Minor Project: Driver Drowsiness detection

- Detect the driver whether the drowsy or awake, considering features like status of the eye, mouth (for yawning), different head angles while feeling sleepy. As each blink of eye cannot be considered as eye close, predict the status after analyzing multiple frames.
- Triggering alerts if the driver is sleepy (voice alerts are preferred).
- The dataset considered with different lightning conditions would be a better option.

Overview:

My Project of Driver Drowsiness Detection uses the MediaPipe library and OpenCV to perform facial landmark detection and drowsiness detection. It captures video from a webcam and processes each frame to detect facial landmarks and calculate various ratios related to the eyes and lips. Based on these ratios, the code determines if the person is drowsy or not and provides warnings accordingly.

Abstract:

Driver error and carelessness are the main contributors to today's traffic accidents. Major driver errors are most frequently caused by fatigue, alcohol, and reckless driving. This project focuses on a computer-based driver sleepiness detection system for the Intelligent Transportation System that looks for unusual behaviour.Road safety depends on driving assistance systems' capacity to assess a driver's level of alertness safety. By analyzing blink patterns and eye movements, drowsy driving can be identified early enough to prevent crashes. Accidents caused by sleepy driving are becoming more common in modern times. The majority of drivers experience low energy levels as a result of their weariness or fatigue from all the labour. As a result, people frequently experience sleepiness when driving. Accidents are much more likely to occur as a result of these fatigues. The goal of this project is to create a model that can recognise when a driver is getting sleepy or drowsy and to alert the user when this happens. The majority of high-end vehicles have these models built into the vehicle itself, but public transportation vehicles do not have access to this technology. Python is the implementation language I’ve employing for this project.

Introduction:

A growing number of vocations in today's society require long-term care. In order to respond swiftly to unanticipated situations, drivers need to keep a close eye on the road. Driver fatigue is a major factor in many traffic accidents. Technologies that can identify and warn a driver when they are in a bad psychophysical condition must be developed in order to lower the frequency of fatigue-related auto accidents. The development of such systems, however, faces many difficulties, one of which is the quick and precise assessment of a driver's fatigue symptoms. One technical choice for putting driver sleepiness detection systems into practice is the use of a vision-based technique. I describe an easy-to-use and flexible vision-based solution for bus driver monitoring. The system's modules include eye detection, eye openness estimation, fusion, sleepy measure estimation and classification. The global accident death rate has increased to 21%. This demonstrates how serious the issue is. A safe device called drowsiness detection can prevent accidents from being brought on by drivers who nod off behind the wheel. The goal of this Python project is to create a drowsiness detection model that can recognise when a driver's eyes are closed for a brief period of time.

Methodology Adopted:

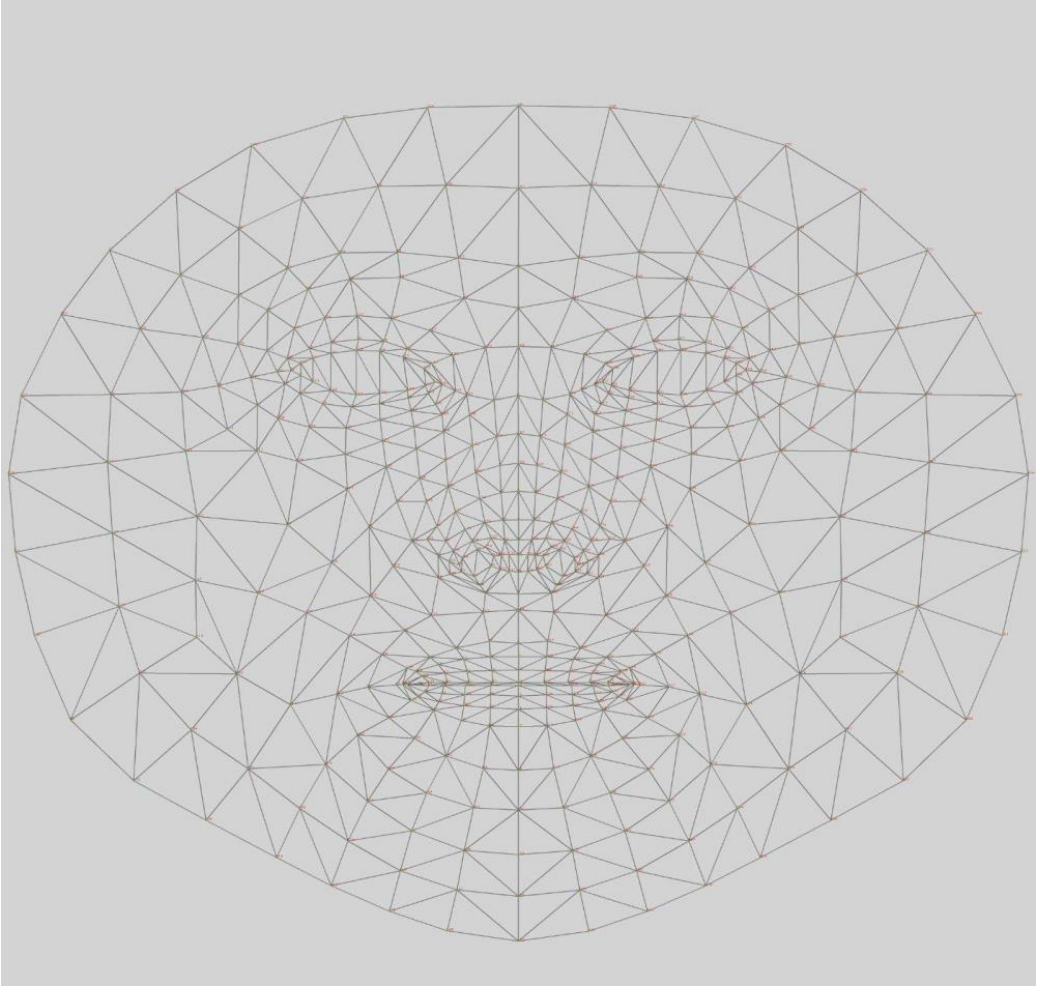
Euclidean distance

This detection can be quickly done using “sshape predictor face landmarks” that mark the essential landmarks on the face. Thus python also provides the flexibility for detecting such serious and significant detection via OpenCV modules. One can easily make their own phone camera for detecting drowsiness

Eye aspect ratio

A drowsy driver alert system that has been developed using such a technique in which the Video Stream Processing (VSP) is analyzed by eye blink concept through an Eye Aspect Ratio (EAR) and Euclidean distance of the eye. Face landmark algorithm is also used.

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



Algorithm:

1. Import the required libraries: ‘mediapipe’, ‘cv2’, ‘scipy’, ‘threading’, and ‘pyttsx3’.

2. Define helper functions:

- ‘run_speech(speech, speech_message)’: Use the ‘pyttsx3’ library to convert the provided speech message into audio and play it.
- ‘draw_landmarks(image, outputs, land_mark, color)’: Draw circles on the image at the specified landmark points using the provided color.

3. Define drowsiness detection functions:

- ‘euclidean_distance(image, top, bottom)’: Calculate the Euclidean distance between two points in the image using their coordinates.
- ‘get_aspect_ratio(image, outputs, top_bottom, left_right)’: Calculate the aspect ratio between two sets of landmarks to determine the state of eyes or lips.

4. Set constants and variables:

- Define configuration parameters such as ‘STATIC_IMAGE’, ‘MAX_NO_FACES’,
- ‘DETECTION_CONFIDENCE’, and ‘TRACKING_CONFIDENCE’.
- Define color constants for drawing landmarks.
- Define landmark lists for lips, eyes, and face.
- Initialize the ‘FaceMesh’ model with the specified configuration parameters.

5. Main loop:

- Start capturing video frames from the webcam using ‘cv.VideoCapture()’.
- Process each frame using the ‘FaceMesh’ model to detect facial landmarks.
- Draw landmarks on the image for visualization purposes.
- Calculate the aspect ratios for eyes and lips and determine drowsiness based on redefined thresholds.
- If drowsiness is detected, provide warning messages using text-to-speech.
- Display the processed image with landmarks, frame count, warnings given, eye ratios, and lip ratio using ‘cv.imshow()’ and ‘cv.putText()’.
- Continue the loop until the user presses the Esc key.

6. Clean up: Release the video capture and close all OpenCV windows.

Program:

```
# Import necessary libraries

import time

import mediapipe as mp

import cv2 as cv

from scipy.spatial import distance as dis

import threading

import pyttsx3

import asyncio

import winsdk.windows.devices.geolocation as wdg

import requests

# URL for sending Telegram messages

base_url = "https://api.telegram.org/bot5838800886:AAFEc7L2M7Qj4G6lRkVdTY7Y-eopJaGLX88/sendMessage"
```

```
# Asynchronously fetches the current coordinates

async def getCoords():

locator = wdg.Geolocator()

pos = await locator.get_geoposition_async()

return [pos.coordinate.latitude, pos.coordinate.longitude]

# Obtains the current location coordinates

def getLoc():

try:

return asyncio.run(getCoords())

except Exception as e:

print(e)

print("ERROR: You need to allow applications to access your location in Windows settings")

# Parameters for Telegram API request

parameters = {

"chat_id": "2028919343",

"text": f"Found not opening Eyes for relatively Long Time at the geo-coordinate: {getLoc()}"

}

# Sends location information via Telegram API

def get_loc():

requests.get(base_url, data=parameters)

time.sleep(5)

while True:
```

```
break

# Runs speech synthesis and output

def run_speech(speech, speech_message):

speech.say(speech_message)

speech.runAndWait()

# Draws landmarks on the image

def draw_landmarks(image, outputs, land_mark, color):

height, width = image.shape[:2]

for face in land_mark:

point = outputs.multi_face_landmarks[0].landmark[face]

point_scale = ((int)(point.x * width), (int)(point.y * height))

cv.circle(image, point_scale, 2, color, 1)

# Calculates the Euclidean distance between two points in the image

def euclidean_distance(image, top, bottom):

height, width = image.shape[0:2]

point1 = int(top.x * width), int(top.y * height)

point2 = int(bottom.x * width), int(bottom.y * height)

distance = dis.euclidean(point1, point2)

return distance

# Calculates the aspect ratio based on eye landmarks

def get_aspect_ratio(image, outputs, top_bottom, left_right):

landmark = outputs.multi_face_landmarks[0]

top = landmark.landmark[top_bottom[0]]
```

```
bottom = landmark.landmark[top_bottom[1]]

top_bottom_dis = euclidean_distance(image, top, bottom)

left = landmark.landmark[left_right[0]]

right = landmark.landmark[left_right[1]]

left_right_dis = euclidean_distance(image, left, right)

aspect_ratio = left_right_dis / top_bottom_dis

return aspect_ratio

# Configuration parameters

face_mesh = mp.solutions.face_mesh

draw_utils = mp.solutions.drawing_utils

landmark_style = draw_utils.DrawingSpec((0, 255, 0), thickness=1, circle_radius=1)

connection_style = draw_utils.DrawingSpec((0, 0, 255), thickness=1, circle_radius=1)

STATIC_IMAGE = False

MAX_NO_FACES = 2

DETECTION_CONFIDENCE = 0.6

TRACKING_CONFIDENCE = 0.5

COLOR_RED = (0, 0, 255)

COLOR_BLUE = (255, 0, 0)

COLOR_GREEN = (0, 255, 0)

LIPS = [61, 146, 91, 181, 84, 17, 314, 405, 321, 375, 291, 308, 324, 318, 402, 317, 14, 87, 178,
88, 95,

185, 40, 39, 37, 0, 267, 269, 270, 409, 415, 310, 311, 312, 13, 82, 81, 42, 183, 78]

RIGHT_EYE = [33, 7, 163, 144, 145, 153, 154, 155, 133, 173, 157, 158, 159, 160, 161, 246]
```

```
LEFT_EYE = [362, 382, 381, 380, 374, 373, 390, 249, 263, 466, 388, 387, 386, 385, 384,
398]

LEFT_EYE_TOP_BOTTOM = [386, 374]

LEFT_EYE_LEFT_RIGHT = [263, 362]

RIGHT_EYE_TOP_BOTTOM = [159, 145]

RIGHT_EYE_LEFT_RIGHT = [133, 33]

UPPER_LOWER_LIPS = [13, 14]

LEFT_RIGHT_LIPS = [78, 308]

FACE = [10, 338, 297, 332, 284, 251, 389, 356, 454, 323, 361, 288, 397, 365, 379, 378, 400,

377, 152, 148, 176, 149, 150, 136, 172, 58, 132, 93, 234, 127, 162, 21, 54, 103, 67, 109]

# Initialize the face mesh model

face_model = face_mesh.FaceMesh(static_image_mode=STATIC_IMAGE,

max_num_faces=MAX_NO_FACES,

min_detection_confidence=DETECTION_CONFIDENCE,

min_tracking_confidence=TRACKING_CONFIDENCE)

# Initialize the video capture

capture = cv.VideoCapture(0)

frame_count = 0

unconcious_count = 0

warnings_given = 0

min_frame = 6

min_tolerance = 5.0

# Initialize the speech synthesis engine

speech = pyttsx3.init()
```

```
while True:

result, image = capture.read()

if result:

image_rgb = cv.cvtColor(image, cv.COLOR_BGR2RGB)

outputs = face_model.process(image_rgb)

if outputs.multi_face_landmarks:

# Draw face landmarks

draw_landmarks(image, outputs, FACE, COLOR_GREEN)

# Draw left eye landmarks

draw_landmarks(image, outputs, LEFT_EYE_TOP_BOTTOM, COLOR_RED)

draw_landmarks(image, outputs, LEFT_EYE_LEFT_RIGHT, COLOR_RED)

# Calculate left eye aspect ratio

ratio_left = get_aspect_ratio(image, outputs, LEFT_EYE_TOP_BOTTOM,
LEFT_EYE_LEFT_RIGHT)

# Draw right eye landmarks

draw_landmarks(image, outputs, RIGHT_EYE_TOP_BOTTOM, COLOR_RED)

draw_landmarks(image, outputs, RIGHT_EYE_LEFT_RIGHT, COLOR_RED)

# Calculate right eye aspect ratio

ratio_right = get_aspect_ratio(image, outputs, RIGHT_EYE_TOP_BOTTOM,
RIGHT_EYE_LEFT_RIGHT)

# Calculate average aspect ratio

ratio = (ratio_left + ratio_right) / 2.0

if ratio > min_tolerance:

frame_count += 1
```

```
unconcious_count += 1

else:

frame_count = 0

unconcious_count = 0

if frame_count > min_frame:

warnings_given += 1

message = 'Drowsy Alert: It seems you are sleeping. Please wake up.'

t = threading.Thread(target=run_speech, args=(speech, message))

t.start()

if unconcious_count > 250:

telegram_send = threading.Thread(target=get_loc)

telegram_send.start()

# Draw lip landmarks

draw_landmarks(image, outputs, UPPER_LOWER_LIPS, COLOR_BLUE)

draw_landmarks(image, outputs, LEFT_RIGHT_LIPS, COLOR_BLUE)

# Calculate lip aspect ratio

ratio_lips = get_aspect_ratio(image, outputs, UPPER_LOWER_LIPS,
LEFT_RIGHT_LIPS)

if ratio_lips < 1.8:

message = 'Drowsy Warning: Please focus on the road.'

p = threading.Thread(target=run_speech, args=(speech, message))

p.start()

# Draw text on the image

font = cv.FONT_HERSHEY_SIMPLEX
```

```
font_size = 1.1

font_color = (255, 255, 255)

font_thickness = 2

cv.putText(image, f"Frame Count: {frame_count}", (30, 30), font, font_size, font_color, font_thickness, cv.LINE_AA)

cv.putText(image, f"Warnings Given: {warnings_given}", (100, 30), font, font_size, font_color, font_thickness, cv.LINE_AA)

cv.putText(image, f"Unconscious Count: {unconcious_count}", (560, 30), font, font_size, font_color, font_thickness, cv.LINE_AA)

cv.putText(image, f"Eye Aspect Ratio: {round(ratio, 3)}", (30, 60), font, font_size, font_color, font_thickness, cv.LINE_AA)

cv.putText(image, f"Lip Aspect Ratio: {round(ratio_lips, 3)}", (30, 90), font, font_size, font_color, font_thickness, cv.LINE_AA)

cv.imshow("FACE MESH", image)

if cv.waitKey(1) & 255 == 27:

break

capture.release()

cv.destroyAllWindows()
```

“””This code uses the MediaPipe library to perform face detection and facial landmark detection. It calculates the aspect ratios of the eyes and lips to determine if the person's eyes are closed or if they are not focusing on the road. If the aspect ratios indicate drowsiness, it gives warnings and sends a message with the geo-coordinate of the location to a specified Telegram chat using the Telegram Bot API.

Note: The code assumes that you have the necessary libraries installed and the required dependencies set up correctly. Make sure to install the required packages (MediaPipe, cv2, scipy, threading, pyttsx3, asyncio, winsdk, and requests) before running the code.”””

Code Description :

The code starts by importing the necessary libraries, including OpenCV (cv2), MediaPipe, scipy, threading, pyttsx3, asyncio, winsdk, and requests.

1. Next, several constants are defined, including the facial landmark indices for different facial features such as eyes, lips, and face. These indices are used to extract the landmarks from the face mesh model.

2. The code initializes the face mesh model using the FaceMesh class from the MediaPipe library. It sets the configuration parameters such as static_image_mode (whether to detect faces in static images), max_num_faces (maximum number of faces to detect), min_detection_confidence (minimum confidence score for face detection), and min_tracking_confidence (minimum confidence score for face tracking).

3. The code initializes the video capture using cv.VideoCapture(0), which captures video from the default webcam.

4. It then sets up variables for counting frames, detecting unconsciousness, and tracking the number of warnings given.

5. The code defines a function named run_speech that utilizes the pyttsx3 library to convert text to speech and speak out the given message.

6. Inside the main loop, the code reads frames from the video capture using capture.read(). It converts the frame to RGB format and processes it using the face mesh model by calling

face_model.process(image_rgb).

7. If the face mesh model detects any faces in the frame (i.e., outputs.multi_face_landmarks is not empty), the code proceeds to draw the facial landmarks on the frame using the draw_landmarks function. This function takes the image, the outputs from the face mesh model, a list of landmark indices for a specific facial feature, and a color as arguments. It uses OpenCV's cv.line function to draw lines connecting the specified landmarks, forming the shape of the facial feature.

8. The code calculates the aspect ratios of the left and right eyes using the get_aspect_ratio function. This function takes the image, the outputs from the face mesh model, a list of landmark indices representing the topbottom points of the eye, and a list of landmark indices representing the left-right points of the eye. It calculates the Euclidean distances between the

top-bottom and left-right points and computes the aspect ratio as the ratio of the horizontal distance to the vertical distance. The aspect ratios of both eyes are then averaged to obtain the overall eye aspect ratio.

9. If the eye aspect ratio is above a certain threshold (min_tolerance), it indicates drowsiness, and the frame count and unconscious count are incremented. If the frame count exceeds a certain threshold (min_frame), a warning message is given using the run_speech function, and the warnings given count is incremented.

10.If the unconscious count exceeds a certain threshold (250 frames), indicating prolonged unconsciousness, a separate thread is started to send a message with the geo-coordinate of the location using the get_loc function.

11.The code then calculates the aspect ratio of the lips using the get_aspect_ratio function, similar to the eye aspect ratio calculation. If the lip aspect ratio falls below a certain threshold (1.8), indicating the person is not focusing on the road, a warning message is given using the run_speech function.

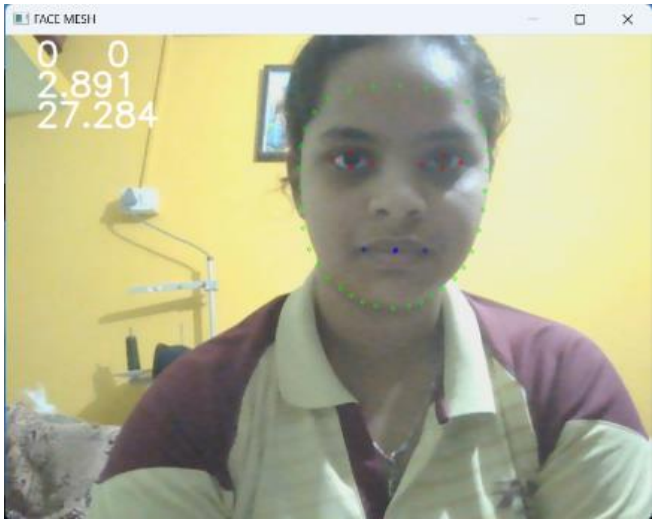
12.Textual information, including frame count, warnings given, unconscious count, eye aspect ratio, and lip aspect ratio, is drawn on the frame using OpenCV's cv.putText function.

13.The processed frame with the drawn landmarks and text is displayed using cv.imshow.

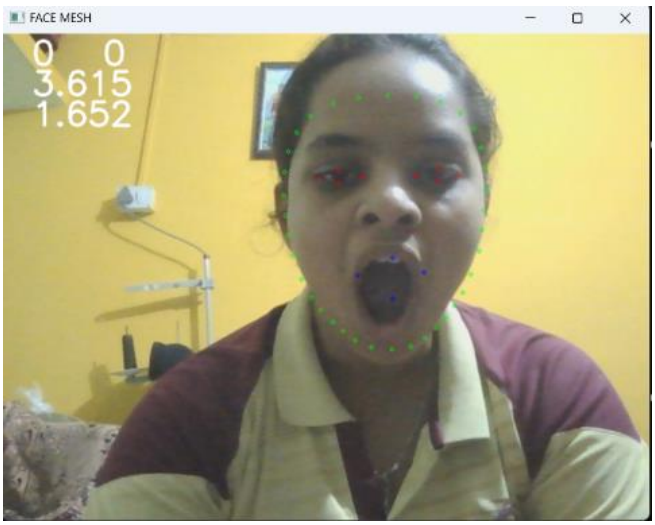
14.The code checks for the 'Esc' key press (`cv.waitKey(1) & 255 == 27

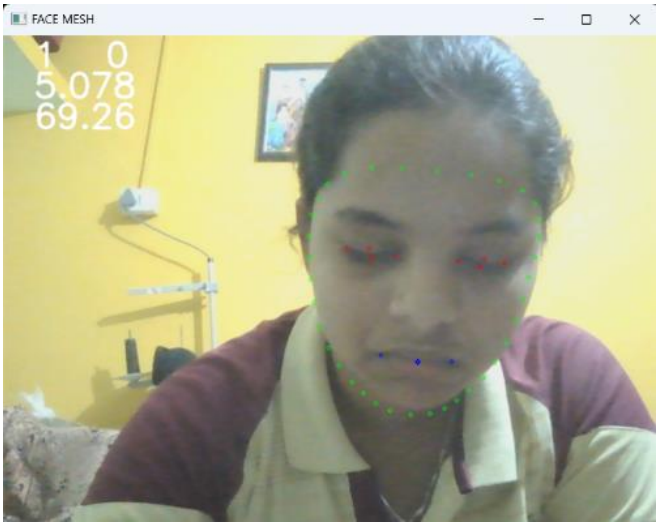
Result:

a)



b)





c)

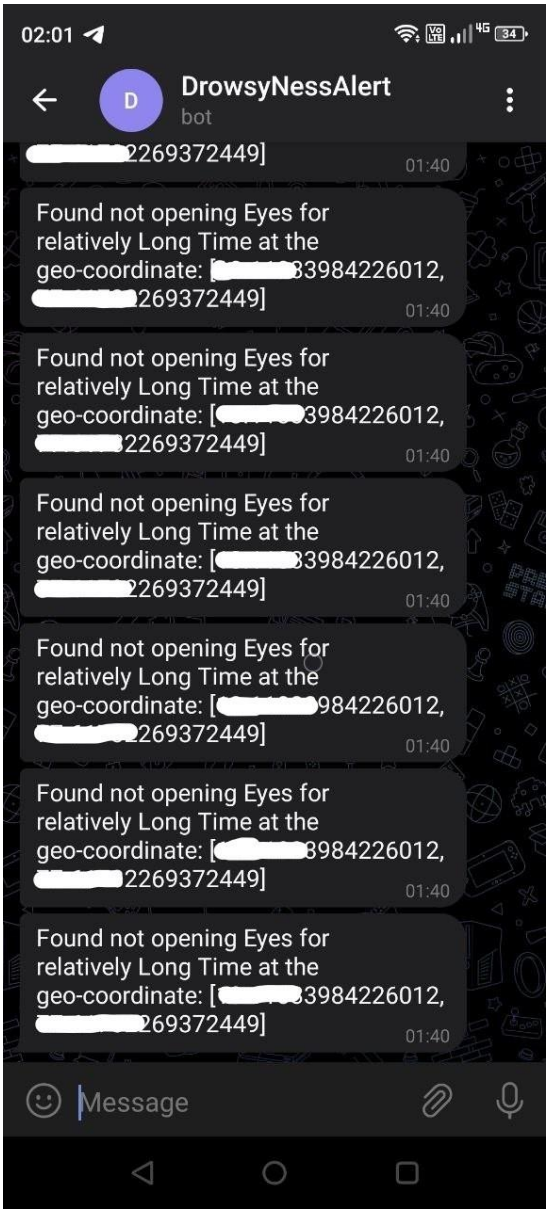


Figure a: Normal Scenario

Figure b: With Mouth Open (Drowsiness)

Figure c: With eyes closed (Sleep)

Figure d: Alert system for closing the eyes for relatively long time to police network and close members

d)

Conclusion:

The model for detecting tiredness can do this by observing the driver's eye movement. To pinpoint the region of interest, the model makes use of the eye's aspect ratio. The aspect ratio of the eye can be calculated using the EAR function. The alert is generated if the detection counter value exceeds the driver code-set threshold value. The main objective of this project's development is to lessen the amount of collisions brought on by fatigued drivers.

The newly designed driver abnormality monitoring system is capable of quickly spotting signs of intoxication, exhaustion, and unsafe driving habits. Drowsy driving-related accidents may be prevented using the proposed technology. If the camera generates a greater resolution, even if the driver wears glasses, the device works well in low light conditions. Various self-developed image processing methods are used to collect information about the

head and eye positions. The technology can determine if the eyes are open or closed during the monitoring. A warning signal is given when the eyes are closed for an extended period of time. Continuous eye closures are used to determine the driver's alertness state.

References:

[1] https://github.com/google/mediapipe

The official documentation for the MediaPipe library, which provides various cross-platform solutions for perception tasks, including face detection and tracking.

[2] https://developers.google.com/mediapipe/solutions/vision/face_landmarker/

The MediaPipe Face Landmarker task lets you detect face landmarks and facial expressions in images and videos

[3] https://docs.opencv.org/

OpenCV is a popular computer vision library that provides tools and functions for image and video processing.

[4] https://pyttsx3.readthedocs.io/en/latest/ pyttsx3 is a Python library for text-to-speech conversion.

[5] https://youtu.be/uIcKIRtSdCs Drowsiness Detection, using Mediapipe FaceMesh a video reference.

[6] https://github.com/sant-elam/DrowsinessDetection

[7] https://github.com/sant-elam/FaceMesh

Thank you.

Submitted by :

Name: Gaganasri M N

Subject Line: ITP-MINOR-JUNE.

gaganasri0112@gmail.com