DROWSINESS DETECTION SYSTEM

Report For the Assignment

Group No #39

Members:

MD Sohrab (BT17CSE041)

Vikas Jaiswar (BT19CSE042)

Prakshay Joshi (BT19CSE043)

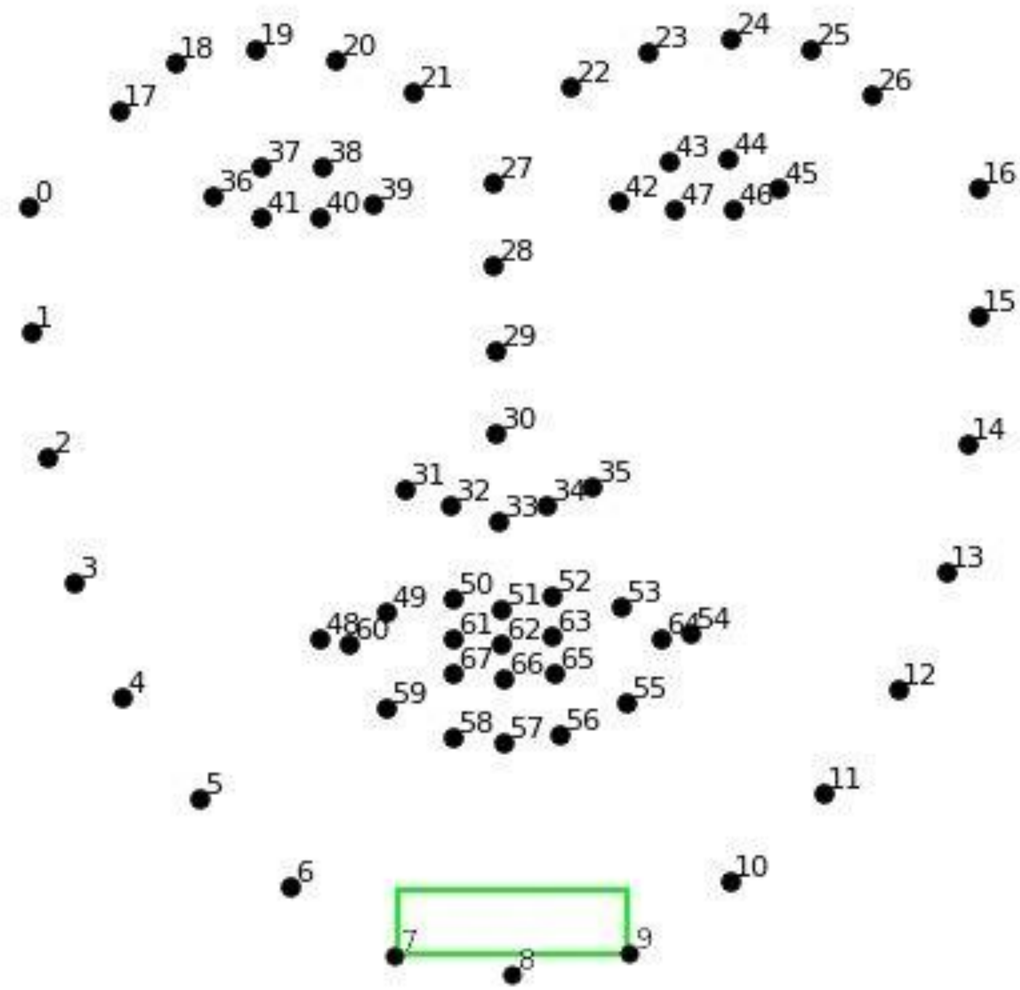
1. **Problem Statement :**

   Drowsiness Detection System.

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.

Logic of project :

The project includes direct working with the 68 facial landmark detector and also the face detector of the Dlib library. The 68 facial landmark detector is a robustly trained efficient detector which detects the points on the human face using which we determine whether the eyes are open or they are closed.

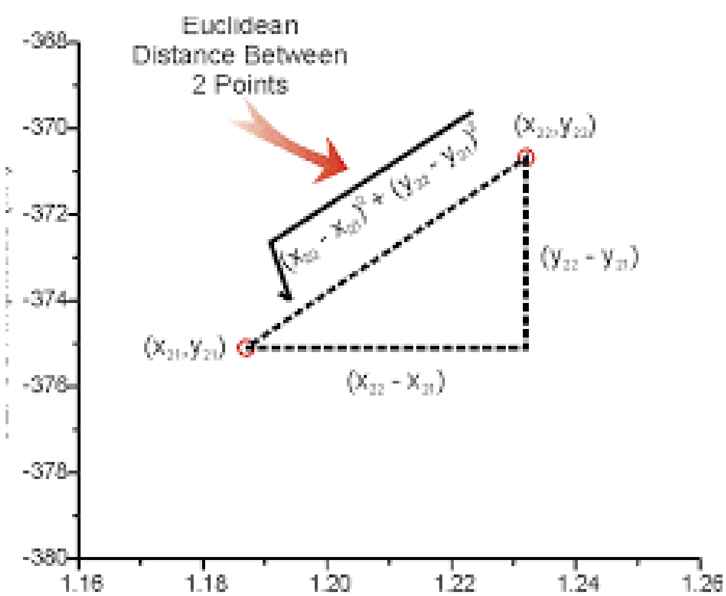


Dependencies :

1. **import cv2**
2. **import imutils**
3. **import dlib**

The working of the project

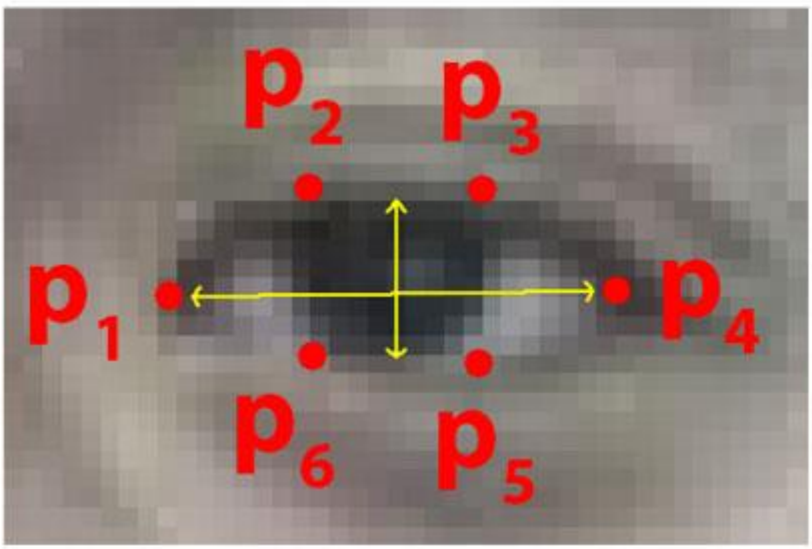
- The landmarks are detected using the detector.
- Now we are taking the ratio which is described as *'Sum of distances of vertical landmarks divided by twice the distance between horizontal landmarks'*.
- Now this ratio is totally dependent on your system which you may configure accordingly for the thresholds of sleeping, drowsy, active.



Methodology :

Each eye is represented by 6 (x, y)-coordinates, starting at the left-corner of the eye (as if you were looking at the person), and then working clockwise around the eye.

It checks 20 consecutive frames and if the Eye Aspect ratio is less than 0.25, Alert is generated.



Relation:

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

IMPLEMENTATION AND RESULT:

Step 1 – Take Image as Input from a Camera

With a webcam, we will take images as input. So to access the webcam, we made an infinite loop that will capture each frame. We use the method provided by OpenCV, cv2.VideoCapture(0) to access the camera and set the capture object (cap). cap.read() will read each frame and we store the image in a frame variable.

Step 2 – Detect Face in the Image and Create a Region of Interest (ROI)

To detect the face in the image, we need to first convert the image into grayscale as the OpenCV algorithm for object detection takes gray images in the input. We don't need color information to detect the objects. We will be using haar cascade classifier to detect faces. This line is used to set our classifier face = cv2.CascadeClassifier(' path to our haar cascade xml file'). Then we perform the detection using faces = face.detectMultiScale(gray). It returns an array of detections with x,y coordinates, and height, the width of the boundary box of the object. Now we can iterate over the faces and draw boundary boxes for each face.

```
for (x,y,w,h) in faces:
        cv2.rectangle(frame, (x,y), (x+w, y+h), (100,100,100), 1 )
```

Step 3 – Detect the eyes from ROI and feed it to the classifier

The same procedure to detect faces is used to detect eyes. First, we set the cascade classifier for eyes in leye and reye respectively then detect the eyes using left_eye = leye.detectMultiScale(gray). Now we need to extract only the eyes data from the full image. This can be achieved by extracting the boundary box of the eye and then we can pull out the eye image from the frame with this code.

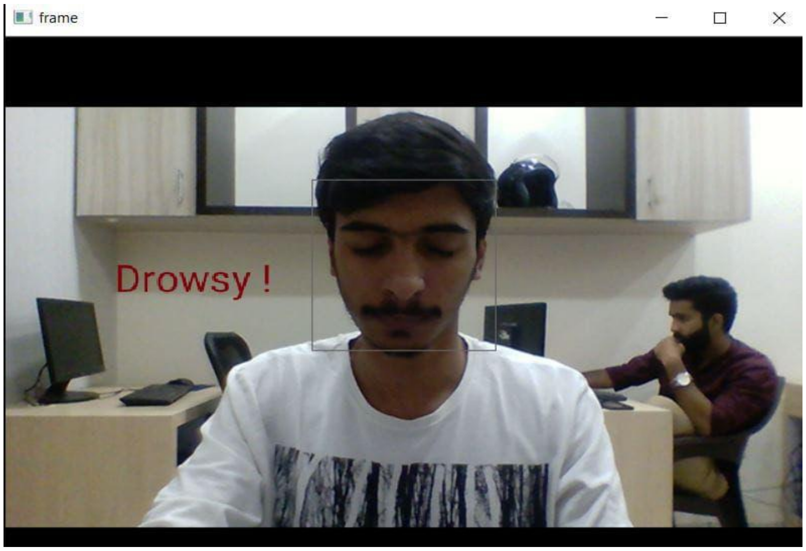
l_eye = frame[y : y+h, x : x+w]
l_eye only contains the image data of the eye. This will be fed into our CNN classifier which will predict if eyes are open or closed. Similarly, we will be extracting the right eye into r_eye.

Step 4 – Classifier will Categorize whether Eyes are Open or Closed

We are using CNN classifier for predicting the eye status. To feed our image into the model, we need to perform certain operations because the model needs the correct dimensions to start with. First, we convert the color image into grayscale using r_eye = cv2.cvtColor(r_eye, cv2.COLOR_BGR2GRAY). Then, we resize the image to 24*24 pixels as our model was trained on 24*24 pixel images cv2.resize(r_eye, (24,24)). We normalize our data for better convergence r_eye = r_eye/255 (All values will be between 0-1). Expand the dimensions to feed into our classifier. We loaded our model using model = load_model('models/cnnCat2.h5') . Now we predict each eye with our model
lpred = model.predict_classes(l_eye). If the value of lpred[0] = 1, it states that eyes are open, if value of lpred[0] = 0 then, it states that eyes are closed.

Step 5 – Calculate Score to Check whether Person is Drowsy

The score is basically a value we will use to determine how long the person has closed his eyes. So if both eyes are closed, we will keep on increasing score and when eyes are open, we decrease the score. We are drawing the result on the screen using cv2.putText() function which will display real time status of the person


frame
Drowsy !


frame
Active :)


frame
SLEEPING !!!