**Mini Project**

**Computer Vision & Image Processing**

**Topic: -**

**Drivers-Drowsiness-Detection**

Submitted to-

Professor Rashmi Panda

Submitted by-

1. Yukti Agarwal (2018UGEC039R)
2. Abhinav Srivastav (2018UGEC034R)
3. Punit Pushkar (2018UGEC020R)

**Driver drowsiness detection is a project built using Dlib and OpenCV with Python as a backend language.**

**Introduction -**

Feeling sleepy on a long drive is a common behavioural issue encountered in the drivers worldwide. We have an endless list of accident happened due to this issue. It’s a common human tendency to feel tired and sleepy when they are asked to do something without any body movement. Eventually due to this the body is retarded and hence we start feeling to close our eyes to replenish our energy. While driving this can prove to be catastrophic resulting in the loss of someone’s life.

The project is developed keeping the same issue in mind. We have tried to assimilate all the problems and then build a software that can actually detect whether the driver’s eyes are open or not. This will not reduce the number of accidents happening but will also provide the traffic management system a better method to evaluate the driver’s attention. The result associated with the project is accumulated after a substantial trial.

**LOGIC-**

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.

**Working-**

* The landmarks on the face 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.

**Code-**

#Importing OpenCV Library for basic image processing functions

import cv2

# Numpy for array related functions

import numpy as np

# Dlib for deep learning-based Modules and face landmark detection

import dlib

#face\_utils for basic operations of conversion

from imutils import face\_utils

#Initializing the camera and taking the instance

cap = cv2.VideoCapture(0)

#Initializing the face detector and landmark detector

detector = dlib.get\_frontal\_face\_detector()

predictor = dlib.shape\_predictor("shape\_predictor\_68\_face\_landmarks.dat")

#status marking for current state

sleep = 0

drowsy = 0

active = 0

status=""

color = (0,0,0)

#Function to calculate the Euclidean Distance btw two landmarks

def compute(ptA , ptB):

dist = np.linalg.norm(ptA - ptB)

return dist

def blinked(a,b,c,d,e,f):

#distance btw the vertical extreme landmarks for eyes

up = compute(b,d) + compute(c,e)

#distance btw the Horizontal extreme landmarks for eyes

down = compute(a,f)

ratio = up/(2.0\*down)

#Checking if it is blinked (Experimental Data)

if(ratio>0.25):

return 2

elif(ratio>0.21 and ratio<=0.25):

return 1

else:

return 0

while True:

\_, frame = cap.read()

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

faces = detector(gray)

#detected face in faces array

for face in faces:

x1 = face.left()

y1 = face.top()

x2 = face.right()

y2 = face.bottom()

face\_frame = frame.copy()

cv2.rectangle(face\_frame, (x1, y1), (x2, y2), (0, 255, 0), 2)

landmarks = predictor(gray, face)

landmarks = face\_utils.shape\_to\_np(landmarks)

#The numbers are actually the landmarks which will show eye

#Numbers in square brackets is 1 less than the actual valus of the landmark due to 0 based indexing of the Array

left\_blink = blinked(landmarks[36],landmarks[37],

landmarks[38], landmarks[41], landmarks[40], landmarks[39])

right\_blink = blinked(landmarks[42],landmarks[43],

landmarks[44], landmarks[47], landmarks[46], landmarks[45])

#Now judging what to do for the eye blinks

# Theincrement in each of the status(Active, Drowsy & sleepy) in the following

# conditional statement is done to produce delay to provide smoothness in display of the result

if(left\_blink==0 or right\_blink==0):

sleep+=1

drowsy=0

active=0

if(sleep>6):

status="SLEEPING !!!"

color = (255,0,0)

elif(left\_blink==1 or right\_blink==1):

sleep=0

active=0

drowsy+=1

if(drowsy>6):

status="Drowsy !"

color = (0,0,255)

else:

drowsy=0

sleep=0

active+=1

if(active>6):

status="Active :)"

color = (0,255,0)

cv2.putText(frame, status, (100,100), cv2.FONT\_HERSHEY\_SIMPLEX, 1.2, color,3)

for n in range(0, 68):

(x,y) = landmarks[n]

#landmark attributes

cv2.circle(face\_frame, (x, y), 1, (255, 255, 255), -1)

cv2.imshow("Frame", frame)

cv2.imshow("Result of detector", face\_frame)

key = cv2.waitKey(1)

if key == 27:

break

**Conclusion –**

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-the-wild.

**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 a greater 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.





