INRTRODUCTION

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

The former technique, while more accurate, is not realistic since highly sensitive electrodes would have to be attached directly on the driver’ body and hence which can be annoying and distracting to the driver. In addition, long time working would result in perspiration on the sensors, diminishing their ability to monitor accurately. The second technique is to measure physical changes (i.e. open/closed eyes to detect fatigue) is well suited for real world conditions since it is non-intrusive by using a video camera to detect changes. In addition, micro sleeps that are short period of sleeps lasting 2 to 3 minutes are good indicators of fatigue. Thus, by continuously monitoring the eyes of the driver one can detect the sleepy state of driver and a timely warning is issued.

OBJECTIVES

The objective of this project is to develop a model of detection of drowsiness system and alarm alert. The total concentration and focus will be placed on designing a drowsiness detection system that will correctly monitor real time state of open and closed eye of driver. By monitoring the eyes constantly, it can be observed that the driver symptoms fatigue can be detected early to avoid an accident.

This project outlines the design and development of a system that focuses on driver's drowsiness detection and prediction.

1. Monitoring the driver behaviour by observing the manoeuvre stability and performance.
2. Validate and measure the progress by using Specific algorithm.
3. Warning the drivers if the behaviour beyond the thresholds.

Here we will employ machine learning methods to classify the data of actual human behaviour during drowsiness.

Devices to detect when drivers are trying to sleep and to provide alert warnings to them of the risk. Drowsiness driving has become a serious concern where even researches are unable to decide which factor lead to the accident. Hence the main objective of the proposed system is to detect drowsiness of the driver using image processing techniques to detect open and closed eye.

PROJECT CATEGORY

This project as title “**Real Time Drowsiness Detection System**” is comes under the **Computer Vision System**. This application is developed with the help of Python and OpenCV. This application is also based on image processing methodologies so it can also be called Image Processing application.

TOOLS/PLATFORMS

1. **Python:** Python is a high-level programming language designed to be easy to read and simple to implement. It is open source, which means it is free to use, even for commercial applications. Python can run on Mac, Windows, and Unix systems and has also been ported to Java and .NET virtual machines. Python is considered a scripting language, like Ruby or Perl and is often used for creating Web applications and dynamic Web content. It is also supported by a number of 2D and 3D imaging programs, enabling users to create custom plug-ins and extensions with Python. Examples of applications that support a Python API include GIMP, Inkscape, Blender, and Autodesk Maya.

2. **OpenCV**: OpenCV stands for Open Source Computer Vision. It's an Open Source BSD licensed library that includes hundreds of advanced Computer Vision algorithms that are optimized to use hardware acceleration. OpenCV is commonly used for machine learning, image processing, image manipulation, and much more. OpenCV has a modular structure. There are shared and static libraries and a CV Namespace. In short, OpenCV is used in our application to easily load bitmap files that contain landscaping pictures and perform a blend operation between two pictures so that one picture can be seen in the background of another picture. This image manipulation is easily performed in a few lines of code using OpenCV versus other methods. OpenCV.org is a must if you want to explore and dive deeper into image processing and machine learning in general.

3. **Machine learning**: Machine learning is the kind of programming which gives computers the capability to automatically learn from data without being explicitly programmed. This means in other words that these programs change their behaviour by learning from data. Python is clearly one of the best languages for machine learning. Python does contain special libraries for machine learning namely scipy, pandas and numpy which great for linear algebra and getting to know kernel methods of machine learning. The language is great to use when working with machine learning algorithms and has easy syntax relatively.

4. **Dlib**: DLib is an open source library implementing a variety of machine learning algorithms, including classification, regression, clustering, data transformation, and structured prediction. DLib is much like DMTL in that it provides a generic high-performance machine learning toolkit with many different algorithms, but DLib is more recently updated and has more examples. DLib also contains much more supporting functionality.

Hardware and Software Requirements

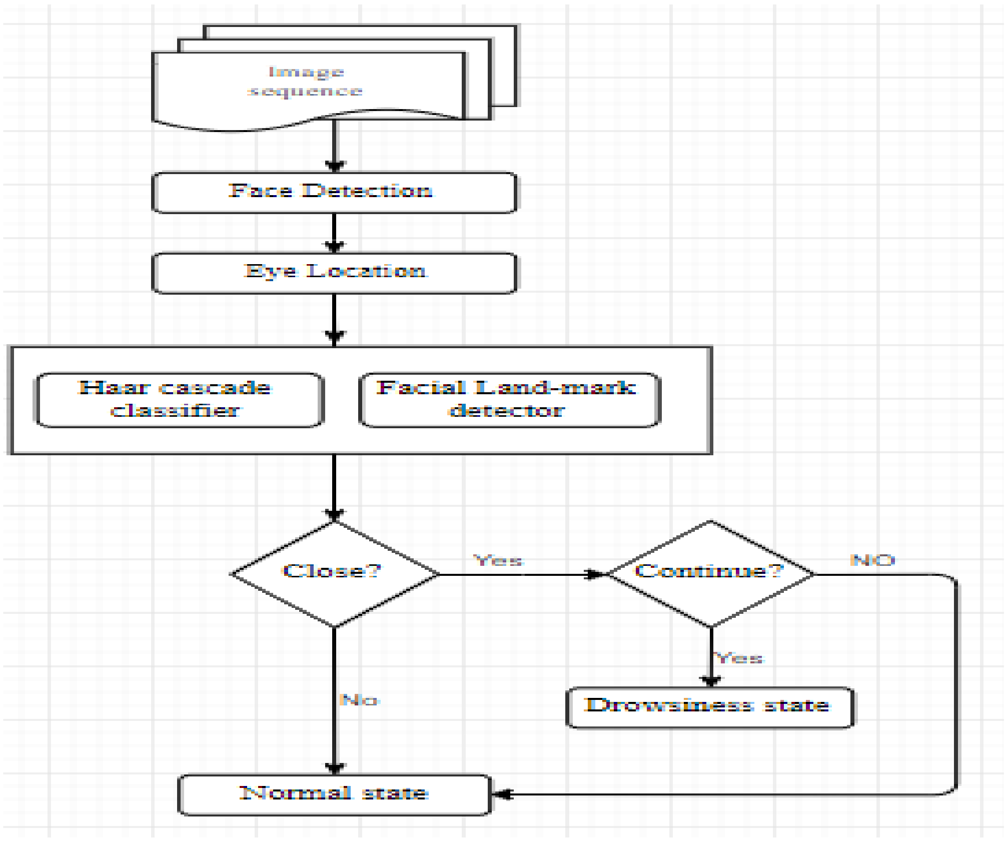
HARDWARE RESOURCES:

1. System Processor: Pentium IV or later.
2. Bus: 32- Bit.
3. RAM: 512MB DDR RAM.
4. Hard drive: 20GB.
5. Display: SVGA Colour
6. Key board: Windows compatible.
7. Web Camera (OPTIONAL).

SOFTWARE RESOURCES:

1. Python 2.7 or above version
2. Operating System: Windows 7/8 or above
3. Libraries: OpenCV and Dlib

SYSTEM DESIGN



1. 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. Fig. 3.4 represents five haar like features & example is shown in Fig.3.5

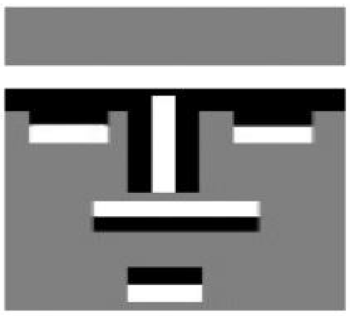


Fig 3.4

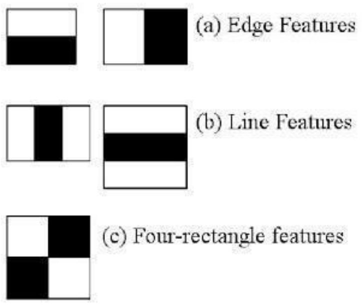


Fig 3.5

2. 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 labelled facial landmarks on an image. These images are manually labelled, 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.
The indexes of the 68 coordinates can be visualized on the image below:

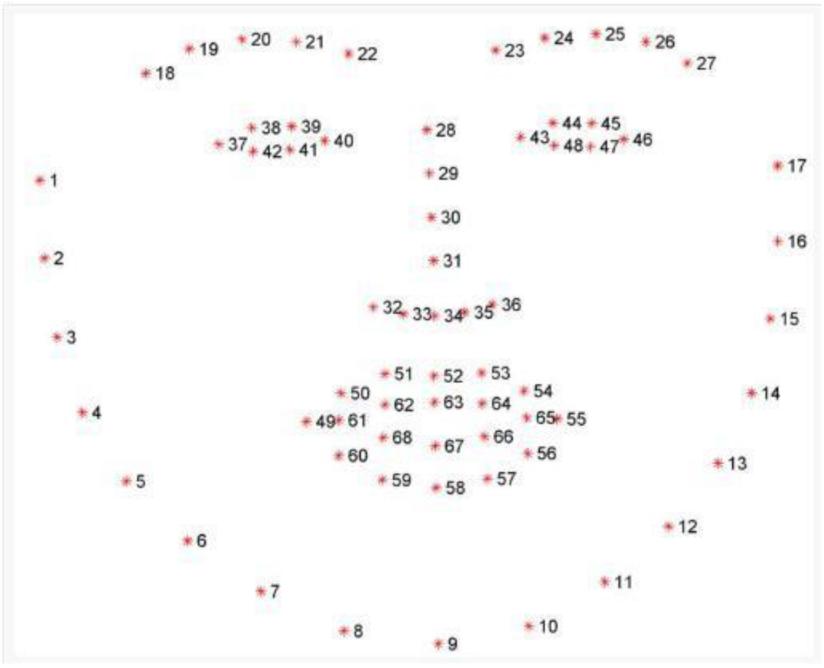


Fig.3.6: Visualizing the 68 facial landmark coordinates

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.

3. 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. pFinally, 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 non-blinking patterns.
Eye Aspect 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 = $\|p2 - p6\| + \|p3 - p5\|$ (1)
$2\|p1 - p4\|$
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.

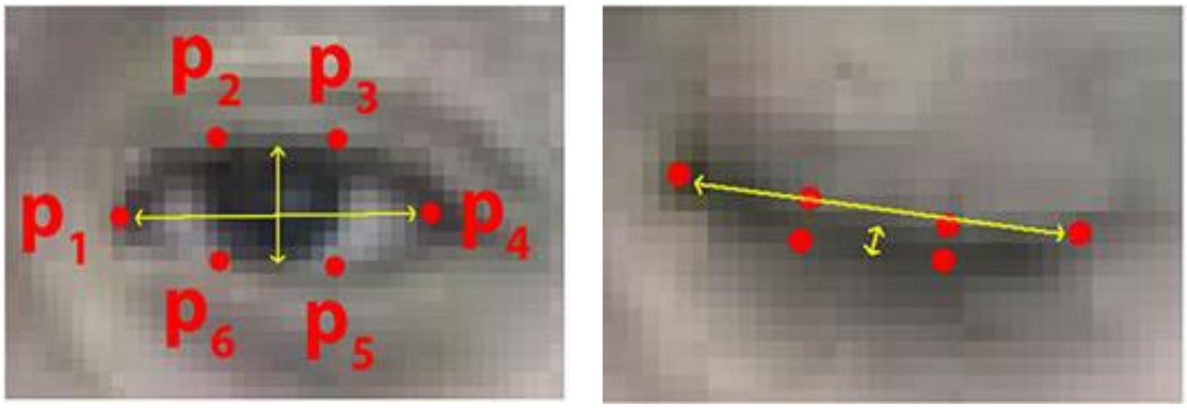


Fig 3.8: Open and closed eyes with landmarks p(i) automatically detected. The eye aspect ratio EAR in Eq. (1) plotted for several frames of a video sequence.

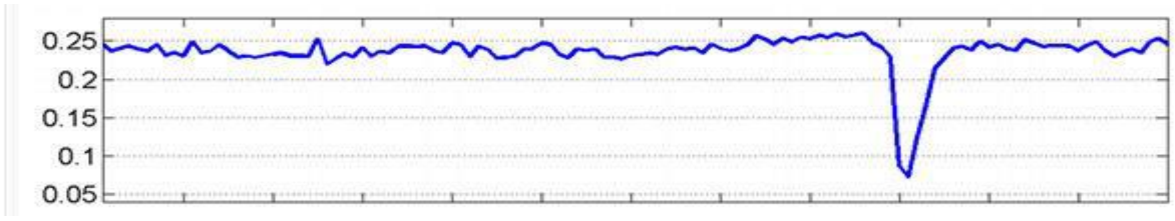
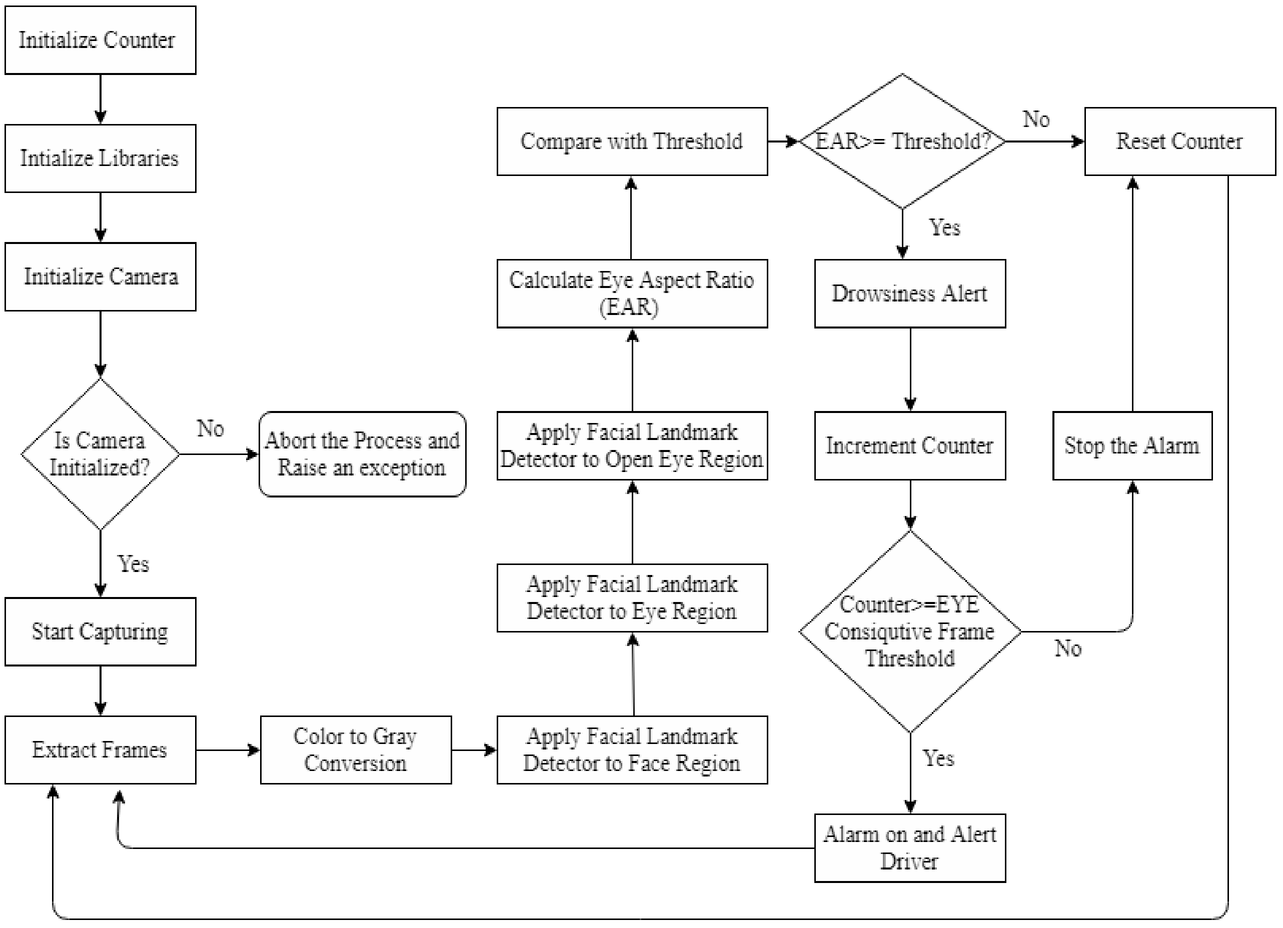


Fig.3.9 EAR for single blink

4. 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”.

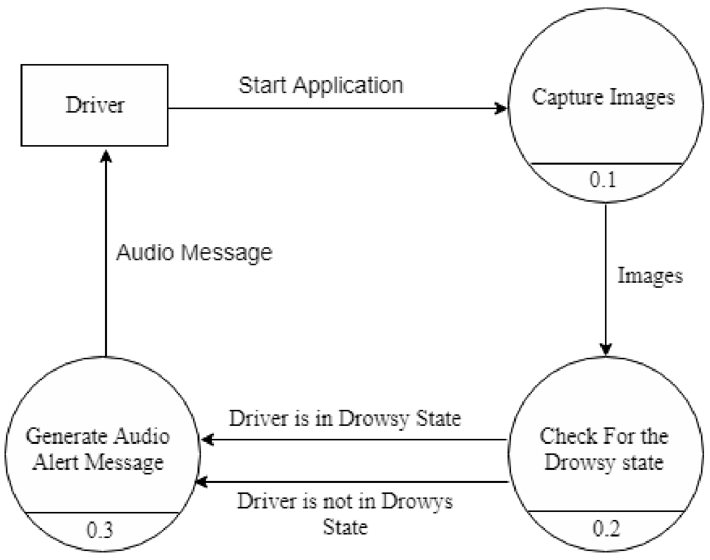
5. Drowsiness Detection:
The last step of the algorithm is to determine the person’s condition based on a pre-set condition for drowsiness. The average blink duration of a person is 100-400 milliseconds (i.e. 0.1-0.4 of a second). Hence if a person is drowsy his eye closure must be beyond this interval.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

MODELING DIAGRAM (PROJECT WORK FLOW)



Initialize Counter
Intialize Libraries
Initialize Camera
Is Camera Initialized?
No
Abort the Process and Raise an exception
Yes
Start Capturing
Extract Frames
Color to Gray Conversion
Apply Facial Landmark Detector to Face Region
Apply Facial Landmark Detector to Eye Region
Apply Facial Landmark Detector to Open Eye Region
Calculate Eye Aspect Ratio (EAR)
Compare with Threshold
EAR>= Threshold?
No
Reset Counter
Yes
Drowsiness Alert
Increment Counter
Counter>=EYE Consiqutive Frame Threshold
No
Stop the Alarm
Yes
Alarm on and Alert Driver

Fig 4.1 Flowchart of System



Driver
Start Application
Capture Images
0.1
Images
Check For the Drowsy state
0.2
Driver is in Drowsy State
Driver is not in Drowys State
Generate Audio Alert Message
0.3
Audio Message

Fig 4.2 DFD Level 0

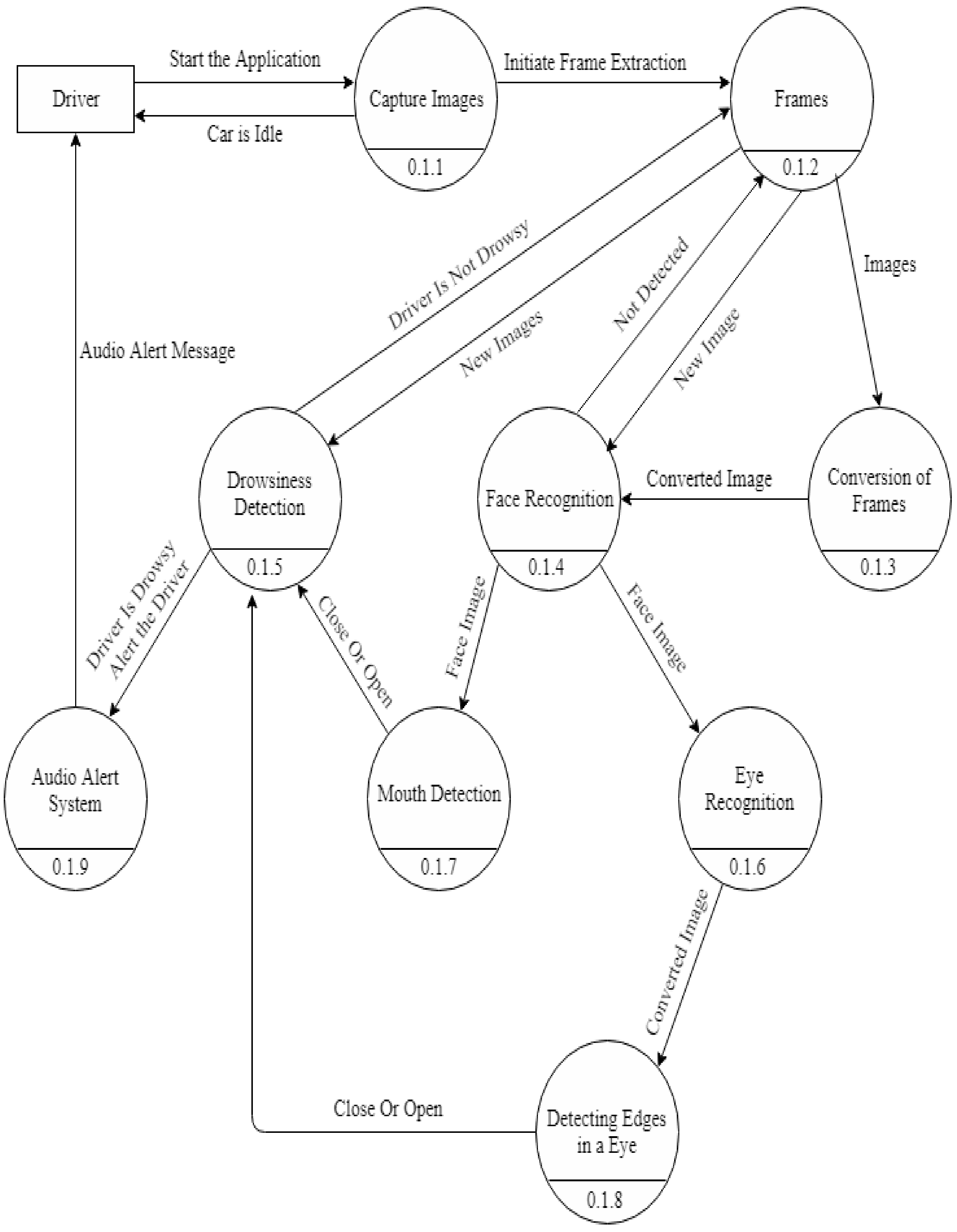


Fig 4.3 DFD Level 1

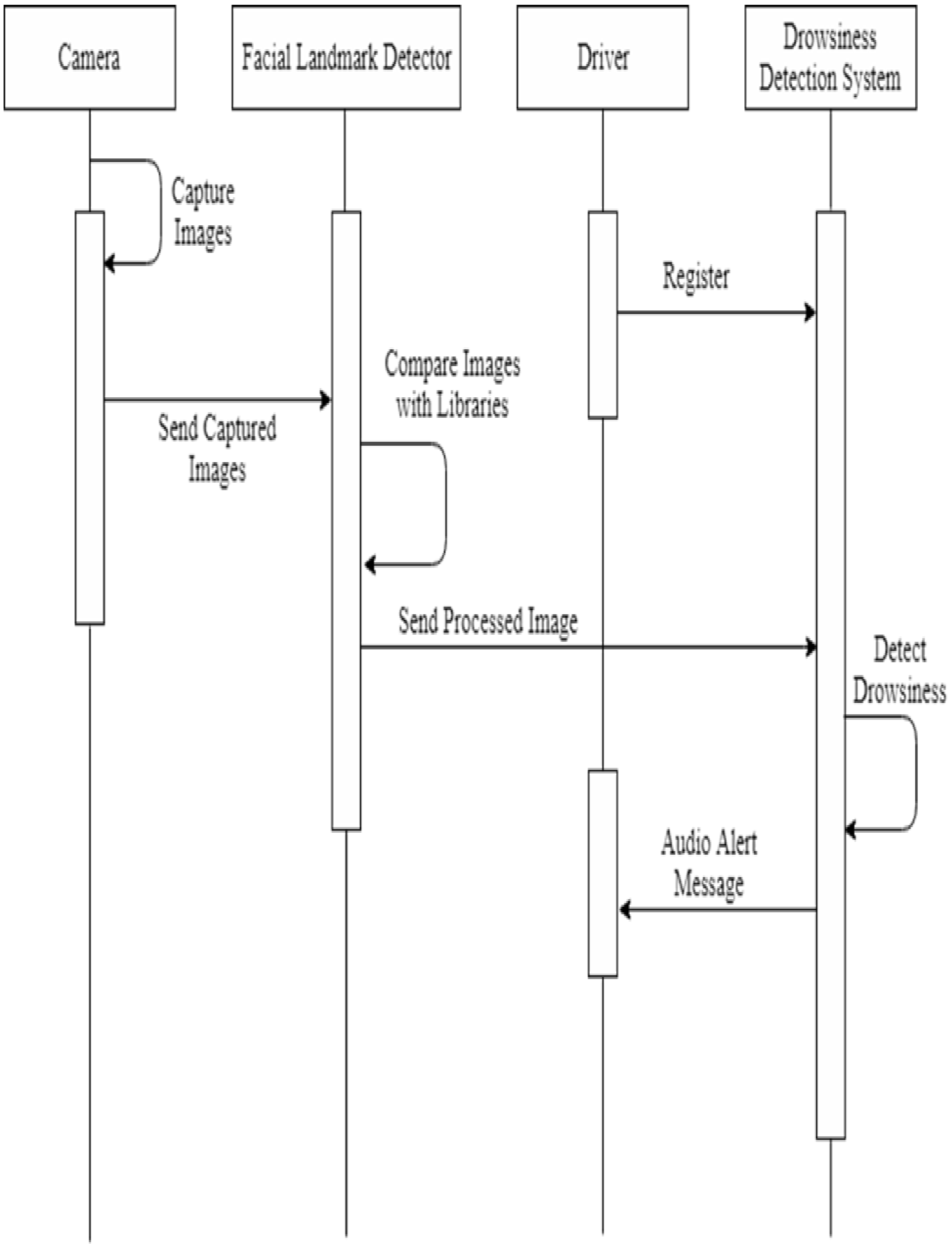


Fig 4.4 Sequence Diagram

MODULES USED IN THE PROJECT

1) Webcam: The webcam is used mainly to capture the static image or the video stream. This captured image or video stream is given as input to the system.

2) Detect Drowsy: This module includes the following things: Image processing, Face detection, Eye region detection, Detect closeness of eye.

3) Image Processing: The image processing module will process the obtained image or video input fed into the system. It processes the input and convert it into frames for further processing.

4) Face Detection: The face detection technique is used to locate the face from the image. The video stream which is given as input is converted into frames and the face is detected from those frames. Haar classifier in Viola Jones algorithm is used to detect the face from a given image.

5) Eye Detection: The position of the driver's eye is determined by using appropriate threshold. In this work, edge detection of the eyes region is considered.

6) Detect Closeness of Eye: After locating the eye region in the frame the system finds out whether the eye is in a closed state or in the open state. If the eye remains open then the system gives the message that the driver is not feeling drowsy. If the eyes remain closed then the system gives the message that the driver is feeling drowsy.

7) Output: The output from the system includes: Alert,Message to the cab owner,Indication to nearby vehicles. The indication is given in order to prevent road accidents which occur due to the drowsiness of the cab drivers.

a) Alert: The system continuously checks whether the driver is feeling drowsy or not. If the driver feels drowsy then an alert signal in the form of an alarm sound or a beeo sound is generated. This is done in order to make the driver to wake up from a sleepy state.

RESULTS

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 analysed. 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 analysing 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.

Screenshots of the system shown below:

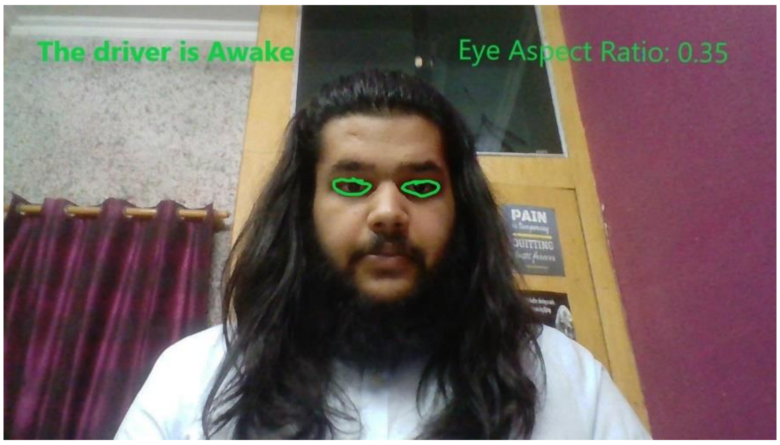


Figure 1: When a driver is awake

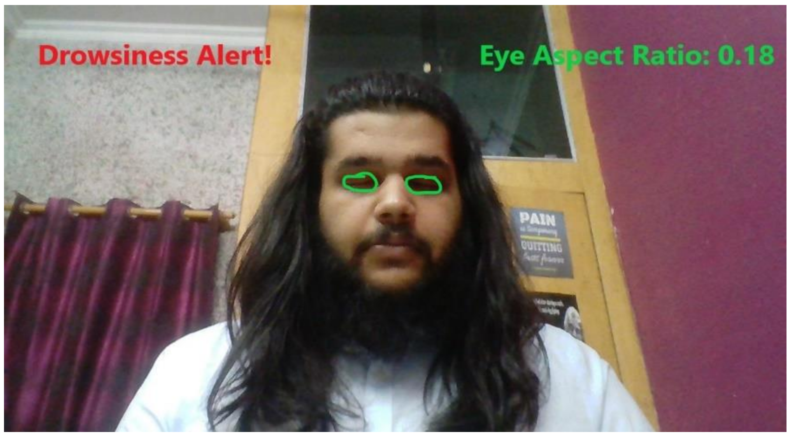


Figure 2: When a driver closes the eye to sleep.

FUTURE SCOPE OF THE PROJECT

In last decades many researchers worked on drowsiness detection and got significant success. However, still, research is going on in this field to make the system more reliable. There are some challenges with this application. For example, if a driver is wearing spectacles, having a partial face that could not be seen by a camera or sensor. 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.

1) For future work, we will extend our application to detect drowsiness even when the subject is using sunglass or colour spectacles. Another future study can be done to detect drowsiness is to develop a system which can detect drowsiness in the night or with very low illumination.

2) Real-time data are always unconstrained and with unconstrained nature of the subject/driver, it becomes very difficult to find facial landmarks. Therefore, to overcome this constraint a more reliable application can be developed which can detect face even when user/subject is not friendly.

3) To increase the sensitivity of application, a multimodal system can be developed which not uses only eye feature, but also facial expression, mouth feature and combine all features to make the decision where driver/subject is sleeping or not.

We can also provide the user with an Android application which will provide with the information of his/her drowsiness level during any journey. The user will know Normal state, Drowsy State, the number of times blinked the eyes according to the number of frames captures.

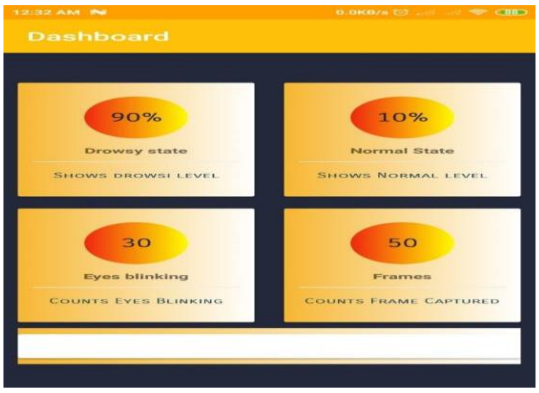


Fig 5.1 Android Application for Future scope

CONCLUSION

Now a day, the accident ratio has been increased in large amount. Especially in a country like India, the rate has become more, and the main reason for these accidents to cause is drivers fatigue or drowsiness. We should always remember life gives us only one chance. Therefore, this is a device which would help a huge amount of percent from accidents. This system is used to detect drowsiness while driving by giving a buzzer and also alerting people by alert messages.

A non-invasive system to localize the eyes and monitor fatigue was developed. Information about the eyes position is obtained through various self-developed image processing algorithms. During the monitoring, the system is able to decide if the eyes are opened or closed. When the eyes have been closed for too long, a warning signal is issued. In addition, during monitoring, the system is able to automatically detect any eye localizing error that might have occurred. In case of this type of error, the system is able to recover and properly localize the eyes. The following conclusions were made:

• Image processing achieves highly accurate and reliable detection of drowsiness.

• Image processing offers a non-invasive approach to detecting drowsiness without the annoyance and interference.

• A drowsiness detection system developed around the principle of image processing judges the driver’s alertness level on the basis of continuous eye closures. With 80% accuracy, it is obvious that there are limitations to the system.

REFERENCES

IEEE standard

Journal Paper,

[1] Facial Features Monitoring for Real Time Drowsiness Detection by Manu B.N, 2016 12th International Conference on Innovations in Information Technology (IIT) [Pg. 78-81]
https://ieeexplore.ieee.org/document/7880030

[2] Real Time Drowsiness Detection using Eye Blink Monitoring by Amna Rahman Department of Software Engineering Fatima Jinnah Women University 2015 National Software Engineering Conference (NSEC 2015)
https://ieeexplore.ieee.org/document/7396336

[3] International Journal of Science, Engineering and Technology Research (IJSETR) Volume 2, Issue 9, September 2013

Names of Websites referred

1. https://www.codeproject.com/Articles/26897/TrackEye-Real-Time-Tracking-Of-Human-Eyes- Using-a

2. https://realpython.com/face-recognition-with-python/
https://www.pyimagesearch.com/2017/04/24/eye-blink-detection-opencv-python-dlib/

3. https://www.pyimagesearch.com/2017/04/03/facial-landmarks-dlib-opencv-python/

4. https://www.pyimagesearch.com/2017/04/10/detect-eyes-nose-lips-jaw-dlib-opencv python/

5. https://www.codeproject.com/Articles/26897/TrackEye-Real-Time-
Tracking-Of-HumanEyesUsing-a

6. https://docs.opencv.org/3.4/d7/d8b/tutorial_py_face_detection.html

7. https://www.learnopencv.com/training-better-haar-lbp-cascade-eye-detector-opencv/

APPENDIX

Algorithm:

1. Capture the image of the driver from the camera.
2. Send the captured image to haarcascade file for face detection.
3. If the face is detected then crop the image consisting of the face only. If the driver is distracted then a face might not be detected, so play the buzzer.
4. Send the face image to haarcascade file for eye detection.
5. If the eyes are detected then crop only the eyes and extract the left and right eye from that image. If both eyes are not found, then the driver is looking sideways, so sound the buzzer.
6. The cropped eye images are sent to the hough transformations for detecting pupils, which will determine whether they are open or closed.
7. If they are found to be closed for five continuous frames, then the driver should be alerted by playing the buzzer.

Coding:

Importing our required Python packages.

detect_drowsiness.py

```
# import the necessary packages
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 playsound
import argparse
import imutils
import time
import dlib
import cv2
```

Sound Alarm

```
def sound_alarm(path):
# play an alarm sound
playsound.playsound(path)
```

eye_aspect_ratio function

```
def eye_aspect_ratio(eye): 21
# compute the euclidean distances between the two sets of
# vertical eye landmarks (x, y)-coordinates
A = dist.euclidean(eye[1], eye[5])
```

```
B = dist.euclidean(eye[2], eye[4])
# compute the euclidean distance between the horizontal
# eye landmark (x, y)-coordinates
C = dist.euclidean(eye[0], eye[3])
# compute the eye aspect ratio
ear = (A + B) / (2.0 * C)
# return the eye aspect ratio
return ear
```

Parsing command Line Argument

```
# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-p", "--shape-predictor", required=True,
help="path to facial landmark predictor")
ap.add_argument("-a", "--alarm", type=str, default="",
help="path alarm .WAV file")
ap.add_argument("-w", "--webcam", type=int, default=0,
help="index of webcam on system")
args = vars(ap.parse_args())
```

Defining EYE_AR_THRESH

```
EYE_AR_THRESH = 0.3
EYE_AR_CONSEC_FRAMES = 48
COUNTER = 0
ALARM_ON = False 22
```

Facial landmark predictor

```
# initialize dlib's face detector (HOG-based) and then create
# the facial landmark predictor
print("[INFO] loading facial landmark predictor...")
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor(args["shape_predictor"])
```

Extracting the eye regions

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

Instantiate VideoStrem

```
# start the video stream thread
print("[INFO] starting video stream thread...")
vs = VideoStream(src=args["webcam"]).start()
time.sleep(1.0)
# loop over frames from the video stream
while True:
frame = vs.read()
frame = imutils.resize(frame, width=450)
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

```
# detect faces in the grayscale frame
rects = detector(gray, 0) 23
```

Facial landmark detection to localize each of the important regions of the face:

```
# loop over the face detections
for rect in rects:
# determine the facial landmarks for the face region, then
# convert the facial landmark (x, y)-coordinates to a NumPy
# array
shape = predictor(gray, rect)
shape = face_utils.shape_to_np(shape)
# extract the left and right eye coordinates, then use the
# coordinates to compute the eye aspect ratio for both
eyes
leftEye = shape[lStart:lEnd]
rightEye = shape[rStart:rEnd]
leftEAR = eye_aspect_ratio(leftEye)
rightEAR = eye_aspect_ratio(rightEye)
# average the eye aspect ratio together for both eyes
ear = (leftEAR + rightEAR) / 2.0
```

Visualize each of the eye regions

```
# compute the convex hull for the left and right eye, then
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) 24
```

Check to see if the person in our video stream is starting to show symptoms of drowsiness:

```
# check to see if the eye aspect ratio is below the blink
# threshold, and if so, increment the blink frame counter
if ear < EYE_AR_THRESH:
COUNTER += 1
# if the eyes were closed for a sufficient number of
# then sound the alarm
if COUNTER >= EYE_AR_CONSEC_FRAMES:
# if the alarm is not on, turn it on
if not ALARM_ON:
ALARM_ON = True
# check to see if an alarm file was supplied,
# and if so, start a thread to have the alarm
# sound played in the background
if args["alarm"] != "":
t = Thread(target=sound_alarm,
args=(args["alarm"],))
t.deamon = True
```

```
t.start()
# draw an alarm on the frame
cv2.putText(frame, "DROWSINESS ALERT!", (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
# otherwise, the eye aspect ratio is not below the blink
# threshold, so reset the counter and alarm
else:
COUNTER = 0
ALARM_ON = False 25
```

Displaying the output frame:

```
# draw the computed eye aspect ratio on the frame to help
# with debugging and setting the correct eye aspect ratio
# thresholds and frame counters
cv2.putText(frame, "EAR: {:.2f}".format(ear), (300, 30),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
# show the frame
cv2.imshow("Frame", frame)
key = cv2.waitKey(1) & 0xFF
# if the `q` key was pressed, break from the loop
if key == ord("q"):
break
# do a bit of cleanup
cv2.destroyAllWindows()
vs.stop()
```