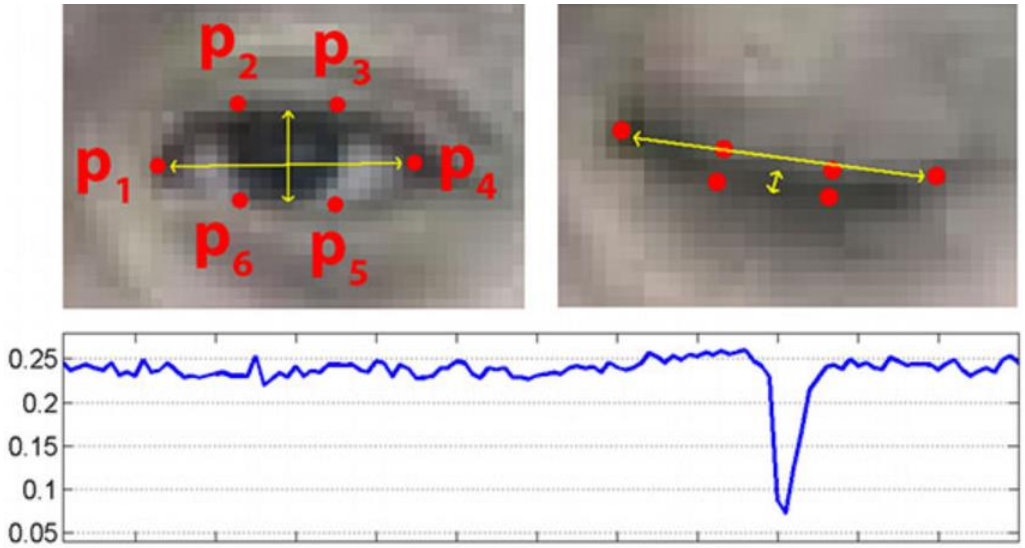

Figure 3.3: Top-left: A visualization of eye landmarks when then the eye is open — the eye aspect ratio here would be large(r) and relatively constant over time. However, once the person blinks (top-right) the eye aspect ratio decreases dramatically, approaching zero. Bottom: Plotting the eye aspect ratio over time. The dip in the eye aspect ratio indicates a blink (Figure 1 of Soukupová and Čech)[1].

Thus the EAR is calculated and when the EAR is continuously low for over 2 secs we can eliminate the possibility that the driver is blinking and be sure that he is either sleeping or about doze off. Then we can send an alert in form of a sound signal to warn the driver.

The EAR for both the eyes is calculated and averaged, when the average value goes below 0.2, the eyes are in closed state.

Chapter 4
Algorithm and Results

Let's now discuss flow of the algorithm. The code reads the provided input file (or the webcam feed (or) the feed of the camera placed directly opposite to the driver). Then we proceed to analyse the feed frame by frame. Each frame is passed through the get_frontal_face_detector() function. The function returns a set of bounding points for each detected face in the frame. We then consider the face with maximum area as the driver would be the one closet to the camera. This function uses face detection techniques discussed in chapter 2.

Once we have identified the face of the driver, we can start by calculating the position vectors for different parts of the face. The shape predictor we use in this program is based on a 68 point landmark prediction of the face as discussed in chapter 3. The predictor returns the 68 landmarks of the detected. We can now proceed to calculate the EAR of both the eyes.

Once the EAR is calculated and averaged for both the eyes, we compare it with a fixed value of 0.2. If the calculated value is less than 0.2 we start a counter to count the number of successive frames for which the value is less than 0.2. Once the counter reaches the count of 8 or above we can be definitive that the driver is drowsy and not blinking. At this point we can send an alert to the driver to help him get his attention back. This process is repeated over and over. The process has also been laid out briefly in form of a flow chart in Figure 5.1.

4.1 Flowchart

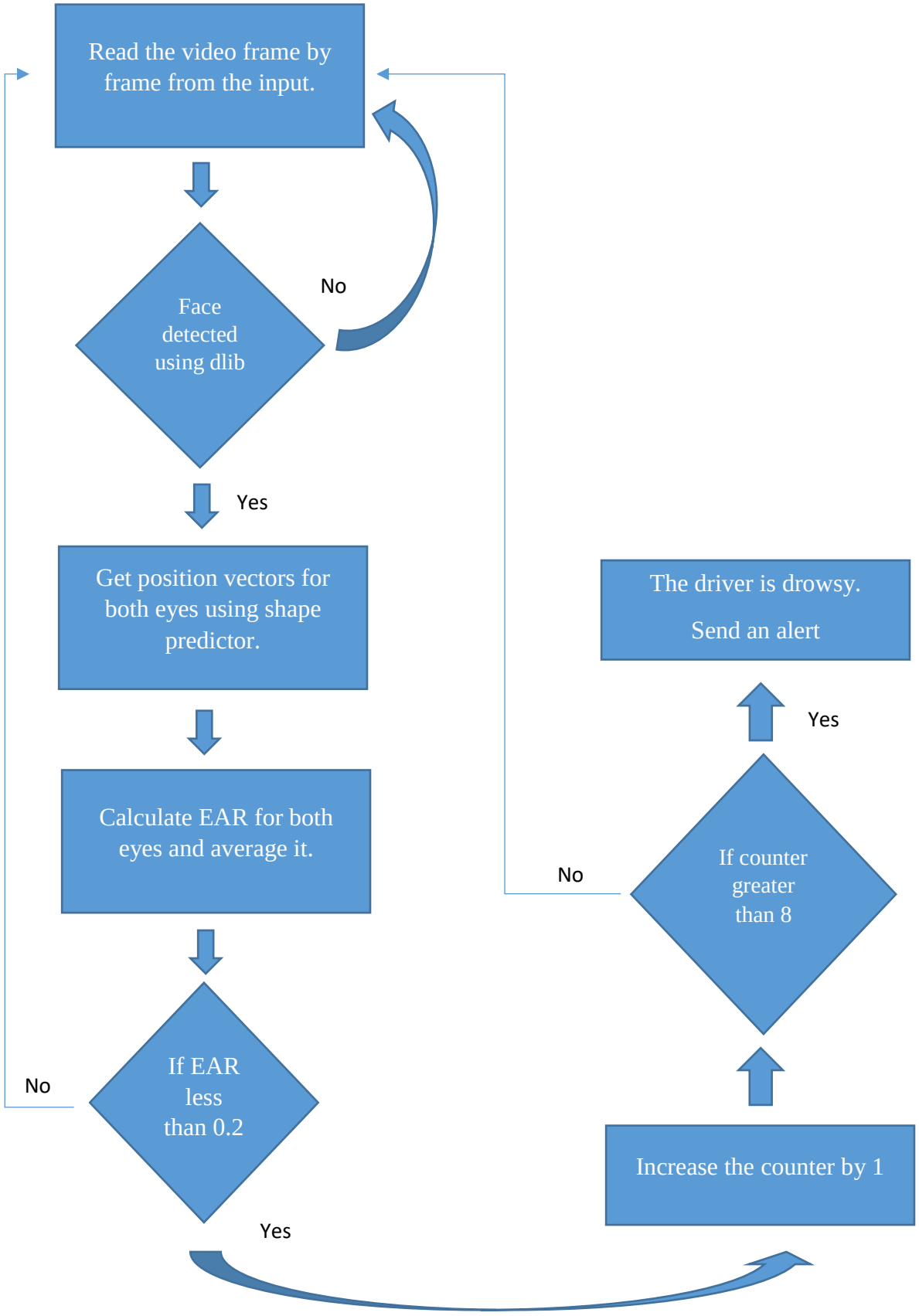

Figure 4.1: A simple flow chart of the algorithm implemented.

4.2 Results

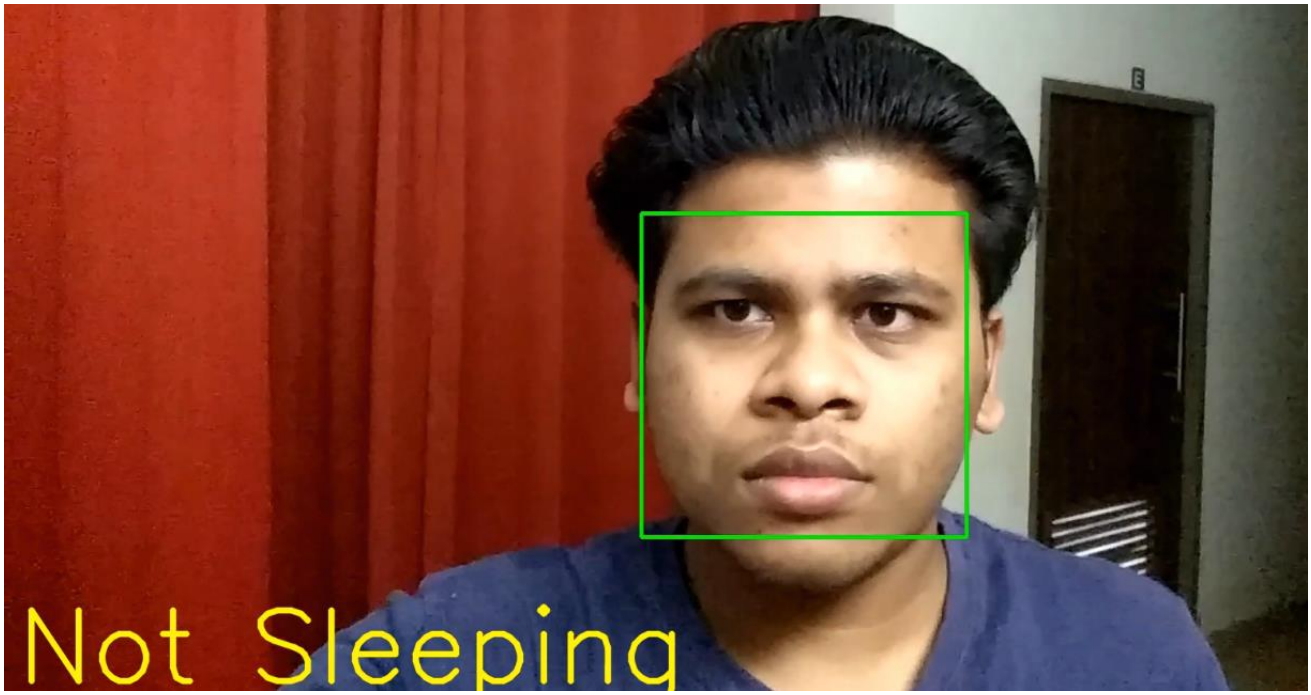

Figure 4.2: Status when the eyes are open and under over illumination.

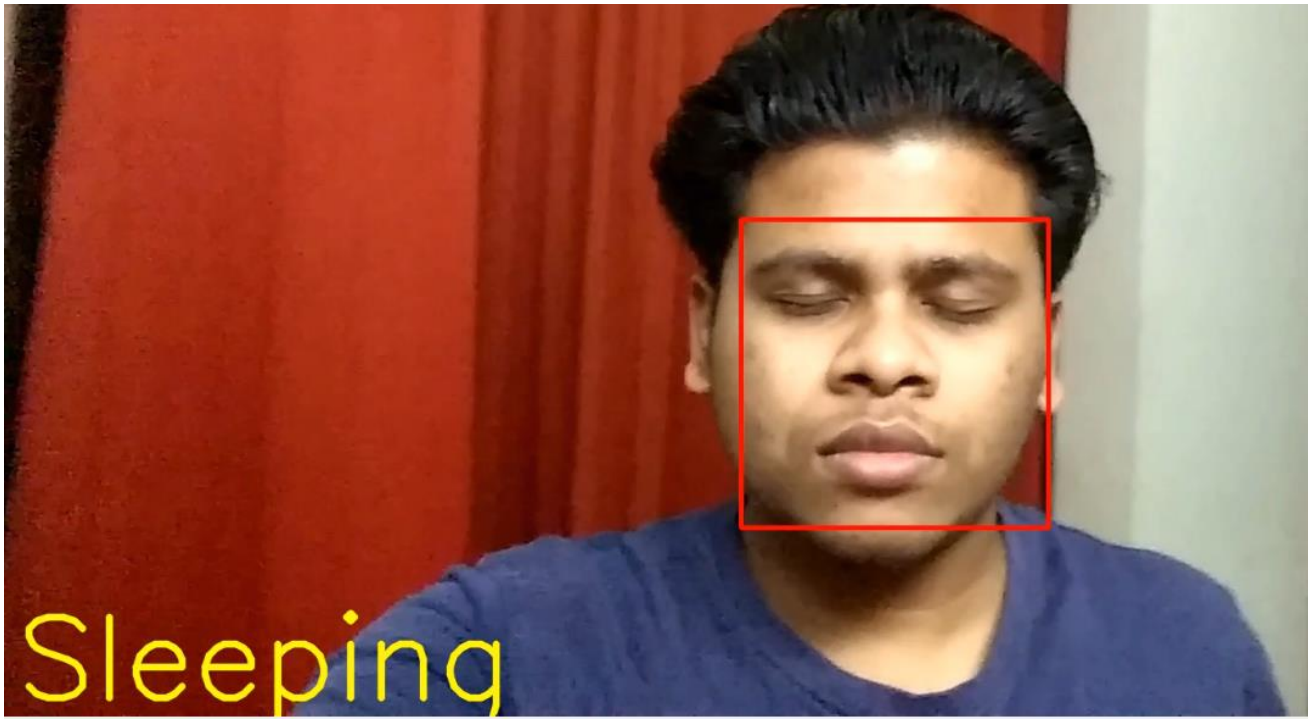

Figure 4.3: Status when the eyes are closed and under over illumination

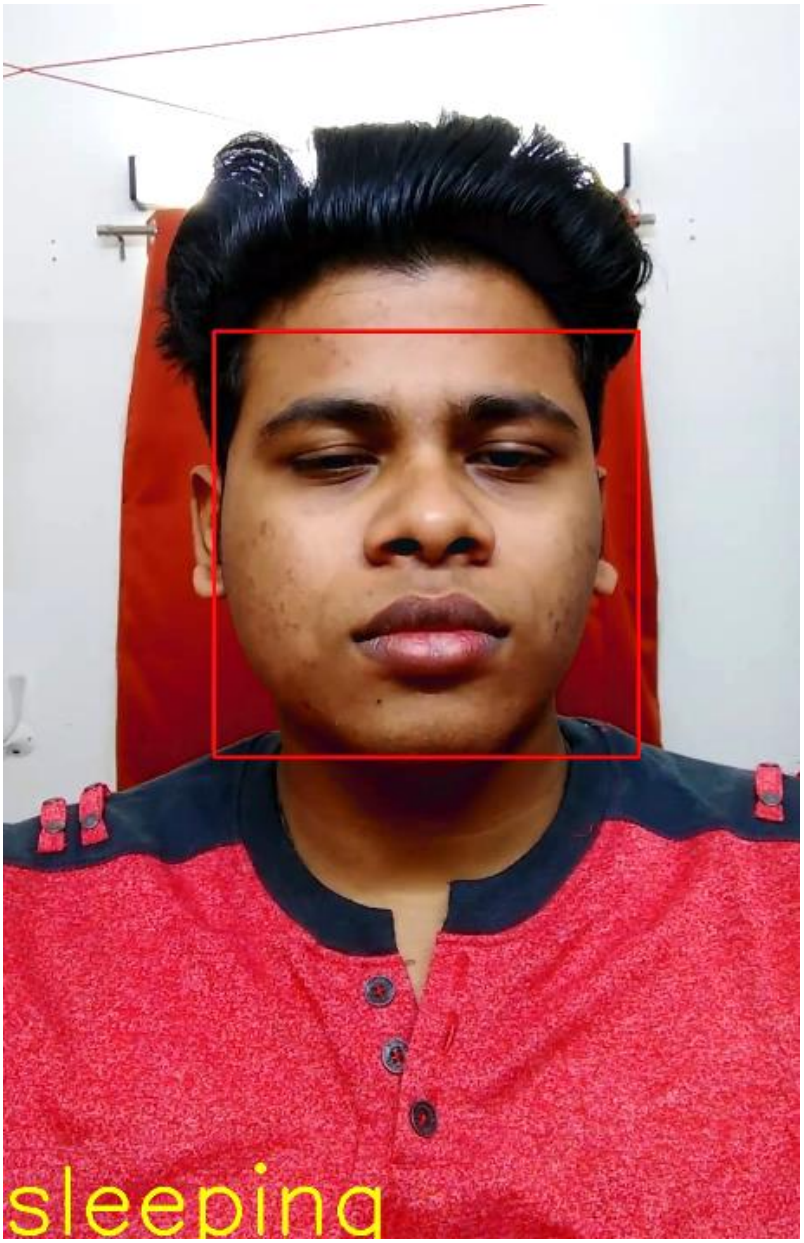

Figure 4.4: Status when the eyes are closed and under normal illumination

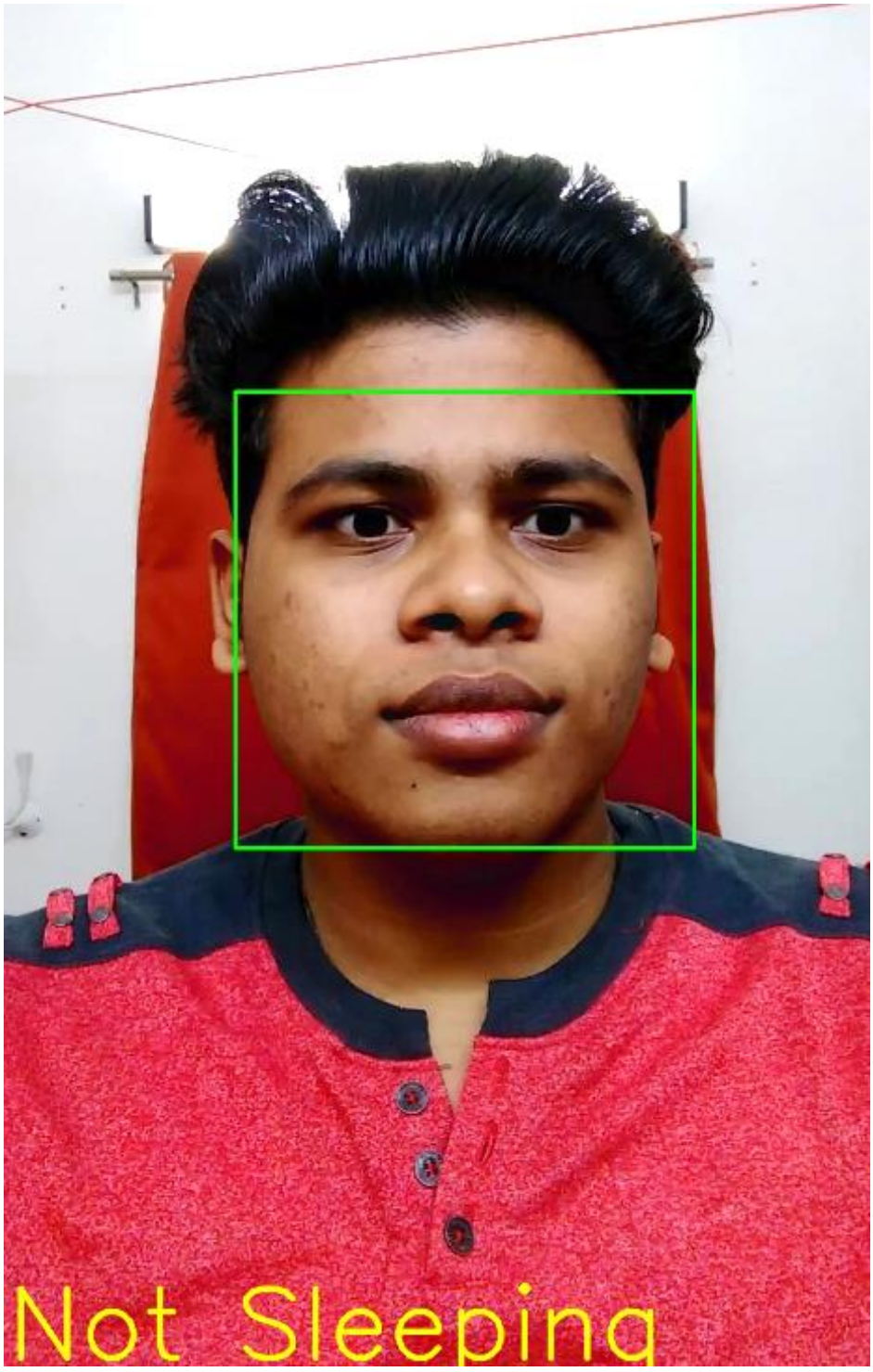

Figure 4.5: Status when the eyes are open and under normal illumination

Figure 4.6: Status when the eyes are open and under low illumination

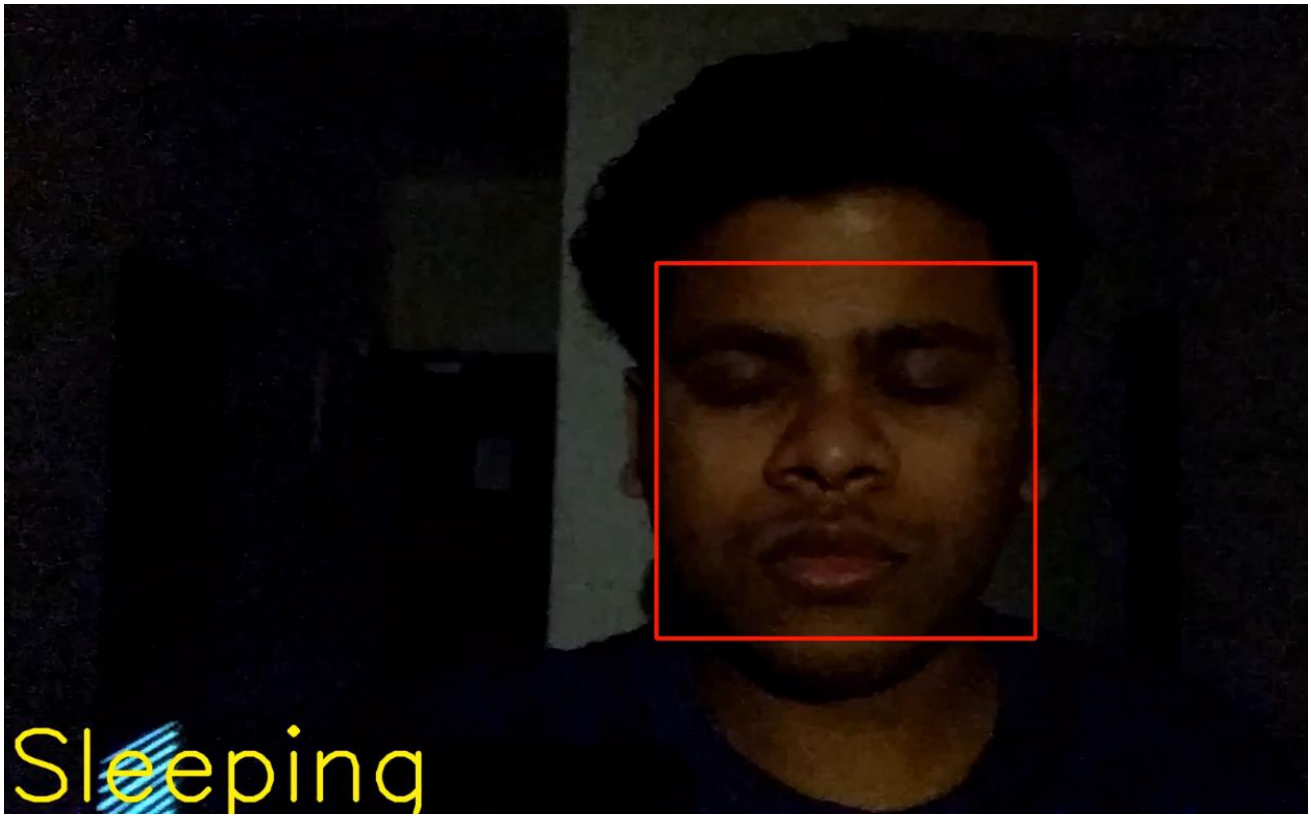

Figure 4.7: Status when the eyes are closed and under low illumination

Chapter 5
Conclusion

This drowsiness detector hinged on two important computer vision techniques:

- Facial landmark detection
- Eye aspect ratio

Facial landmark prediction is the process of localizing key facial structures on a face, including the eyes, eyebrows, nose, mouth, and jawline.

Specifically, in the context of drowsiness detection, we only needed the eye regions. Once we have our eye regions, we can apply the eye aspect ratio to determine if the eyes are closed. If the eyes have been closed for a sufficiently long enough period of time, we can assume the user is at risk of falling asleep and sending an alert to grab their attention.

The system was tested for different people in different ambient lighting conditions (daytime and night-time). The face is kept at an optimum distance, then the system is able to detect drowsiness. The result is good and can be implemented in real-time systems as well.

5.1 Future Work

In the real time driver drowsiness detection system is required to slow down a vehicle automatically when drowsiness level crosses a certain limit. Instead of threshold drowsiness level it is suggested to design a continuous scale driver drowsiness detection system. It monitors the level of drowsiness continuously and when this level exceeds a certain value a signal is generated which controls the hydraulic braking system of the vehicle.

In addition to this, we can simultaneously monitor the head pose of the driver and use the data along with drowsiness detection for a more accurate monitoring of the driver.

References

1.) Soukupová, Tereza and Jan Cech. "**Real-Time Eye Blink Detection using Facial Landmarks.**" (2016).

2.) A. Asthana, S. Zafeoriou, S. Cheng, and M. Pantic. "**Incremental face alignment in the wild**". *In Conference on Computer Vision and Pattern Recognition, 2014*.

3.) L. M. Bergasa, J. Nuevo, M. A. Sotelo, and M. Vazquez. "**Real-time system for monitoring driver vigilance**". *In IEEE Intelligent Vehicles Symposium, 2004.*

4.) M. Chau and M. Betke. "**Real time eye tracking and blink detection with USB cameras**". *Technical Report 2005-12, Boston University Computer Science, May 2005.*

5.) T. Danisman, I. Bilasco, C. Djeraba, and N. Ihaddadene. "**Drowsy driver detection system using eye blink patterns**". *In Machine and Web Intelligence (ICMWI), Oct 2010.*

6.) H. Dinh, E. Jovanov, and R. Adhami. "**Eye blink detection using intensity vertical projection**". *In International Multi-Conference on Engineering and Technological Innovation, IMETI 2012*.

7.) M. Divjak and H. Bischof. "**Eye blink based fatigue detection for prevention of computer vision syndrome**". *In IAPR Conference on Machine Vision Applications, 2009.*

8.) T. Drutarovsky and A. Fogelton. "**Eye blink detection using variance of motion vectors"**. *In Computer Vision - ECCV Workshops. 2014*.

9.) W. H. Lee, E. C. Lee, and K. E. Park. "**Blink detection robust to various facial poses**". *Journal of Neuroscience Methods, Nov. 2010.*

10.) Medicton group. "**The system I4Control**". *http://www.i4tracking.cz/.*

11.) G. Pan, L. Sun, Z. Wu, and S. Lao. "**Eyeblink-based anti-spoofing in face recognition from a generic webcamera**". *In ICCV, 2007.*

12.) S. Ren, X. Cao, Y. Wei, and J. Sun. "**Face alignment at 3000 fps via regressing local binary features**". *In Proc. CVPR, 2014.*

13.) A. Sahayadhas, K. Sundaraj, and M. Murugappan. "**Detecting driver drowsiness based on sensors: A review**". *MDPI open access: sensors, 2012*.

14.) F. M. Sukno, S.-K. Pavani, C. Butakoff, and A. F. Frangi. "**Automatic assessment of eye blinking patterns through statistical shape models**". *In ICVS, 2009.*

15.) D. Torricelli, M. Goffredo, S. Conforto, and M. Schmid. "**An adaptive blink detector to initialize and update a view-basedremote eye gaze tracking system in a natural scenario**". *Pattern Recogn. Lett., 30(12):1144–1150, Sept. 2009.*

16.) X. Xiong and F. De la Torre. "**Supervised descent methods and its applications to face alignment**". *In Proc. CVPR, 2013.*

17.) Z. Yan, L. Hu, H. Chen, and F. Lu. "**Computer vision syndrome: A widely spreading but largely unknown epidemic among computer users**". *Computers in Human Behaviour, (24):2026–2042, 2008.*

18.) F. Yang, X. Yu, J. Huang, P. Yang, and D. Metaxas. "**Robust eyelid tracking for fatigue detection"**. *In ICIP, 2012.*

19.) S. Zafeiriou, G. Tzimiropoulos, and M. Pantic. "**The 300 videos in the wild (300-VW) facial landmark tracking in-the-wild challenge"**. *In ICCV Workshop, 2015. http://ibug.doc.ic.ac.uk/resources/300-VW/*

20.) J. Cech, V. Franc, and J. Matas. "**A 3D approach to facial landmarks: Detection, refinement, and tracking**". *In Proc. International Conference on Pattern Recognition, 2014.*

21.) www.wikipedia.org

22.) www.dlib.net

23.) http://iipimage.sourceforge.net/documentation/images/

24.) https://www.pyimagesearch.com/2015/03/23/sliding-windows-for-object-detection-with-python-and-opencv/

25.) https://www.flickr.com/photos/unavoidablegrain/8123343395

26.) https://www.learnopencv.com/histogram-of-oriented-gradients/

Appendix

Appendix – A: Face detection code

```
import sys
import dlib
detector = dlib.get_frontal_face_detector()
win = dlib.image_window()
for f in sys.argv[1:]:
    print("Processing file: {}".format(f))
    img = dlib.load_rgb_image(f)
    dets = detector(img, 1)
    print("Number of faces detected: {}".format(len(dets)))
    for i, d in enumerate(dets):
        print("Detection {}: Left: {} Top: {} Right: {} Bottom: {}".format(
            i, d.left(), d.top(), d.right(), d.bottom()))

    win.clear_overlay()
    win.set_image(img)
    win.add_overlay(dets)
    dlib.hit_enter_to_continue()
```

Appendix – B: Python code for the program

```
# To run this file make sure you installed
#       1.) OpenCV Version 3 or above
#       2.) dlib and it's dependencies
#       3.) Python 3
#       4.) Numpy
#       5.) Scipy
#
# Execute the file by the following command
#      python3  driver_drowsiness_detection.py input.mp4
# Where input.mp4 is the input file to the program.
# The output file gets stored in the same folder
# as the driver_drowsiness_detection.py file as output.mp4

import sys
import dlib
import cv2
import numpy as np
from scipy.spatial import distance

#Function to calculate the EYE ASPECT RATIO of both eyes.
def EAR(vec):
    a = distance.euclidean(vec[1], vec[5])
    b = distance.euclidean(vec[2], vec[4])
    c = distance.euclidean(vec[0], vec[3])
    ear = (a + b) / (2.0 * c)
    return ear

#Making sure there is an input file
if len(sys.argv) != 2:
```

```
    exit()

detect = dlib.get_frontal_face_detector()

#Download the landmark detector from
# http://dlib.net/files/shape_predictor_68_face_landmarks.dat.bz2
# Extract it and save it as "face_landmarks_68.dat" , in the
# same folder as the driver_drowsiness_detection.py file
predict = dlib.shape_predictor("face_landmarks_68.dat")

win = dlib.image_window()
cap = cv2.VideoCapture(sys.argv[1])
# Check if camera opened successfully
if (cap.isOpened() == False):
  print("Unable to read camera feed")
  exit()

frame_width = int(cap.get(3))
frame_height = int(cap.get(4))

out = cv2.VideoWriter('output.mp4',cv2.VideoWriter_fourcc('H','2','6','4'), 5,
(frame_width,frame_height))
font = cv2.FONT_HERSHEY_SIMPLEX
count = 0
while(True):
#Decreasing the frame rate from 30FPS to 5FPS for better performance
    ret, img = cap.read()
    ret, img = cap.read()
    ret, img = cap.read()
    ret, img = cap.read()
    ret, img = cap.read()
    ret, img = cap.read()
```

```
# Checking whether the frame is captured properly or not
    if ret == True:
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        win.clear_overlay()
        win.set_image(img)

        #Detecting the face in the rame if any

        dets = detect(img, 1)
        vec = np.empty([68, 2], dtype = int)
        status="Not Sleeping"
        for k, d in enumerate(dets):
            print("Detection {}: Left: {} Top: {} Right: {} Bottom: {}".format(
                k, d.left(), d.top(), d.right(), d.bottom()))

            # Get the landmarks/parts for the face in box d
            shape = predict(img, d)
            for b in range(68):
                vec[b][0] = shape.part(b).x
                vec[b][1] = shape.part(b).y

            # Calculating EAR by above defined function
            right_ear=EAR(vec[42:48])
            left_ear=EAR(vec[36:42])
            img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)

            # Keeping a track of total amount of time
            # for which the eyes are closed.
            # If it is more than 2 secs, we can be sure
            # the driver is dosing off
```

```
            if (right_ear+left_ear)/2 <0.2:
                if(count < 20):
                    count =count +1
                if (count > 8):
                    status= "Sleeping"

                    cv2.rectangle(img,(d.left(),d.top()),(d.right(),d.bottom()),(0,0,255),3)
                else:

                            cv2.rectangle(img,(d.left(),d.top()),(d.right(),d.bottom()),(0,255,0),3)
            else:
                if(count > 1 ):
                    count = count -1
                cv2.rectangle(img,(d.left(),d.top()),(d.right(),d.bottom()),(0,255,0),3)


            cv2.putText(img, status, (20,(frame_height-20)), font,
4,(0,255,255),6,cv2.LINE_AA)

            # Writing the file to disk.
            out.write(img)
        #cv2.imshow(status, img)
        win.add_overlay(dets)
        win.set_title(status)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    else:
        break
# Destroying the created windows.
cap.release()
out.release()
cv2.destroyAllWindows()
```