REPORT

(2020-21)

Driver Drowsiness Detection System

Institute of Engineering & Technology

Supervised By:	**Submitted By:**
Priya Agrawal	Vivek Kumar Singh(181500819**)**
(Technical Trainer)	Jitendra Parmar(181500299)
	Abhishek Gupta (181500016)

Declaration

I hereby declare that the work which is being presented in the Mini Project**"Driver DrowsinessDetection System",**in partial fulfillment of the requirements for Mini Project, is an authentic record of my own work carried under the supervision of "**Priya Agrawal**"

Name of Candidate: Vivek Kumar Singh
Roll. No.:181500819
Course: B.Tech(CSE)
Year: 2020

Name of Candidate:Jitendra Parmar
Roll. No. :181500299
Course:B.Tech(CSE)
Year: 2020

Name of Candidate: Abhishek Gupta
Roll. No.: 181500016
Course: B.Tech(CSE)
Year: 2020

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 we aim to develop a prototype drowsiness detection system. This system works by monitoring the eyes of the driver and sounding an alarm 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 alarm is sounded. The programming for this is done in OpenCV using the Haarcascade library for the detection of facial features.The report proposed the results and solutions on the limited implementation of the various techniques that are introduced in the project. Whereas the implementation of the project give the real world idea of how the system works and what changes can be done in order to improve the utility of the overall .

Furthermore, the paper states the overview of the observations made by the authors in order to help further optimization in the mentioned field to achieve the utility at a better efficiency for a safer road.

Keywords—Driver drowsiness eye detection.

Certificate

This is certify that the project entitled Driver Drowsiness Detection System submitted by Vivek Kumar Singh (181500819), Jitendra Parmar (181500299) , Abhishek Gupta (181500016) is a record of bonafide work carried out by them, in the partial fulfilment of the requirement for the award of Degree of Bachelor of Engineering (Computer Science & Engineering) at GLA University, Mathura. This work is done during year 2020, under our Guidance

Priya Agrawal

(Project Guide)

Acknowledgement

The satisfaction that accompanies that the successful completion of any task would be incomplete without the mention of people whose ceaseless cooperation made it possible, whose constant guidance and encouragement crown all efforts with success. I am very grateful to my project supervisor **Priya Agrawal**, for the guidance, inspiration and constructive suggestions that helpful me in the preparation of this project. I won't forget to also mention my Team mates; Jitendra Parmar ,Abhishek Gupta ,Vivek Kumar Singhfor their wonderful and skillful guidance in assisting in with the necessary support to ensure that project is a success. I also thank my parents and family at large for their moral and financial support in funding the project to ensure successful completion of the project.

CONTENTS

1.Introduction **09**

2. Problem definition **10**

3. Objectives **11**

4.Hardware and Software Requirements **12**

5. Technologies Used **13**

1. **OpenCv**
   1.1What is OpenCV?
   1.2What is Computer Vision?
   1.3 The origin of OpenCV
   1.4 OpenCV structure and content
   1.5Why OpenCV?

2. **Applications of Computer Vision** **16**
   2.1.Facial Recognition
   2.2 Image Search and Object Recognition

6. Classifiers Used **17**

1. haarcascade_frontalface_default.xml
2. shape_predictor_68_face_landmarks.dat

7. System Description **20**

7.1.start/stop
7.2Face Detection
7.3 Eye Detection
7.4Recognition of Eye's State
7.5 Eye State Determination
7.6 Drowsiness Detection

8.Algorithm Stages **30**

8.1Image Capture
8.2Dividing into Frames
8.3Face Detection
8.4Eye Detection
8.5State of eye

9.Implementation **32**

10. Results and Discussions **34**

11.Future Work **35**

12.Conclusions **36**

13.Coding Part **37**

14.Screenshots **41**

15. References **50**

16.Certificate **51**

Introduction

Driver fatigue is a significant factor in a large number of vehicle accidents. Recent statistics estimate that annually 1,200 deaths and 76,000 injuries can be attributed to fatigue related crashes.

Driver drowsiness resulting in reduced vehicle control is one of the major causes of road accidents. Driving performance deteriorates with increased drowsiness with resulting crashes constituting 20%-23% of all vehicle accidents .The National Highway Traffic Safety Administration (NHTSA) conservatively estimates that 100 000 reported crashes are caused by drowsy drivers each year in the U.S. alone. These crashes result in more than 1500 fatalities, 71 000 injuries, and an estimated $12.5 billion in diminished productivity and property loss.

Many efforts have been made recently to develop on-board detection of driver drowsiness. A number of approaches have been investigated and applied to characterize driver drowsiness using physiological measures, ocular measures, and performance measures. Physiological measures are based on physiological signals such as brain waves, heart rate, pulse rate, and respiration.

These measures are believed to be the most accurate for determining drowsiness. However, these techniques are intrusive since they require that electrodes be attached to the drivers. Various studies have also shown the possibility of drowsiness detection by means of ocular measures. These methods generally capture the driver's eye state using computer vision systems to indicate his/her drowsiness level. The Percentage of eyelid Closure (PERCLOS) has been used by many researchers as a valid ocular measure to indicate drowsiness.

Other measures include the duration of eye closure and blink rate. A driver state of drowsiness can also be characterized by the resulting vehicle behavior such as the lateral position, steering wheel movements, and time-to-line crossing. Although these techniques arenot intrusive, they are subject to several limitations related to the vehicle type, driver experience, and geometric characteristics and condition of the road. Among these various possibilities, the monitoring of a driver's eye state by a camera is considered to be the most promising application due to its accuracy and non-intrusiveness. The driver's symptoms can be monitored to determine the driver's drowsiness early enough to take preventive actions to avoid an accident. Though many studies have developed image-based driver alertness recognition systems using computer vision techniques, many problems still remain. First, eye detection remains a challenging problem with no inexpensive or commercial solutions. For some applications, eye feature detection can be satisfactory, but these only used frontal face images taken with controlled lighting conditions. In a car, the constantly changing lighting conditions cause dark shadows and illumination changes, such that effective techniques in stable lighting often do not work in this challenging environment.

The development of technologies for detecting or preventing drowsiness at the wheel is a major challenge in the field of accident avoidance systems. Because of the hazard that drowsiness presents on the road, methods need to be developed for counteracting its affects.

The aim of this project is to develop a prototype drowsiness detection system. The focus will be placed on designing a system that will accurately monitor the open or closed state of the driver's eyes in real-time.

By monitoring the eyes, it is believed that the symptoms of driver fatigue can be detected early enough to avoid a car accident. Detection of fatigue involves the observation of eye movements and blink patterns in a sequence of images of a face.

First we input the facial image using a webcam. Preprocessing was first performed by binarizing the image. The top and sides of the face were detected to narrow down the area where the eyes exist. Using the sides of the face, the center of the face was found which will be used as a reference when computing the left and right eyes. Moving down from the top of the face, horizontal averages of the face area were calculated. Large changes in the averages were used to define the eye area. There was little change in the horizontal average when the eyes were closed which was used to detect a blink.

For our project face and eye classifiers are required. So we used the learning objects method to create our own haarclassifier .xml files.

Around 2000 positive and 3000 negative samples are taken. Training them is a time intensive process. Finally face.xml and haarcascade-eye.xml files are created.

These xml files are directly used for object detection. It detects a sequence of objects (in our case face and eyes). Haarcascade-eye.xml is designed only for open eyes. So when eyes are closed the system doesn't detect anything. This is a blink. When a blink lasts for more than 5 frames, the driver is judged to be drowsy and an alarm is sounded.

Problem Defination

Fatigue is a safety problem that has not yet been deeply tackled by any country in the world mainly because of its nature. Fatigue, in general, is very difficult to measure or observe unlike alcohol and drugs, which have clear key indicators and tests that are available easily. Probably, the best solutions to this problem are awareness about fatigue-related accidents and promoting drivers to admit fatigue when needed. The former is hard and much more expensive to achieve, and the latter is not possible
without the former as driving for long hours is very lucrative.
In United states from 1989-1993 approximately 100,000 crashes were reported by police per year, all the crashes are related to drowsiness. Fatality Analysis Reporting System (FARS) reported that around 71,000 of all crashes were non-fatal injuries & 1,357 resulted in mortality. Many of the road accidents were not reported & verified by police, because the problem is very large. Nowadays more accident occurs in trucks and cars than vehicles due to drowsiness. Nearly 97% of crashes of vehicles happen due to drowsiness of driver. It results into loss . for eg: human loss, money loss, medical loss. The accident or crashes not only affect the internal system but also to outside world. 70% injury occurs in internal system and 30% injury happen to the external system. Environmental loss is one of the disadvantage of accident. Accidents results in human as well as non human loss.
In the real time driver fatigue detection system it is required to slow down a vehicle automatically when fatigue level crosses a certain limit. Instead of threshold drowsiness level it is suggested to design a continuous scale driver fatigue 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.

Objective

Drowsiness is a process where level of consciousness decrease due to lack of sleep or fatigue and can cause a person falls asleep. When driver is drowsy, the driver could lose control of the car so it was suddenly possible to deviate from the road and crashed into a barrier or a car.
Drowsiness detection techniques, in accordance with the parameters used for detection is divided into two sections i.e. intrusive method and a non-intrusive method. The main difference of these two methods is that the intrusive method.
An instrument connected to the driver and then the value of the instrument are recorded and checked. But intrusive approach has high accuracy, which is proportional to driver discomfort, so this method is rarely used.

Drowsiness detection is a safety technology that can prevent accidents that are caused by drivers who fell asleep while driving .The objective of this Python project is to build a drowsiness detection system that will detect that a person's eyes are closed for a few seconds. This system will alert the driver when drowsiness is detected.

Driver drowsiness is a significant factor in the increasing number of accidents on today's roads and has been extensively accepted. This proof has been verified by many researchers that have demonstrated ties between driver drowsiness and road accidents. Although it is hard to decide the exact number of accidents due to drowsiness, it is much likely to be underestimated.

The above statement shows the significance of a research with the objective of reducing the dangers of accidents anticipated to drowsiness. So far, researchers have tried to model the behavior by creating links between drowsiness and certain indications related to the vehicle and to the driver .

This paper represents a new way towards safety as well as security of automobiles. We are using the concept of eye recognition system, Drowsiness detection . Nowadays driver fatigue related crashes has increased. The main objective of our project is to develop nonintrusive system which will detect the fatigue or drowsiness of driver and will issue a warning with the help of alarm. As most of the accidents are caused due to drowsiness so this project will help to decrease the crashes or accidents. In this project we will detect the eye blinking with the help of webcam. If the eyes of the person are closed for more interval of time then this will result into the warning in the form of sound.

Hardware and Software Requirements :

a) Hardware:

- Minimum 2GB RAM
- i3 Processor
- Laptop with basic Hardware
- Webcam

b) Software:

- Pycharm
- Anconda
- Operating System(Window)
- Programming Language
  - Python with OpenCv

OpenCV

OpenCV [OpenCV] is an open source computer vision library.detection. OpenCV is coded with optimized C and can take work with multicore processors. If we desire more automatic optimization using Intel architectures.

One of OpenCVs goals is to provide a simple-to-use computer vision infrastructure which helps people to build highly sophisticated vision applications fast. The OpenCV library, containing over 500 functions, spans many areas in vision. Because computer vision and machine learning oft en goes hand-in-hand.

This sub library is focused on statistical pattern recognition and clustering. The MLL is very useful for the vision functions that are the basis of OpenCV's usefulness, but is general enough to be used for any machine learning problem.

What Is Computer Vision:

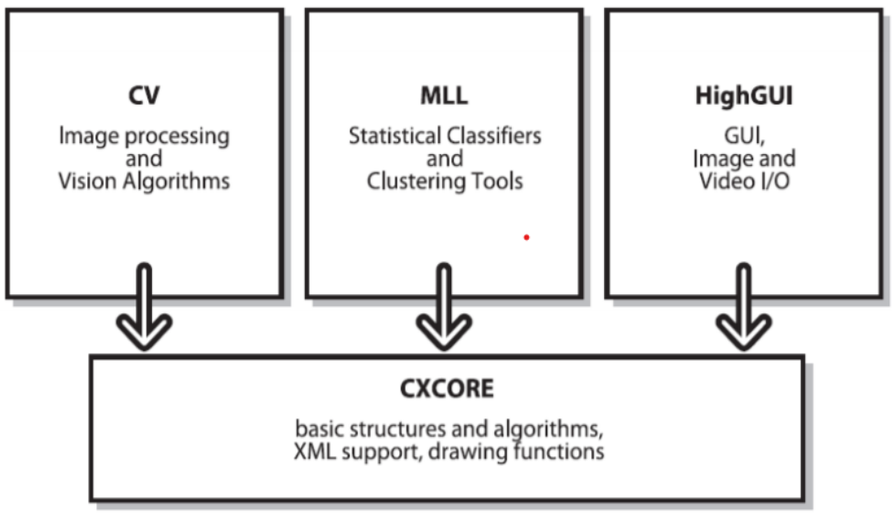
Computer vision is the transforming of data from a still, or video camera into either a representation or a new decision. All such transformations are performed to achieve a particular goal. A computer obtains a grid of numbers from a camera or from the disk, and that's that. Usually, there is no built in pattern recognition or automatic control of focus and aperture, no cross-associations with years of experience. For the most part, vision systems are still fairly .

The Origin of OpenCV :

OpenCV came out of an Intel Research initiative meant to advance CPU-intensive applications. Toward this end, Intel launched various projects that included real-time ray tracing and also 3D display walls. One of the programmers working for Intel at the time was visiting universities. He noticed that a few top university groups, like the MIT Media Lab, used to have well-developed as well as internally open computer vision infrastructures—code that was passed from one student to another and which gave each subsequent student a valuable foundation while developing his own vision application. Instead of having to reinvent the basic functions from beginning, a new student may start by adding to that which came before.

OpenCV Structure and Content :

OpenCV can be broadly structured into five primary components, four of which are shown in the figure. The CV component contains mainly the basic image processing and higher-level computer vision algorithms; MLL the machine learning library includes many statistical classifiers as well as clustering tools. HighGUI component contains I/O routines with functions for storing, loading video & images, while CXCore contains all the basic data structures and content.



WhyOpenCV:

Specific :

OpenCV was designed for image processing. Every function and data structure has been designed with an Image Processing application in mind. You can get almost everything in the world by means of toolboxes. It may be financial toolboxes or specialized DNA toolboxes.

Speedy :

Matlab is just way too slow. Matlab itself was built upon Java. Also Java was built upon C. So when we run a Matlab program, our computer gets busy trying to interpret and compile all that complicated Matlab code. Then it is turned into Java, and finally executes the code.

Efficient :

With OpenCV, we can get away with as little as 10mb RAM for a real-time application. Although with today's computers, the RAM factor isn't a big thing to be worried about. However, our drowsiness detection system is to be used inside a car in a way that is non-intrusive and small; so a low processing requirement is vital.
Thus we can see how OpenCV is a better choice than Matlab for a real-time drowsiness detection system.

Applications of Computer Vision

There are some practical applications of computer vision:

1.Facial Recognition - This is a very important application of computer vision where electronics use facial recognition technology to basically validate the identity of the user.

2. Image Search and Object Recognition — Now we could search objects in an image using image search. A very good example is google lens where we could search a particular object within the image by clicking the photo of the image and the computer vision algorithm will search through the catalogue of images and extract information out of the image

Classifiers

1.haarcascade_frontalface_default.xml :

Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images.

most of them are irrelevant. For example, consider the image below. Top row shows two good features. The first feature selected seems to focus on the property that the region of the eyes is often darker than the region of the nose and cheeks. The second feature selected relies on the property that the eyes are darker than the bridge of the nose. But the same windows applying on cheeks or any other place is irrelevant. So how do we select the best features out of 160000+ features? It is achieved by **Adaboost**.

For this, we apply each and every feature on all the training images. For each feature, it finds the best threshold which will classify the faces to positive and negative. But obviously, there will be errors or misclassifications. We select the features with minimum error rate, which means they are the features that best classifies the face and non-face images. (The process is not as simple as this. Each image is given an equal weight in the beginning. After each classification, weights of misclassified images are increased. Then again same process is done. New error rates are calculated. Also new weights. The process is continued until required accuracy or error rate is achieved or required number of features are found).

Final classifier is a weighted sum of these weak classifiers. It is called weak because it alone can’t classify the image, but together with others forms a strong classifier. The paper says even 200 features provide detection with 95% accuracy. Their final setup had around 6000 features. (Imagine a reduction from 160000+ features to 6000 features. That is a big gain).

So now you take an image. Take each 24x24 window. Apply 6000 features to it. Check if it is face or not. Wow.. Wow.. Isn’t it a little inefficient and time consuming? Yes, it is. Authors have a good solution for that.

In an image, most of the image region is non-face region. So it is a better idea to have a simple method to check if a window is not a face region. If it is not, discard it in a single shot. Don’t process it again. Instead focus on region where there can be a face. This way, we can find more time to check a possible face region.

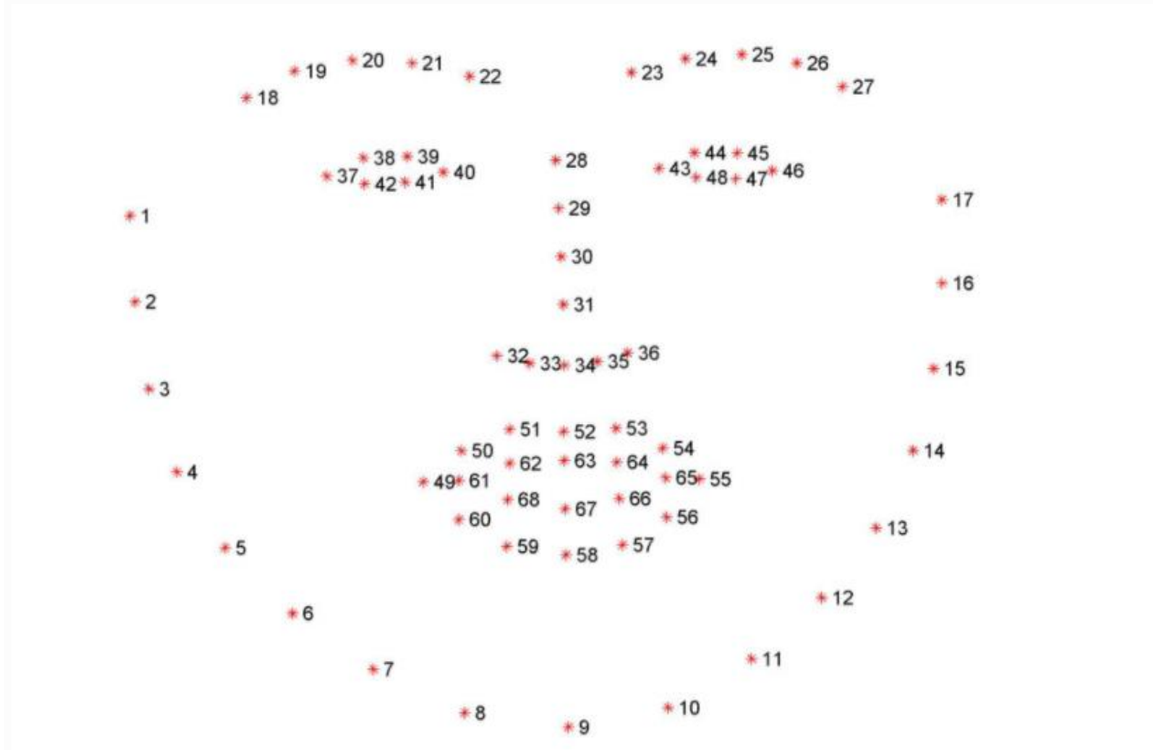
For this they introduced the concept of **Cascade of Classifiers**. Instead of applying all the 6000 features on a window, group the features into different stages of classifiers and apply one-by-one. (Normally first few stages will contain very less number of features). If a window fails the first stage, discard it. We don't consider remaining features on it. If it passes, apply the second stage of features and continue the process. The window which passes all stages is a face region. How is the plan !!!

Authors' detector had 6000+ features with 38 stages with 1, 10, 25, 25 and 50 features in first five stages. (Two features in the above image is actually obtained as the best two features from Adaboost). According to authors, on an average, 10 features out of 6000+ are evaluated per sub-window.

So this is a simple intuitive explanation of how Viola-Jones face detection works. Read paper for more details or check out the references in Additional Resources section.

2.shape_predictor_68_face_landmarks.dat:

It's a landmark's facial detector with pre-trained models, the dlib is used to estimate the location of 68 coordinates (x, y) that map the facial points on a person's face like image below.



These annotations are part of the 68 point **iBUG 300-W dataset** which the dlib facial landmark predictor was trained on.

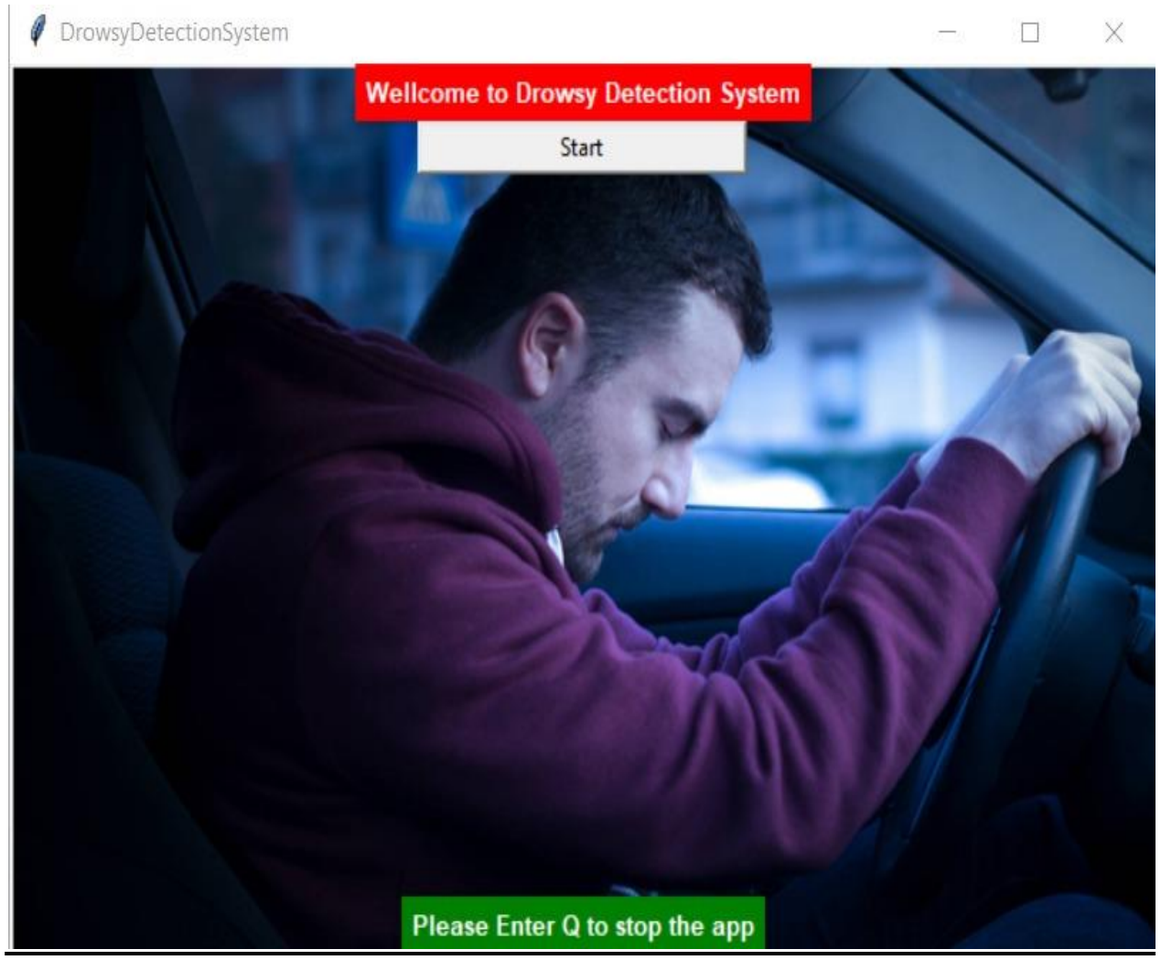
It’s important to note that other flavors of facial landmark detectors exist, including the 194 point model that can be trained on the **HELEN dataset**.

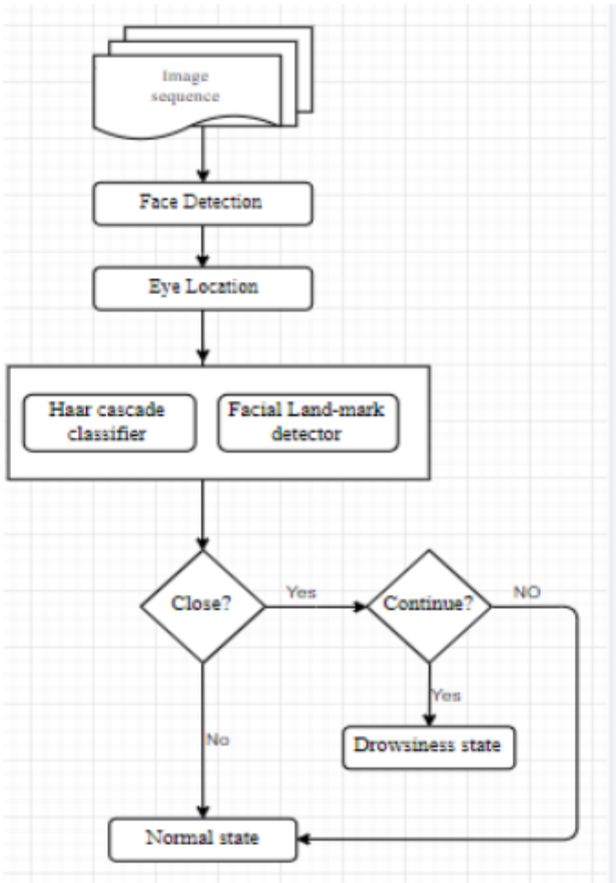
Regardless of which dataset is used, the same dlib framework can be leveraged to train a shape predictor on the input training data — this is useful if you would like to train facial landmark detectors or custom shape predictors of your own.

System Description

1.Start/stop :

The first step in the system is the simple where the user/driver must have to start the the system, the same is shown in the fig .1 as follows





Block diag.

2.Face Detection:

For the face Detection it uses Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images. Here we will work with face detection. Initially, the algorithm needs a lot of positive images (images of faces) and negative images (images without faces) to train the classifier. Then we need to extract features from it. For this, Haar features shown in the below image are used. They are just like our convolutional kernel. Each feature is a single value obtained by subtracting sum of pixels under the white rectangle from sum of pixels under the black rectangle

A cascaded Adaboost classifier with the Haar-like features is exploited to find out the face region. First, the compensated image is segmented into numbers of rectangle areas, at any position and scale within the original image. Due to the difference of facial feature, Haar-like feature is efficient for real-time face detection. These can be calculated according to the difference of sum of pixel values within rectangle areas. The features can be represented by the different composition of the black region and white region. A cascaded Adaboost classifier is a strong classifier which is a combination of several weak classifiers. Each weak classifier is trained by Adaboost algorithm. If a candidate sample passes through the cascaded Adaboost classifier, the face region can be found. Almost all of face samples can pass through and nonface samples can be rejected

3.Eye detection:

In the system we have used 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

Facial landmarks have been successfully applied to face alignment, head pose estimation, face swapping, blink detection and much more. In the context of facial landmarks, our goal is detecting important facial structures on the face using shape prediction methods. Detecting facial landmarks is therefore a twostep process:

- Localize the face in the image.
- Detect the key facial structures on the face ROI.

Localize the face in the image: The face image is localized by Haar feature-based cascade classifiers which was discussed in the first step of our algorithm i.e. face detection. Detect the key facial structures on the face ROI: There are a variety of facial landmark detectors, but all methods essentially try to localize and label the following facial regions:

• Mouth

• Right eyebrow

• Left eyebrow

• Right eye

• Left eye

• Nose

The facial landmark detector included in the dlib library is an implementation of the One Millisecond Face Alignment with an Ensemble of Regression Trees paper by Kazemi and Sullivan (2014). This method starts by using:

1. A training set of labeled facial landmarks on an image. These images are manually labeled, specifying specific (x, y)-coordinates of regions surrounding each facial structure.

2. Priors, of more specifically, the probability on distance between pairs of input pixels. The pre-trained facial landmark detector inside the dlib library is used to estimate the location of 68 (x, y)-coordinates that map to facial structures on the face.

We can detect and access both the eye region by the following facial landmark index show below

• The right eye using [36, 42].

• The left eye with [42, 48].

These annotations are part of the 68 point iBUG 300-W dataset which the dlib facial landmark predictor was trained on. It's important to note that other flavors of facial landmark detectors exist, including the 194 point model that can be trained on the HELEN dataset. Regardless of which dataset is used, the same dlib framework can be leveraged to train a shape predictor on the input training data.

4.Recognition of Eye's State:

The eye area can be estimated from optical flow, by sparse tracking or by frame-to-frame intensity differencing and adaptive thresholding. And Finally, a decision is made whether the eyes are or are not covered by eyelids. A different approach is to infer the state of the eye opening from a single image, as e.g. by correlation matching with open and closed eye templates, a heuristic horizontal or vertical image intensity projection over the eye region, a parametric model fitting to find the eyelids, or active shape models.

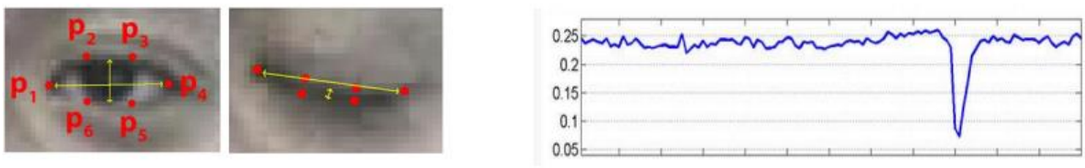
A major drawback of the previous approaches is that they usually implicitly impose too strong requirements on the setup, in the sense of a relative face-camera pose (head orientation), image resolution, illumination, motion dynamics, etc. Especially the heuristic methods that use raw image intensity are likely to be very sensitive despite their real-time performance. Therefore, we propose a simple but efficient algorithm to detect eye blinks by using a recent facial landmark detector. A single scalar quantity that reflects a level of the eye opening is derived from the landmarks. Finally, having a per-frame sequence of the eye-opening estimates, the eye blinks are found by an SVM classifier that is trained on examples of blinking and nonblinking patterns.

Eye Aspected Ratio Calculation:

For every video frame, the eye landmarks are detected. The eye aspect ratio (EAR) between height and width of the eye is computed.

$$EAR = \frac{\|p2 - p6\| + \|p3 - p5\|}{2\|p1 - p4\|} \qquad \text{---------- (1)}$$

where p1, . . ., p6 are the 2D landmark locations, depicted in Fig. 1. The EAR is mostly constant when an eye is open and is getting close to zero while closing an eye. It is partially person and head pose insensitive. Aspect ratio of the open eye has a small variance among individuals, and it is fully invariant to a uniform scaling of the image and in-plane rotation of the face. Since eye blinking is performed by both eyes synchronously, the EAR of both eyes is averaged.



Open and closed eyes with landmarks p(i) automatically detected. The eye aspect ratio EAR in Eq. (1) plotted for several frames of a video sequence.

5. 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”.

6.Drowsiness Detection :

Drowsiness is defined as a decreased level of awareness portrayed by sleepiness and trouble in staying alarm but the person awakes with simple excitement by stimuli. It might be caused by an absence of rest, medicine, substance misuse, or a cerebral issue. It is mostly the result of fatigue which can be both mental and physical. Physical fatigue, or muscle weariness, is the temporary physical failure of a muscle to perform ideally. Mental fatigue is a temporary failure to keep up ideal psychological execution. The onset of mental exhaustion amid any intellectual action is progressive, and relies on an individual's psychological capacity, furthermore upon different elements, for example, lack of sleep and general well-being. Mental exhaustion has additionally been appeared to diminish physical performance. It can show as sleepiness, dormancy, or coordinated consideration weakness. In the past years according to available data driver sleepiness has gotten to be one of the real reasons for street mishaps prompting demise and extreme physical injuries and loss of economy. A driver who falls asleep is in an edge of losing control over the vehicle prompting crash with other vehicle or stationary bodies. Keeping in mind to stop or reduce the number of accidents to a great extent the condition of sleepiness of the driver should be observed continuously.

Measures for detection of Drowsiness

The study states that the reason for a mishap can be categorized as one of the accompanying primary classes: (1) human, (2) vehicular, and (3) surrounding factor. The driver's error represented 91% of the accidents. The other two classes of causative elements were referred to as 4% for the type of vehicle used and 5% for surrounding factors. Several measures are available for the measurement of drowsiness which includes the following:

1. Vehicle based measures.
2. Physiological measures.
3. Behavioral measures

1. **Vehicle based measures.**

Vehicle-based measures survey path position, which monitors the vehicle's position as it identifies with path markings, to determine driver weakness, and accumulate steering wheel movement information to characterize the fatigue from low level to high level. In many research project, researchers have used this method to detect fatigue, highlighting the continuous nature of this non-intrusive and cost-effective monitoring technique. This is done by:

1. Sudden deviation of vehicle from lane position.
2. Sudden movement of steering wheels.
3. Pressure on acceleration paddles.

For each measures threshold values are decided which when crossed indicated that driver is drowsy.

Advantages:

1. It is noninvasive in nature.
2. Provides almost accurate result.

Disadvantages:

1. Vehicle based measures mostly affected by the geometry of road which sometimes unnecessarily activates the alarming system.

2. The driving style of the current driver needs to be learned and modeled for the system to be efficient.

3. The condition like micro sleeping which mostly happens in straight highways cannot be detected.

2. Physiological measures

Physiological measures are the objective measures of the physical changes that occur in our body because of fatigue. These physiological changes can be simply measure by their respective instruments as follows:

ECG (electro cardiogram)

EMG (electromyogram) EOG (electro occulogram)

EEG (electroencephalogram)

Monitoring Heart Rate:

An ECG sensor can be installed in the steering wheel of a car to monitor a driver's pulse, which gives a sign of the driver's level of fatigue indirectly giving the state of drowsiness. Additionally the ECG sensor can be introduced in the back of the seat

Monitoring Brain Waves: Special caps embedded with electrodes measures the brain waves to identify fatigue in drivers and report results in real time. Then each brain waves can be classified accordingly to identify drowsiness

Monitoring muscle fatigue:

As muscle fatigue is directly related to drowsiness. We know during fatigue the pressure on the steering wheel reduces and response of several muscle drastically reduces hence it can be measured by installation of pressure sensors at steering wheel or by measuring the muscle response with applied stimuli to detect the fatigue.

Monitoring eye movements:

Invasive measurement of eye movement and eye closure can be done by using electro occulogram but it will be very uncomfortable for the driver to deal with. Though this method gives the most accurate results regarding drowsiness. But it requires placement of several electrodes to be placed on head, chest and face which is not at all a convenient and annoying for a driver. Also they need to be very carefully placed on respective places for perfect result.

3.Behavioral measures

Certain behavioral changes take place during drowsing like

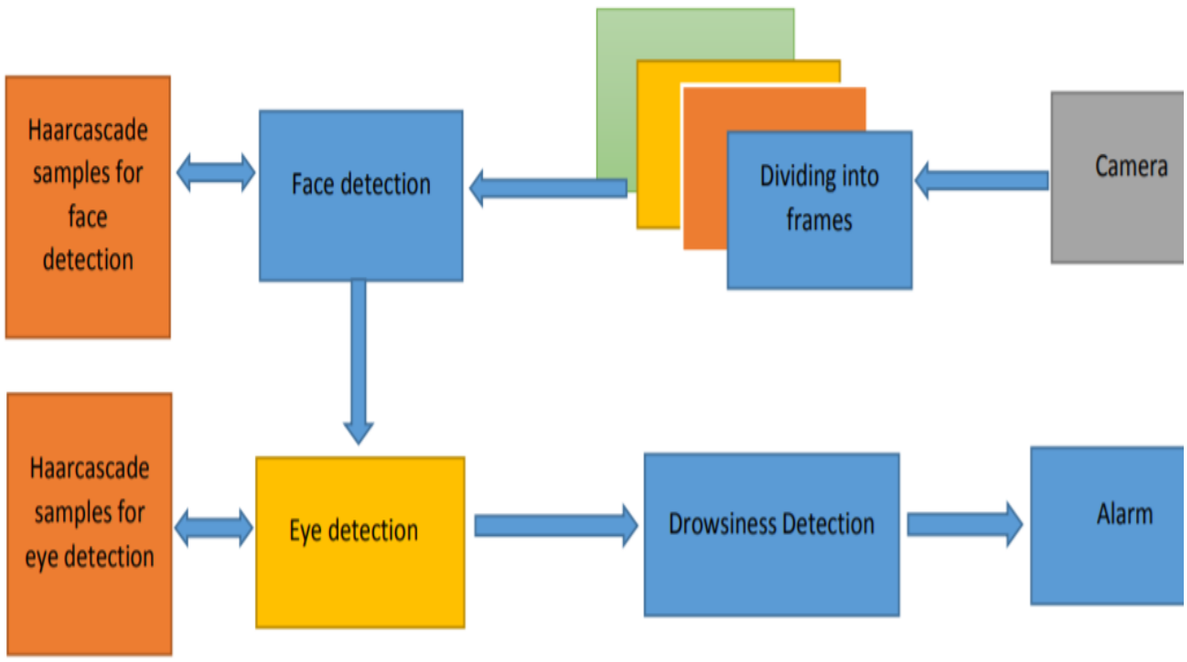
1. Yawning

2. Amount of eye closure

3. Eye blinking

4. Head position

Flowchart of the proposed system :



Algorithm Stages

Image Capture:

Utilizing a web camera introduced inside the automobile we can get the picture of the driver. Despite the fact that the camera creates a video clip, we have to apply the developed algorithm on each edge of the video stream. This paper is only focused on the applying the proposed mechanism only on single frame. The used camera is a low cost web camera with a frame rate of 30 fps in VGA mode. Logitech Camera is used for this process is shown in figure .

Dividing into Frames:

We are dealing with real time situation where video is recorded and has to be processed. But the processing or application of algorithm can be done only on an image. Hence the captured video has to be divided into frames for analyzing.

Face Detection:

In this stage we detect the region containing the face of the driver. A specified algorithm is for detection of face in every frame. By face detection we means that locating the face in a frame or in other words finding location of facial characters through a type of technology with the use of computer. The frame may be any random frame. Only facial related structures or features are detected and all others types of objects like buildings, tree, bodies are ignored.

Eye Detection:

After successful detection of face eye needs to be detected for further processing. In our method eye is the decision parameter for finding the state of driver. Though detection of eye may be easier to locate, but it's really quite complicated. At this point it performs the detection of eye in the required particular region with the use of detection of several features. Generally Eigen approach is used for this process. It is a time taking process. When eye detection is done then the result is matched with the reference or threshold value for deciding the state of the driver.

State of eye:

In this stage, we find the actual state of the eye that if it is closed or open or semi closed or open. The identification of eyes status is most important requirement. It is achieved by an algorithm which will be clarified in the later parts. We channelize a warning message if we obtain that the eyes are in open state or semi open state up to a particular threshold value. If the system detects that the eyes are open then the steps are repeated again and again until it finds a closed eye.

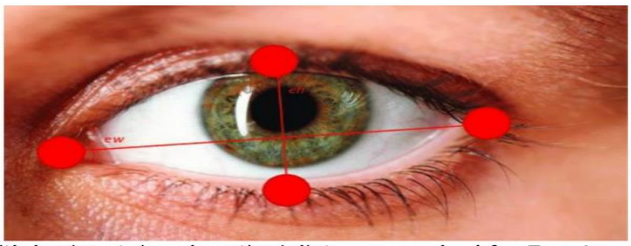
Implementation

In our program we used Dlib, a pre-trained program trained on thedataset to detect human faces using the pre-defined 68 landmarks.

After passing our video feed to the dlib frame by frame, we are able to detect left eye and right eye features of the face

Now, we drew contours around it using OpenCV.

Using Scipy's Euclidean function, we calculated sum of both eyes' aspect ratio which is the sum of 2 distinct vertical distances between the eyelids divided by its horizontal distance.



Eyes with horizontal and vertical distance marked for Eye Aspect Ratio calculation.

Now we check if the aspect ratio value is less than 0.25 (0.25 waschosen as a base case after some tests). If it is less an alarm is sounded and user is warned

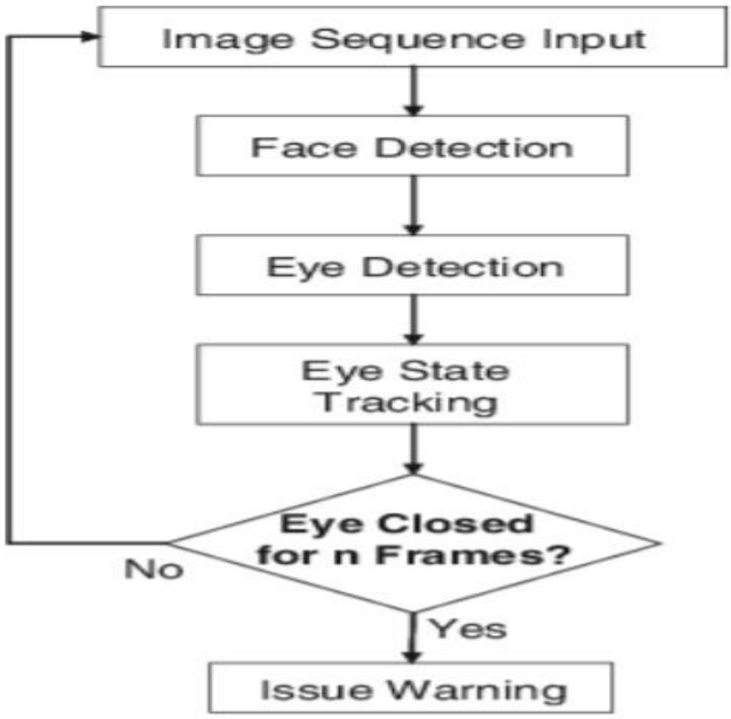


Image Sequence Input
Face Detection
Eye Detection
Eye State Tracking
Eye Closed for n Frames?
No
Yes
Issue Warning

Results and Discussions

Implementation of drowsiness detection with Python and OpenCV was done which includes the following steps: Successful runtime capturing of video with camera. Captured video was divided into frames and each frame were analyzed. Successful detection of face followed by detection of eye. If closure of eye for successive frames were detected, then it is classified as drowsy condition else it is regarded as normal blink and the loop of capturing image and analyzing the state of driver is carried out again and again. In this implementation during the drowsy state the eye is not surrounded by circle or it is not detected, and corresponding message is shown.

Future Work

Our model is designed for detection of drowsy state of eye and give and alert signal or warning in the form of audio alarm. But the response of driver after being warned may not be enough to stop causing the accident meaning that if the driver is slow in responding towards the warning signal then accident may occur. Hence to avoid this we can design and fit a motor driven system and synchronize it with the warning signal so that the vehicle will slow down after getting the warning signal automatically.

Conclusions

A real-time eye blink detection algorithm was presented. We quantitatively demonstrated that Haar feature-based cascade classifiers and regression-based facial landmark detectors are precise enough to reliably estimate the positive images of face and a level of eye openness. While they are robust to low image quality (low image resolution in a large extent) and in-thewild.

Limitations:

Use of spectacles: In case the user uses spectacle then it is difficult to detect the state of the eye. As it hugely depends on light hence reflection of spectacles may give the output for a closed eye as opened eye. Hence for this purpose the closeness of eye to the camera is required to avoid light.

Multiple face problem: If multiple face arises in the window then the camera may detect more number of faces undesired output may appear. Because of different condition of different faces. So, we need to make sure that only the driver face come within the range of the camera. Also, the speed of detection reduces because of operation on multiple faces.

Coading Part

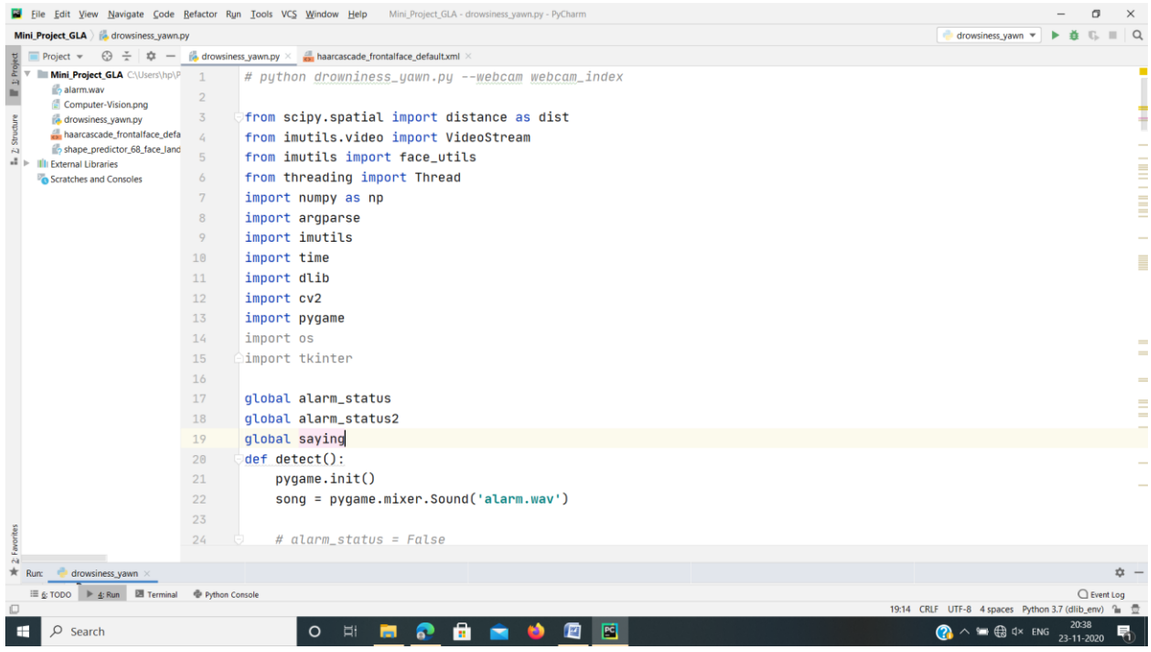
```
# python drowniness_yawn.py --webcam webcam_index

from scipy.spatial import distance as dist
from imutils.video import VideoStream
from imutils import face_utils
from threading import Thread
import numpy as np
import argparse
import imutils
import time
import dlib
import cv2
import pygame
import os
import tkinter

global alarm_status
global alarm_status2
global saying
def detect():
    pygame.init()
    song = pygame.mixer.Sound('alarm.wav')

    # alarm_status = False
    # alarm_status2 = False
    # saying = False

    def alarm():


        while alarm_status:
            print("Drowsy")
            song.play()

        if alarm_status2:
            print('yawning')
            saying = True
            song.play()

    def eye_aspect_ratio(eye):
        A = dist.euclidean(eye[1], eye[5])
        B = dist.euclidean(eye[2], eye[4])

        C = dist.euclidean(eye[0], eye[3])

        ear = (A + B) / (2.0 * C)

        return ear

    def final_ear(shape):
        (lStart, lEnd) =
face_utils.FACIAL_LANDMARKS_IDXS["left_eye"]
        (rStart, rEnd) =
face_utils.FACIAL_LANDMARKS_IDXS["right_eye"]
```

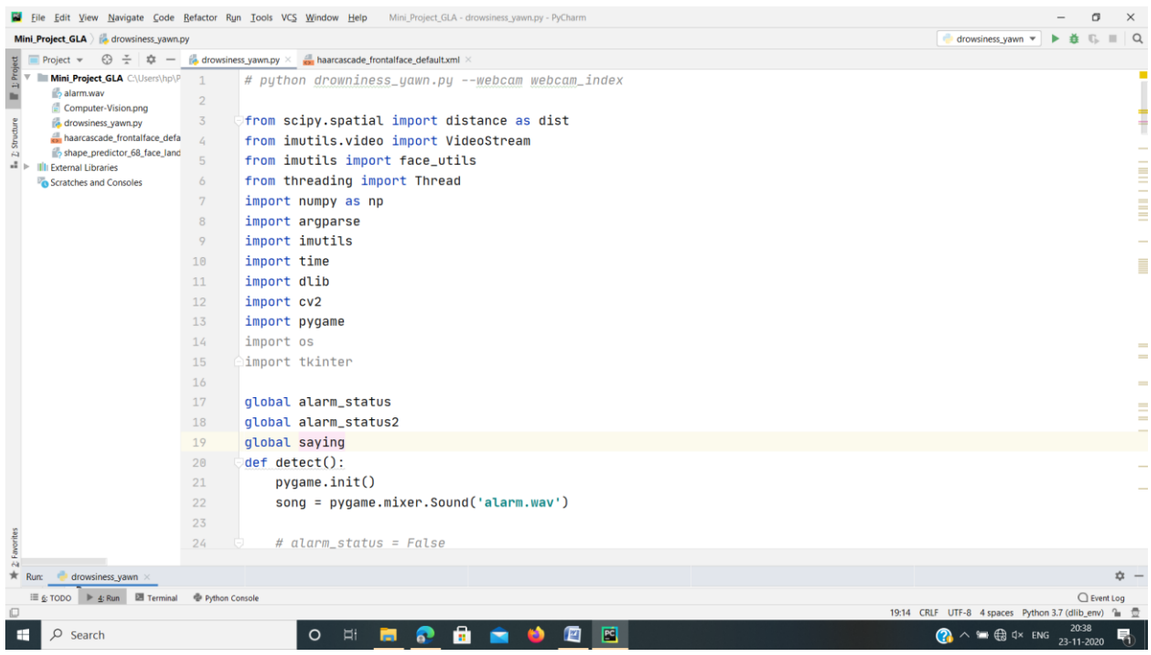
```
        leftEye = shape[lStart:lEnd]
        rightEye = shape[rStart:rEnd]

        leftEAR = eye_aspect_ratio(leftEye)
        rightEAR = eye_aspect_ratio(rightEye)

        ear = (leftEAR + rightEAR) / 2.0
        return (ear, leftEye, rightEye)

    def lip_distance(shape):
        top_lip = shape[50:53]
        top_lip = np.concatenate((top_lip, shape[61:64]))

        low_lip = shape[56:59]
        low_lip = np.concatenate((low_lip, shape[65:68]))

        top_mean = np.mean(top_lip, axis=0)
        low_mean = np.mean(low_lip, axis=0)

        distance = abs(top_mean[1] - low_mean[1])
        return distance

    ap = argparse.ArgumentParser()
    ap.add_argument("-w", "--webcam", type=int, default=0,
                    help="index of webcam on system")
    args = vars(ap.parse_args())

    EYE_AR_THRESH = 0.3
    EYE_AR_CONSEC_FRAMES = 30
    YAWN_THRESH = 20
    alarm_status = False
    alarm_status2 = False
    saying = False
    COUNTER = 0

    print("-> Loading the predictor and detector...")
    detector = dlib.get_frontal_face_detector()
    # detector =
cv2.CascadeClassifier("haarcascade_frontalface_default.xml"
)  # Faster but less accurate then dlib
    predictor =
dlib.shape_predictor('shape_predictor_68_face_landmarks.dat
')

    print("-> Starting Video Stream")
    vs = VideoStream(src=args["webcam"]).start()
    # vs= VideoStream(usePiCamera=True).start()       //For
Raspberry Pi
    time.sleep(1.0)

    while True:

        frame = vs.read()
        frame = imutils.resize(frame, width=450)
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        rects = detector(gray, 0)
```

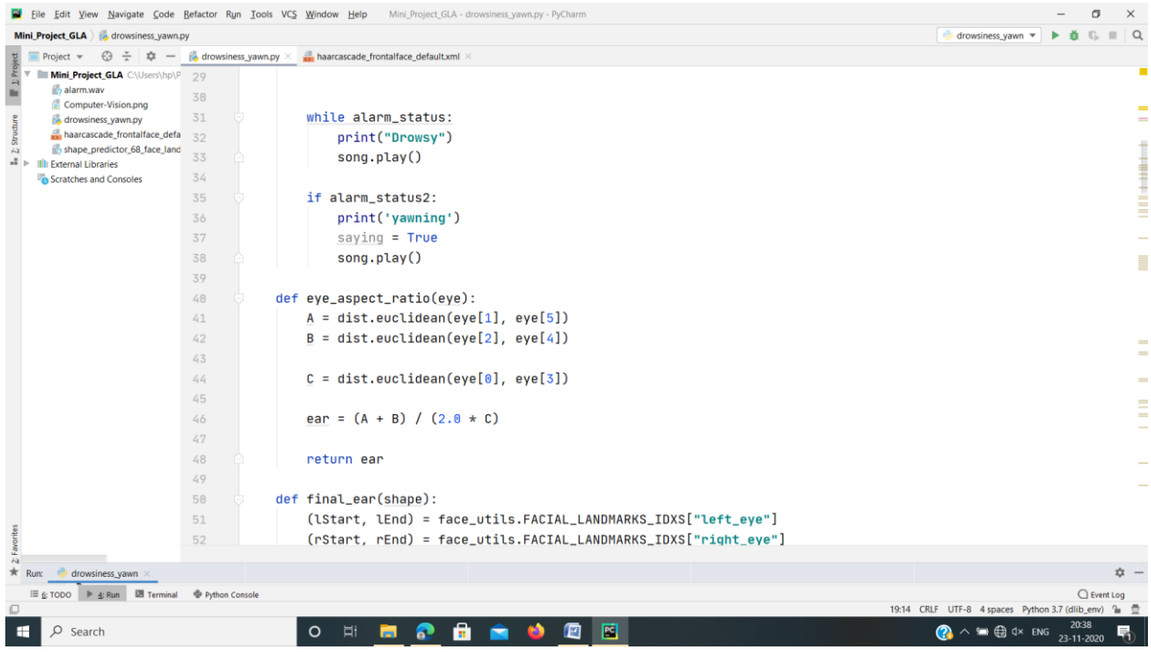
```
        for rect in rects:
            # for (x, y, w, h) in rects:
            # rect = dlib.rectangle(int(x), int(y), int(x +
w), int(y + h))

            shape = predictor(gray, rect)
            shape = face_utils.shape_to_np(shape)

            eye = final_ear(shape)
            ear = eye[0]
            leftEye = eye[1]
            rightEye = eye[2]

            distance = lip_distance(shape)

            leftEyeHull = cv2.convexHull(leftEye)
            rightEyeHull = cv2.convexHull(rightEye)
 cv2.drawContours(frame, [leftEyeHull], -1, (0, 255, 0), 1)
cv2.drawContours(frame, [rightEyeHull], -1, (0, 255, 0), 1)

            lip = shape[48:60]
         cv2.drawContours(frame, [lip], -1, (0, 255, 0), 1)

            if ear < EYE_AR_THRESH:
                COUNTER += 1

                if COUNTER >= EYE_AR_CONSEC_FRAMES:
                    if alarm_status == False:
                        alarm_status = True
                        t = Thread(target=alarm)
                        t.deamon = True
                        t.start()

                    cv2.putText(frame, "DROWSINESS ALERT!",
(10, 30),
                                cv2.FONT_HERSHEY_SIMPLEX,
0.7, (0, 0, 255), 2)

            else:
                COUNTER = 0
                alarm_status = False

            if (distance > YAWN_THRESH):
                cv2.putText(frame, "Yawn Alert", (10, 30),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(0, 0, 255), 2)
                if alarm_status2 == False and saying ==
False:
                    alarm_status2 = True
                    t = Thread(target=alarm)
                    t.deamon = True
                    t.start()
            else:
                alarm_status2 = False

            cv2.putText(frame, "EAR: {:.2f}".format(ear),
(300, 30),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0,
```

```
0, 255), 2)
            cv2.putText(frame, "YAWN:
{:.2f}".format(distance), (300, 60),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0,
0, 255), 2)

        cv2.imshow("Frame", frame)
        key = cv2.waitKey(1) & 0xFF
        if key == ord("q"):
            break

    cv2.destroyAllWindows()
    vs.stop()


from tkinter import *

root = Tk()
root.geometry("644x434")
root.minsize(644, 434)
root.maxsize(644, 434)
root.title("DrowsyDetectionSystem")
photo = PhotoImage(file="Computer-Vision.png")
photoLabel = Label(image=photo)
photoLabel.grid()
photoLabel.place(x=0,y=0)


label1 = Label(text="Wellcome to Drowsy Detection System",
bg="red", fg="white", padx=4, pady=4,
               font=("comicsansms", 10, "bold"))
label1.pack()


def run():
    detect()


btn = Button(root, text="Start", width=25, command=run)
btn.pack()
f1 = Frame(root, bg="grey")
f1.pack(side=BOTTOM)
label2 = Label(f1, text="Please Enter Q to stop the app",
bg="green", fg="white", padx=4, pady=4,
               font=("comicsansms", 10, "bold"))
label2.pack()

root.mainloop()
```

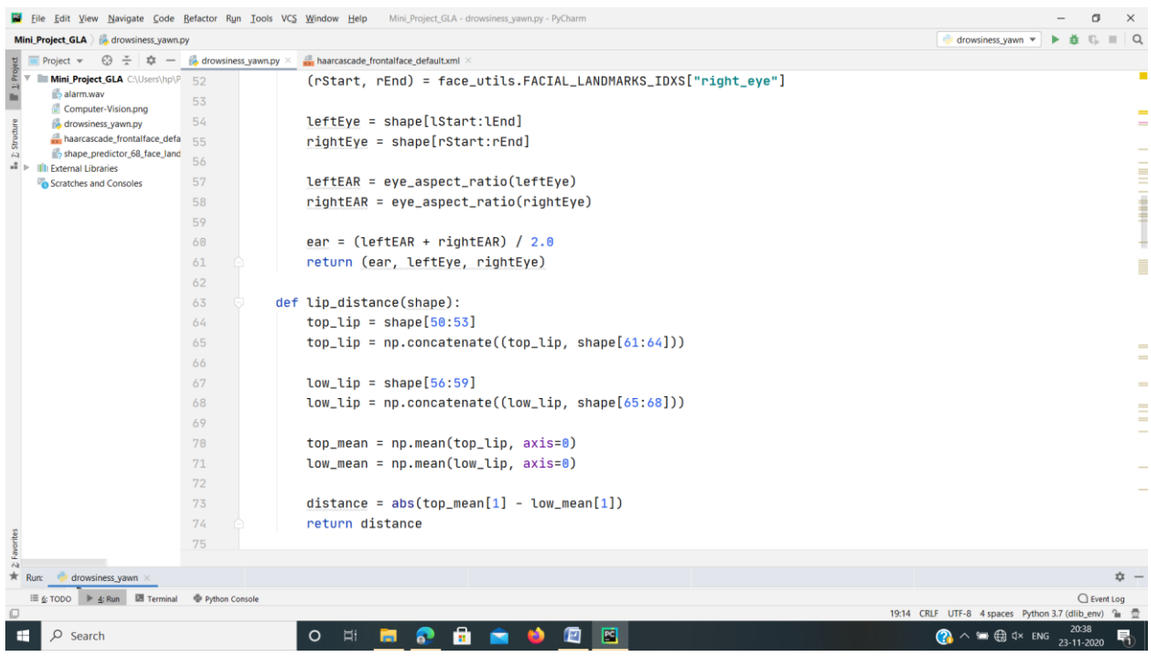
SCREENSHOTS



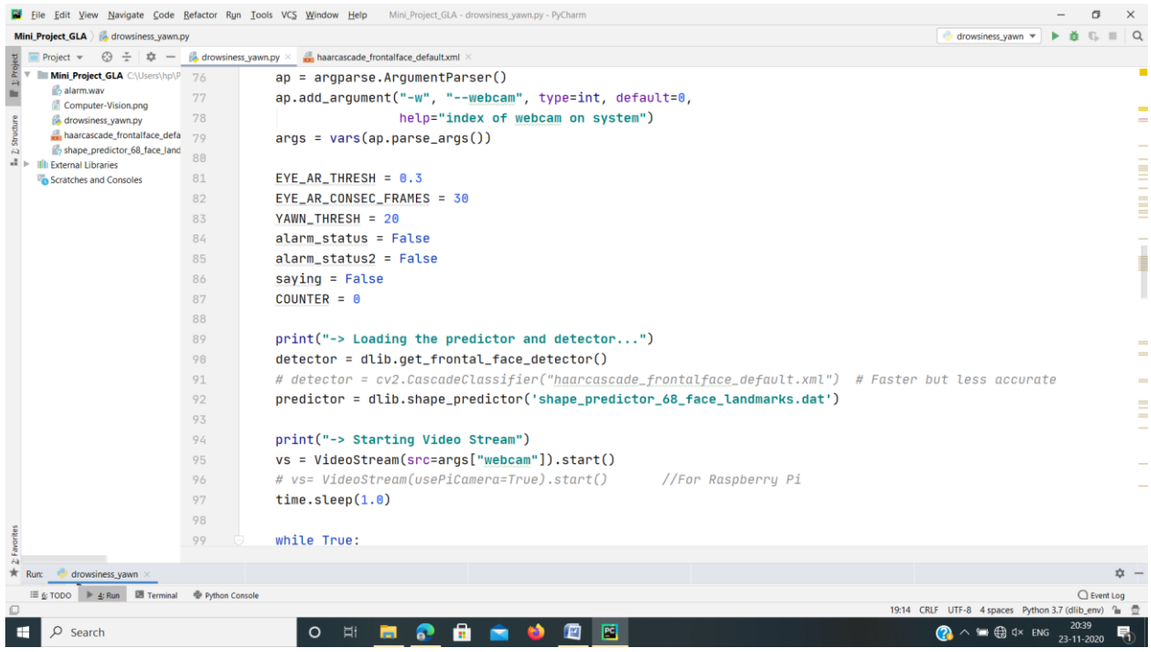




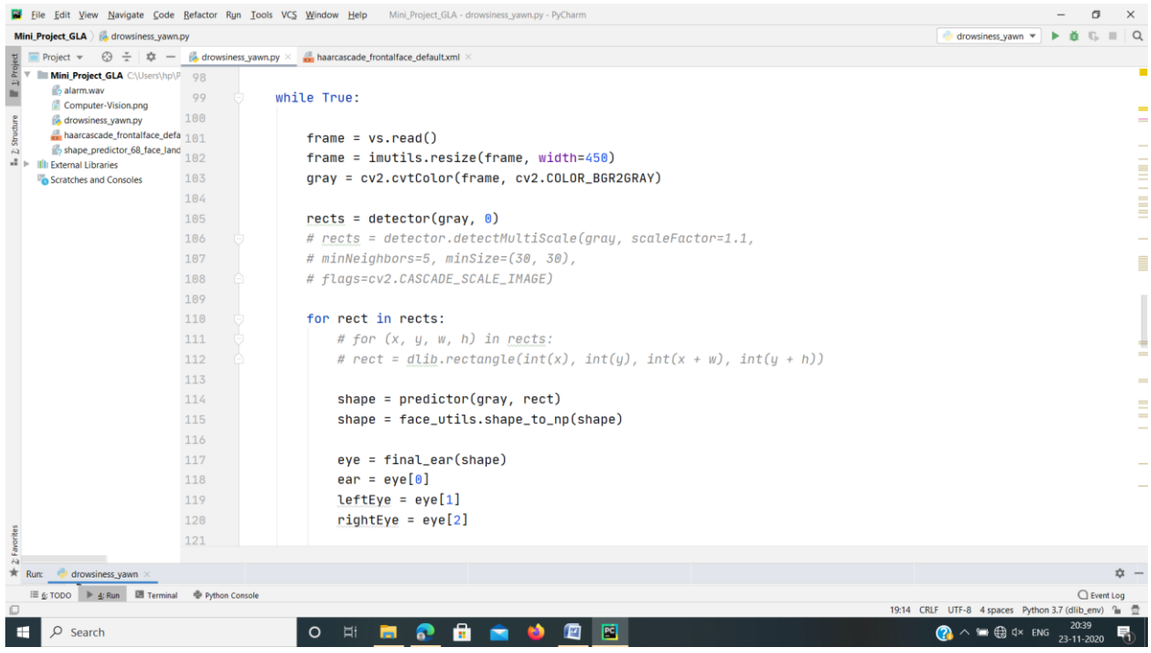
```
        while alarm_status:
            print("Drowsy")
            song.play()

        if alarm_status2:
            print('yawning')
            saying = True
            song.play()

    def eye_aspect_ratio(eye):
        A = dist.euclidean(eye[1], eye[5])
        B = dist.euclidean(eye[2], eye[4])

        C = dist.euclidean(eye[0], eye[3])

        ear = (A + B) / (2.0 * C)

        return ear

    def final_ear(shape):
        (lStart, lEnd) = face_utils.FACIAL_LANDMARKS_IDXS["left_eye"]
        (rStart, rEnd) = face_utils.FACIAL_LANDMARKS_IDXS["right_eye"]
```



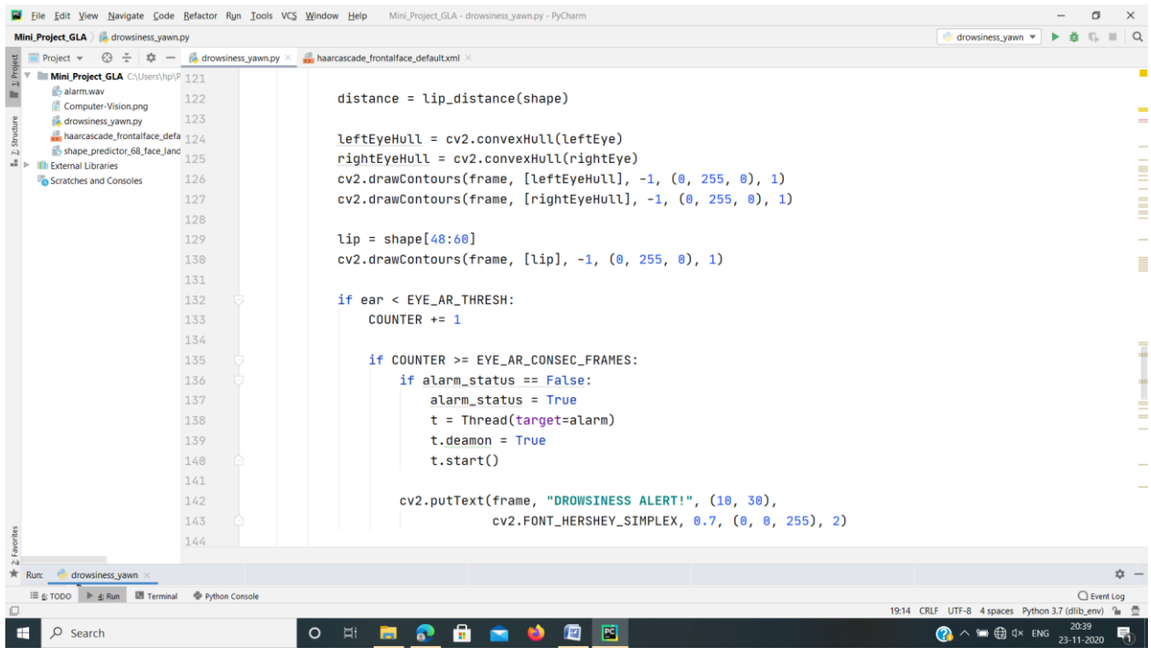
```
        (rStart, rEnd) = face_utils.FACIAL_LANDMARKS_IDXS["right_eye"]

        leftEye = shape[lStart:lEnd]
        rightEye = shape[rStart:rEnd]

        leftEAR = eye_aspect_ratio(leftEye)
        rightEAR = eye_aspect_ratio(rightEye)

        ear = (leftEAR + rightEAR) / 2.0
        return (ear, leftEye, rightEye)

    def lip_distance(shape):
        top_lip = shape[50:53]
        top_lip = np.concatenate((top_lip, shape[61:64]))

        low_lip = shape[56:59]
        low_lip = np.concatenate((low_lip, shape[65:68]))

        top_mean = np.mean(top_lip, axis=0)
        low_mean = np.mean(low_lip, axis=0)

        distance = abs(top_mean[1] - low_mean[1])
        return distance
```



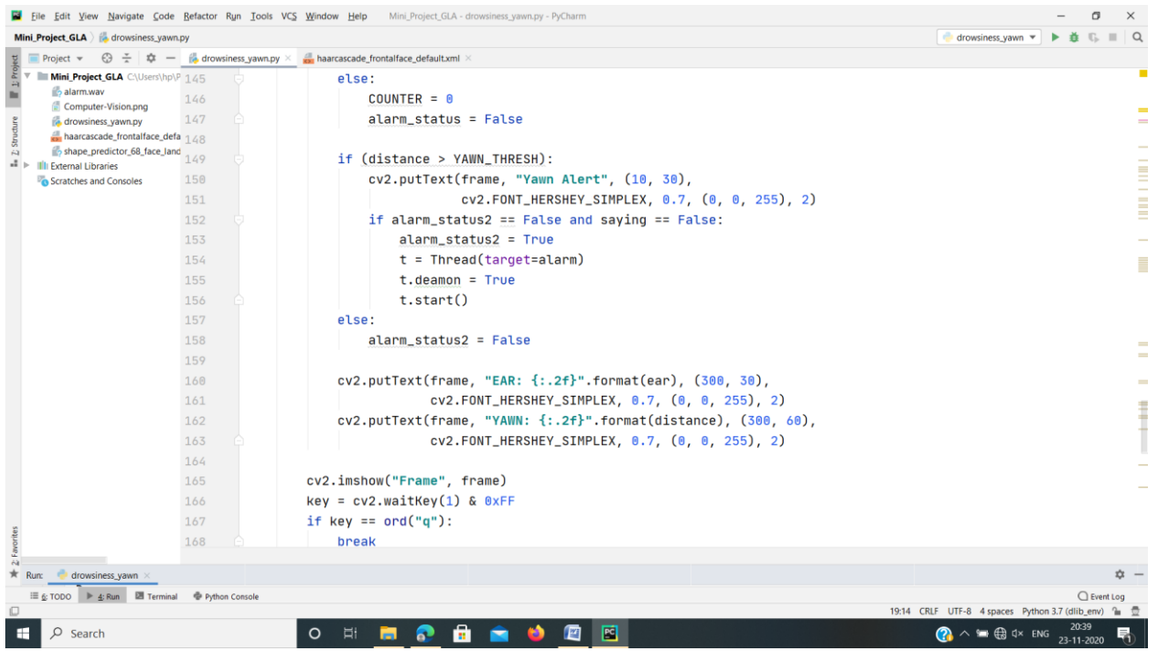
```
    ap = argparse.ArgumentParser()
    ap.add_argument("-w", "--webcam", type=int, default=0,
                    help="index of webcam on system")
    args = vars(ap.parse_args())

    EYE_AR_THRESH = 0.3
    EYE_AR_CONSEC_FRAMES = 30
    YAWN_THRESH = 20
    alarm_status = False
    alarm_status2 = False
    saying = False
    COUNTER = 0

    print("-> Loading the predictor and detector...")
    detector = dlib.get_frontal_face_detector()
    # detector = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")  # Faster but less accurate
    predictor = dlib.shape_predictor('shape_predictor_68_face_landmarks.dat')

    print("-> Starting Video Stream")
    vs = VideoStream(src=args["webcam"]).start()
    # vs= VideoStream(usePiCamera=True).start()       //For Raspberry Pi
    time.sleep(1.0)

    while True:
```



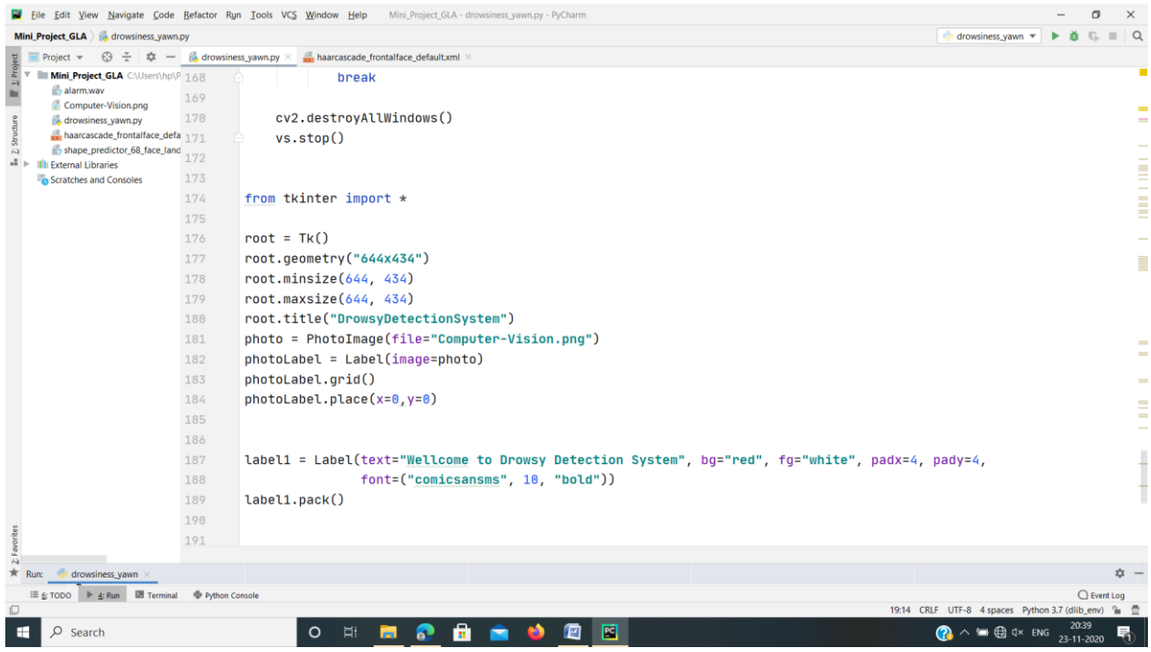
```
    while True:

        frame = vs.read()
        frame = imutils.resize(frame, width=450)
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        rects = detector(gray, 0)
        # rects = detector.detectMultiScale(gray, scaleFactor=1.1,
        # minNeighbors=5, minSize=(30, 30),
        # flags=cv2.CASCADE_SCALE_IMAGE)

        for rect in rects:
            # for (x, y, w, h) in rects:
            # rect = dlib.rectangle(int(x), int(y), int(x + w), int(y + h))

            shape = predictor(gray, rect)
            shape = face_utils.shape_to_np(shape)

            eye = final_ear(shape)
            ear = eye[0]
            leftEye = eye[1]
            rightEye = eye[2]
```



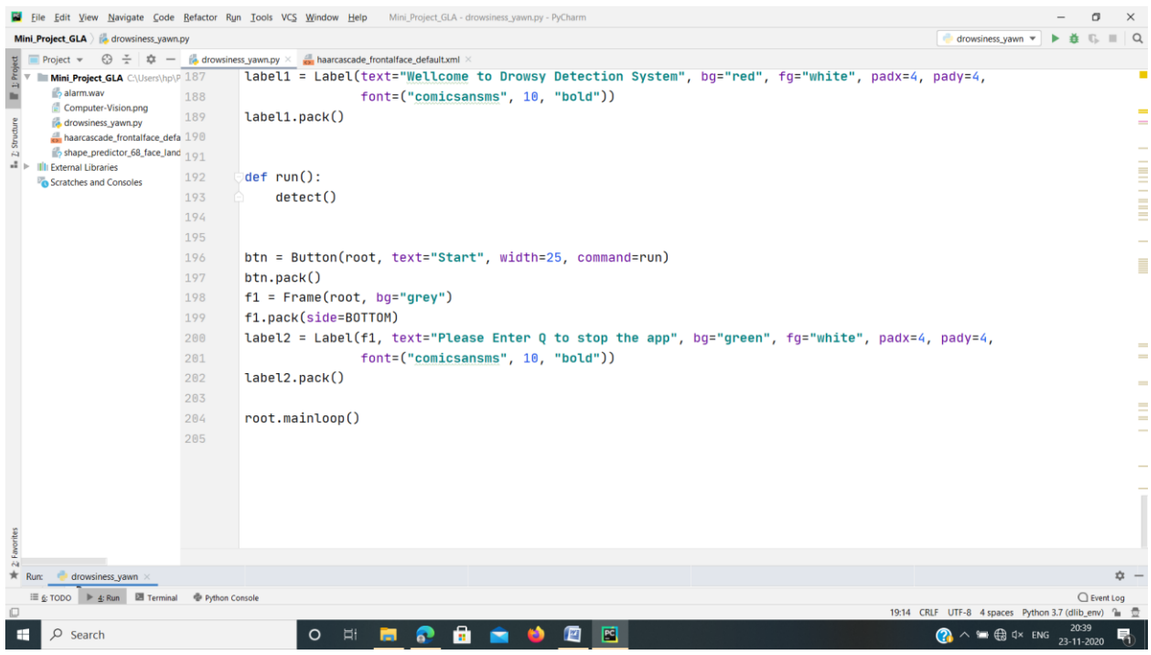
```
            distance = lip_distance(shape)

            leftEyeHull = cv2.convexHull(leftEye)
            rightEyeHull = cv2.convexHull(rightEye)
            cv2.drawContours(frame, [leftEyeHull], -1, (0, 255, 0), 1)
            cv2.drawContours(frame, [rightEyeHull], -1, (0, 255, 0), 1)

            lip = shape[48:60]
            cv2.drawContours(frame, [lip], -1, (0, 255, 0), 1)

            if ear < EYE_AR_THRESH:
                COUNTER += 1

                if COUNTER >= EYE_AR_CONSEC_FRAMES:
                    if alarm_status == False:
                        alarm_status = True
                        t = Thread(target=alarm)
                        t.deamon = True
                        t.start()

                    cv2.putText(frame, "DROWSINESS ALERT!", (10, 30),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
```



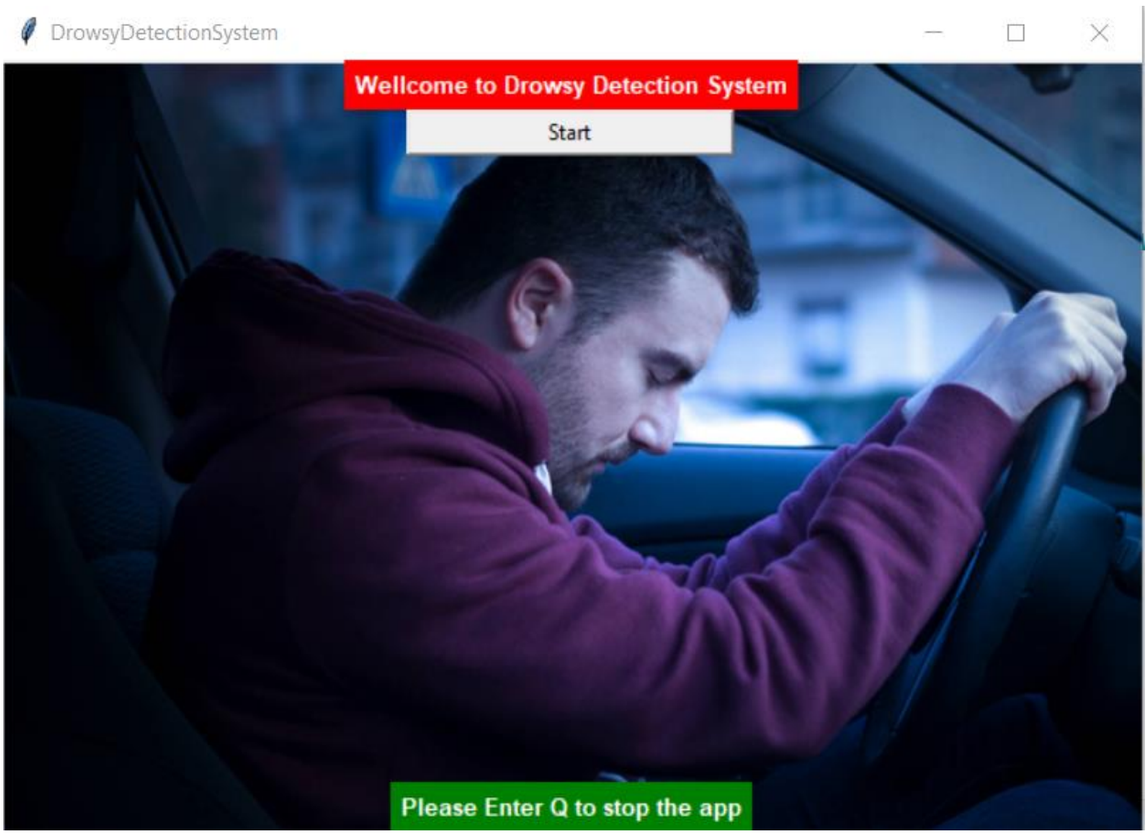
```
            else:
                COUNTER = 0
                alarm_status = False

            if (distance > YAWN_THRESH):
                cv2.putText(frame, "Yawn Alert", (10, 30),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
                if alarm_status2 == False and saying == False:
                    alarm_status2 = True
                    t = Thread(target=alarm)
                    t.deamon = True
                    t.start()
            else:
                alarm_status2 = False

            cv2.putText(frame, "EAR: {:.2f}".format(ear), (300, 30),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
            cv2.putText(frame, "YAWN: {:.2f}".format(distance), (300, 60),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)

        cv2.imshow("Frame", frame)
        key = cv2.waitKey(1) & 0xFF
        if key == ord("q"):
            break
```

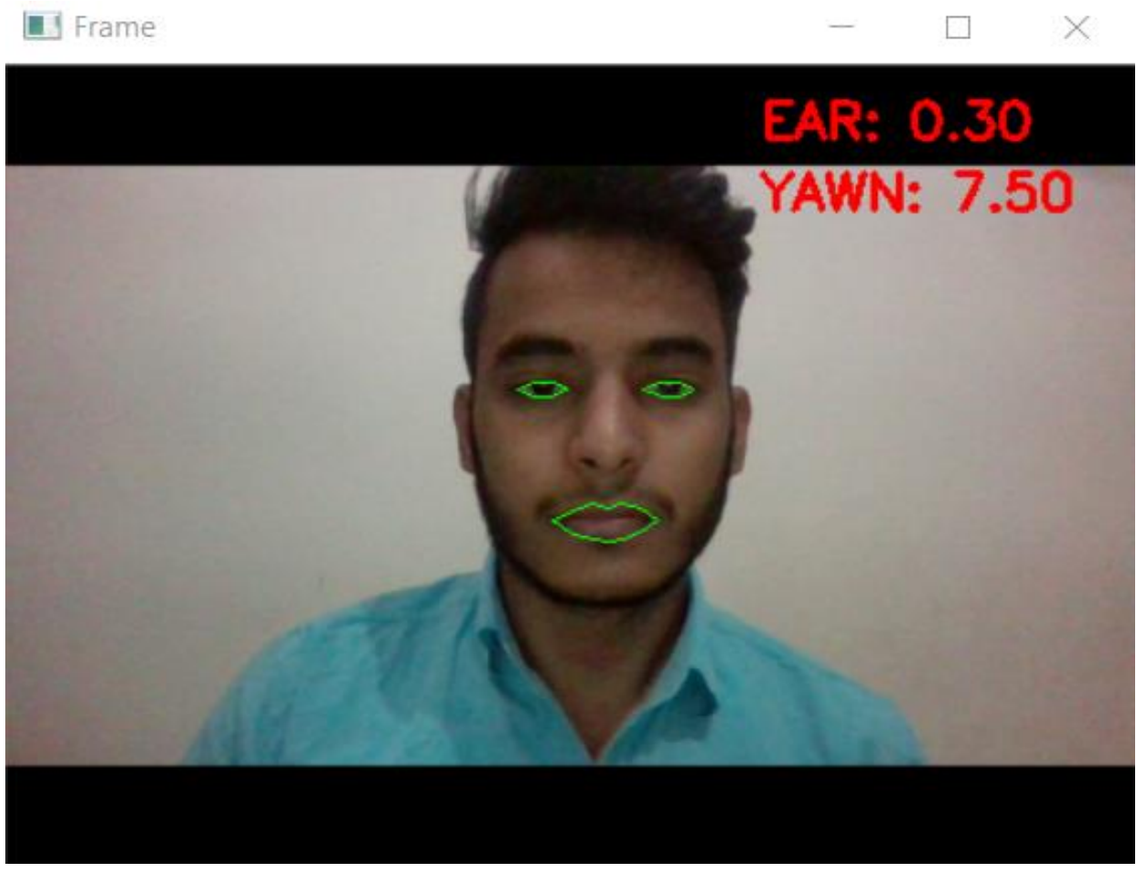


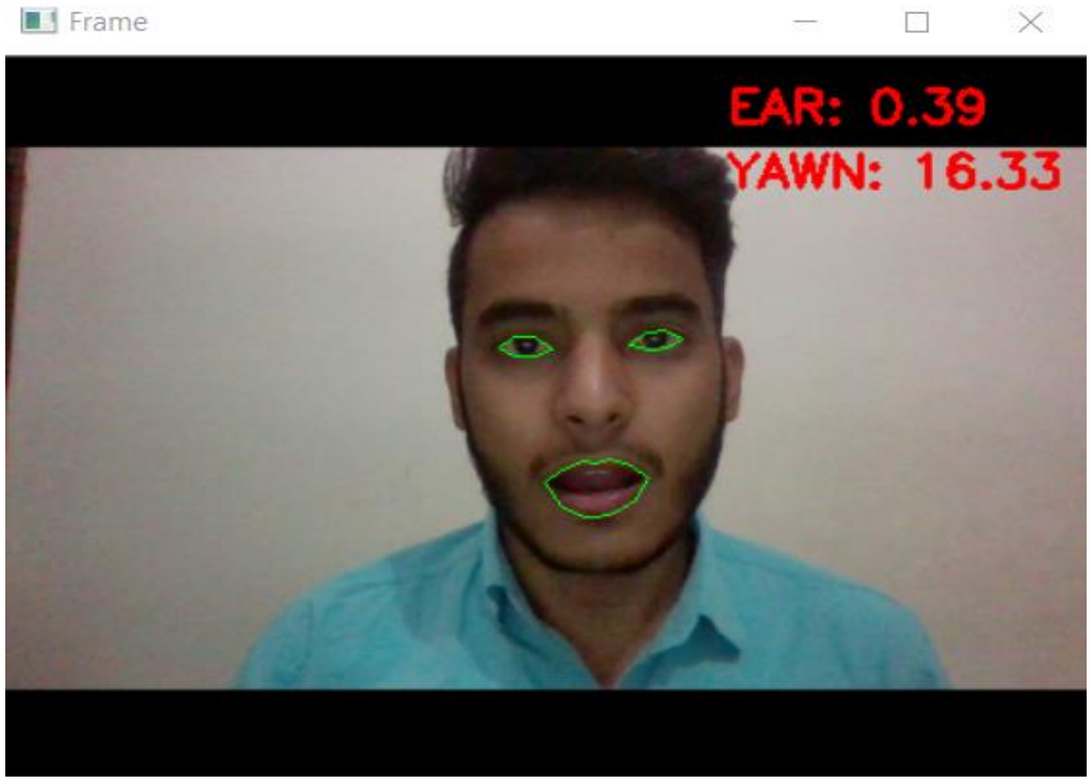
```
            break

    cv2.destroyAllWindows()
    vs.stop()


from tkinter import *

root = Tk()
root.geometry("644x434")
root.minsize(644, 434)
root.maxsize(644, 434)
root.title("DrowsyDetectionSystem")
photo = PhotoImage(file="Computer-Vision.png")
photoLabel = Label(image=photo)
photoLabel.grid()
photoLabel.place(x=0,y=0)


label1 = Label(text="Wellcome to Drowsy Detection System", bg="red", fg="white", padx=4, pady=4,
               font=("comicsansms", 10, "bold"))
label1.pack()
```



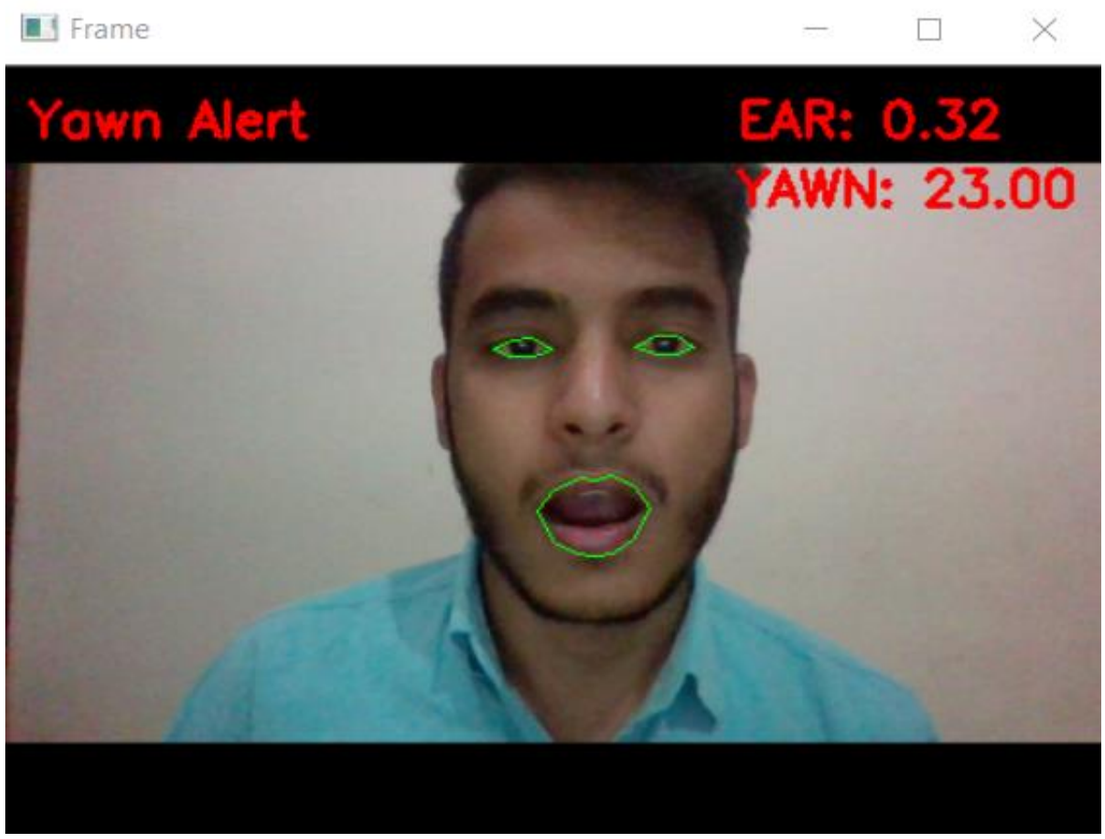
```
label1 = Label(text="Wellcome to Drowsy Detection System", bg="red", fg="white", padx=4, pady=4,
               font=("comicsansms", 10, "bold"))
label1.pack()


def run():
    detect()


btn = Button(root, text="Start", width=25, command=run)
btn.pack()
f1 = Frame(root, bg="grey")
f1.pack(side=BOTTOM)
label2 = Label(f1, text="Please Enter Q to stop the app", bg="green", fg="white", padx=4, pady=4,
               font=("comicsansms", 10, "bold"))
label2.pack()

root.mainloop()
```


DrowsyDetectionSystem
Wellcome to Drowsy Detection System
Start
Please Enter Q to stop the app


Frame
EAR: 0.30
YAWN: 7.50


Frame
EAR: 0.39
YAWN: 16.33

Frame
DROWSINESS ALERT!
EAR: 0.08
YAWN: 8.33


Frame
Yawn Alert
EAR: 0.32
YAWN: 23.00

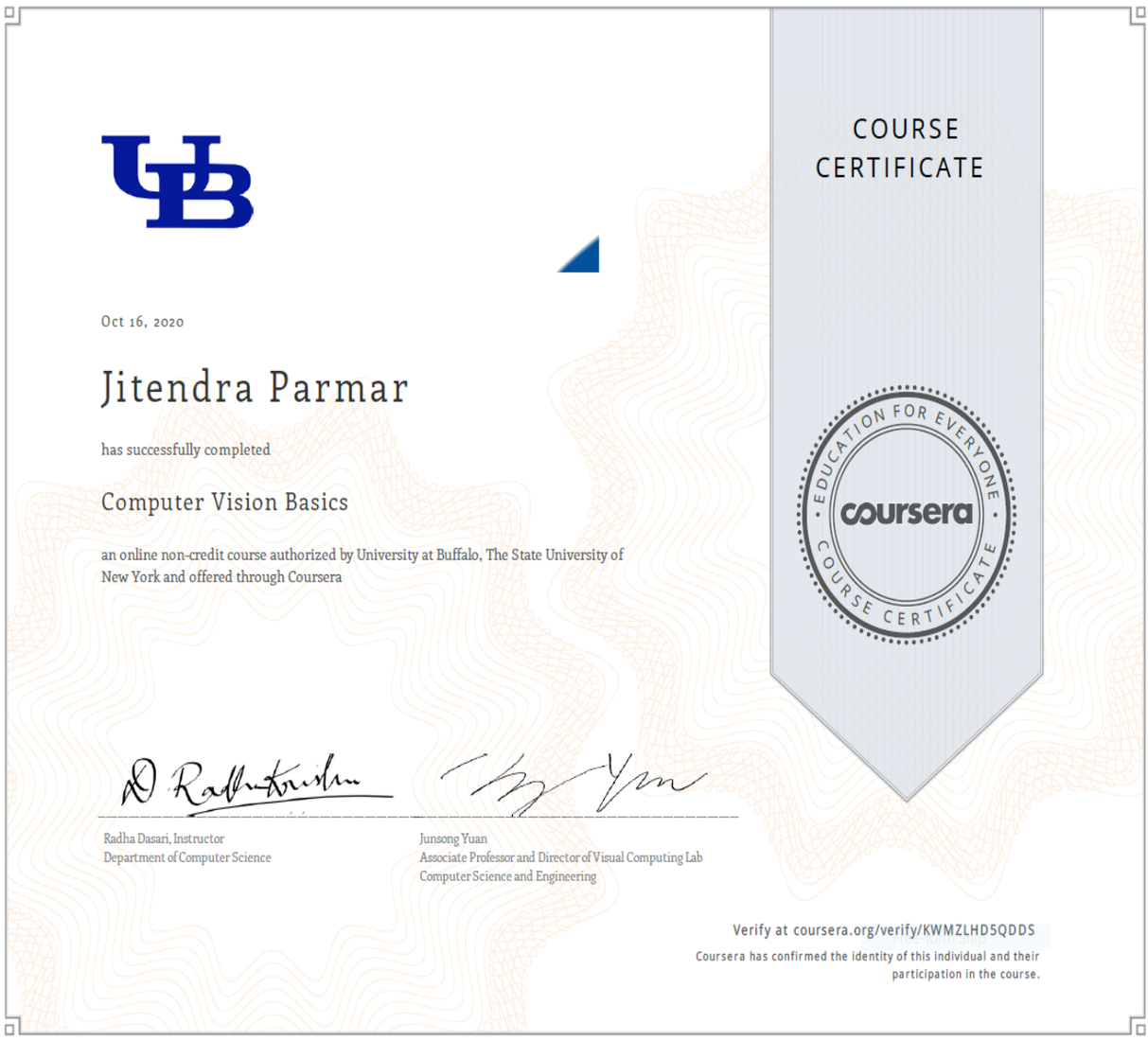
Frame
ALERT!
EAR: 0.16
YAWN: 24.83

REFERENCES

- Book References
  - Think Python: An Introduction to Software Design(Allen B. Downey)
  - OpenCV Computer Vision with Python(Joseph Howse)

- WWW.w3school.com
- WWW.javatpoint.com
- WWW.youtube.com
- www.tutorialspoint.com
- Faculty Guidelines
  - Priya Agrawal
  - Mr. Nikhil Govil

Certificates


COURSE
CERTIFICATE
Oct 16, 2020
Jitendra Parmar
has successfully completed
Computer Vision Basics
an online non-credit course authorized by University at Buffalo, The State University of New York and offered through Coursera
EDUCATION FOR EVERYONE
coursera
COURSE CERTIFICATE
Radha Dasari, Instructor
Department of Computer Science
Junsong Yuan
Associate Professor and Director of Visual Computing Lab
Computer Science and Engineering
Verify at coursera.org/verify/KWMZLHD5QDDS
Coursera has confirmed the identity of this individual and their participation in the course.

coursera
Project
Network
COURSE
CERTIFICATE
ABHISHEK GUPTA
has successfully completed
Image Processing with Python
an online non-credit course authorized by Coursera Project Network and offered through Coursera
EDUCATION FOR EVERYONE
coursera
COURSE CERTIFICATE
David Dalsveen
David Dalsveen
Subject Matter Expert
Freedom Learning Group
Verify at coursera.org/verify/Z6L5RBL34CMA
Coursera has confirmed the identity of this individual and their participation in the course.