**TRAVEL MANAGEMNT SYSTEM**

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **CHAPTER NO** | **CHAPTER NAME** | **PAGE NO** |
| 1 | **Abstract** |  |
| 2 | **problem Definition & Description** |  |
| 3 | **SYSTEM Environment**  3.1 Hardware Requirement  3.2 Software Requirement |  |
| 4 | **SYSTEM STUDY AND ANALYSIS**  4.1 Existing System  4.2 proposed system  Frontend  Backend |  |
| 5 | **SYSTEM DESIGN**  5.1 Data flow diagram  5.2 database design |  |
| 6 | **SYSTEM TESTING** |  |
| 7 | **MODULES** |  |
| 8 | **OUTPUT DESIGN** |  |
| 9 | **PROJECT CODE** |  |
|  |

**CHAPTER – 1**

**ABSTRACT**

Driver Sleeping Detection is program to find when the driver of vehicle is

falling a sleep.

This program can effect to over night drivers and long ride driver.

This program prevent most of the accident happened by the driver who

fell in asleep.

This alert the driver when he fell asleep by detecting the driver’s eye

actions.

It uses python library file Mediapipe to detect the eye actions

**CHAPTER - 2**

**PROBLEM DEFINITION & DESCRIPTION**

Drowsy driving stands out as a significant contributor to road accidents worldwide, leading to numerous injuries and fatalities. In response to this pressing issue, the Driver Drowsiness Detection System emerges as a proactive solution designed to identify and address instances of driver drowsiness. By leveraging advanced technologies such as facial landmark detection and eye tracking, this system aims to detect signs of drowsiness in real-time, allowing for timely interventions to prevent potential accidents. Through its implementation, the system endeavors to enhance road safety by mitigating the risks associated with drowsy driving therebysafeguarding the well-being of drivers and passengers alike.

**CHAPTER - 3**

**SYSTEM ANALYSIS**

**3.1 EXISTING SYSTEM**

By using a non intrusive machine vision based concepts, drowsiness of the driver detected system is developed. Many existing systems require a camera which is installed in front of driver. It points straight towards the face of the driver and monitors the driver‟s eyes in order to identify the drowsiness. For large vehicle such as heavy trucks and buses this arrangement is not pertinent. Bus has a large front glass window to have a broad view for safe driving. If we place a camera on the window of front glass, the camera blocks the frontal view of driver so it is not practical. If the camera is placed on the frame which is just about the window, then the camera is unable to

detain the anterior view of the face of the driver correctly

**3.1.1 DISADVANTAGES**

1. The systems main component is a small camera pointing towards the driver's face scan andmonitoring the driver's eyes to detect drowsiness.

2.Most of the existing systems use external factors and inform the user about the problem and save users after an accident is accord but from research most of the accidents are due to faults in users like drowsiness, sleeping while driving.

3.These methods cant able to detect the facial expressions, yawning head nods, and majorly on eye blink frequencies.

4.Accuracy of the existing method is not good when compared to the proposed mode.

5.Difficult to detect drunken people.

6.it will be problematic when multiple faces detected to camera.

**3.2 PROPOSED SYSTEM**

System is developed for driver drowsiness detection to prevent accidents from happening because of driver fatigue and sleepiness. 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 ide a of how the system works and what changes can be done in order to improve the utility of the overall system.

**3.2.1 ADVANTAGES**

1.Eye closure period for lazy drivers are longer than ordinary blinking. It is additionally exceptionally small longer time may result in extreme crash. So, we'll caution the driver immediately as closed eye is identified.

2.This method will detect a problem before any problem accord and inform the driver and other passengers by raising an alert.

3.In this OpenCV based machine learning techniques are used for automatic detection of drowsiness.

4.Automatic driver mode is started.

5.This system will lead to reduce the road accidents.

6.This system accurately detects the drowsiness.

7.Implementation can be done without using database storage so it will fetch the values without taking any buffering time

**CHAPTER – 4**

**SYSTEM REQUIREMENTS**

**4.1 HARDWARE REQUIREMENT**

The hardware used for the development of the project:

* **PROCESSOR** – Intel(R) Core (TM) i3-5005U
* **RAM** – 4 GB RAM
* **SYSTEM TYPE** – 64-bit OS
* **HARD DISK –** 500GB

**4.2 SOFTWARE REQUIREMENT**

**PYTHON LIBRARIES**

**OpenCV:** Functionality: Utilized for video processing, particularly for reading video streams from webcams.

Installation: pip install opencv-python

**Pandas:** Functionality: Essential for dataset management, providing tools for data preprocessing and analysis.

Installation: pip install pandas

**Date:** Functionality: Provides functionalities for managing date and time data within the system. Installation: No separate installation required as it is part of Python's standard library.

**Mediapipe:** Functionality: Integral for facial landmark detection and eye tracking, enabling real-time analysis of facial features.

Installation: pip install mediapipe

**Utils:** Functionality: Contains custom functions and classes tailored for specific operations within the system.

Installation: No separate installation required as it is part of the project's codebase.

**NumPy**: Functionality: Enables mathematical operations and array manipulation, enhancing computational efficiency.

Installation: pip install numpy

**Playsound:** Functionality: Used for playing audio alerts or warnings when drowsiness is detected.

Installation: pip install playsound

**Pyttsx3**: Functionality: Employs text-to-speech conversion to audibly communicate with the driver.

Installation: pip install pyttsx3

**OPENCV**

OpenCV, or Open Source Computer Vision Library, serves as the backbone of countless computer vision projects worldwide. Its extensive array of functions and algorithms provides developers with the tools needed for tasks ranging from basic image manipulation to advanced object detection and recognition. From reading and writing images to complex video analysis, OpenCV offers a comprehensive suite of tools for nearly every aspect of computer vision development. Its versatility, efficiency, and extensive documentation make it an indispensable resource for both beginners and experts in the field.

At the heart of OpenCV lies a vast collection of modules and libraries catering to various aspects of computer vision. These modules include functionalities for image processing, feature detection, machine learning, and more. Developers can harness the power of OpenCV's core functions, such as image loading, resizing, and color space conversions, to manipulate images with ease. Moreover, specialized modules like cv2.dnn for deep neural network inference and cv2.VideoCapture for video processing enable the implementation of cutting-edge computer vision applications.

OpenCV's continuous development and community support ensure that it remains at the forefront of computer vision innovation. With each release, new features and optimizations are introduced, expanding its capabilities and improving performance. Whether it's for academic research, industrial applications, or hobbyist projects, OpenCV empowers developers to explore the vast potential of computer vision and push the boundaries of what's possible in this dynamic field.

**MEDIAPIPE**

MediaPipe is a powerful framework developed by Google that provides a wide range of tools and pipelines for building real-time machine learning-based solutions in various domains such as augmented reality, gesture recognition, and object detection. Leveraging the capabilities of MediaPipe alongside OpenCV, developers gain access to a comprehensive set of computer vision functionalities. OpenCV, renowned for its versatility and efficiency, complements MediaPipe by offering robust image processing, feature detection, and object recognition capabilities. Through seamless integration, developers can harness the strengths of both frameworks to create sophisticated computer vision applications with ease.

Combining MediaPipe with OpenCV empowers developers to tackle complex computer vision challenges with confidence. OpenCV's extensive library of algorithms for image manipulation, feature extraction, and machine learning seamlessly integrates with MediaPipe's pipelines, providing a unified platform for rapid development and prototyping. Whether it's building custom pose estimation models, implementing real-time hand tracking, or deploying object detection solutions, the synergy between MediaPipe and OpenCV offers unparalleled flexibility and performance, enabling developers to push the boundaries of what's possible in computer vision.

In summary, the integration of MediaPipe with OpenCV forms a formidable toolkit for developers seeking to build sophisticated computer vision applications. By harnessing the strengths of both frameworks, developers can streamline the development process, leverage state-of-the-art machine learning models, and deliver high-performance solutions across a diverse range of domains. Whether it's developing interactive AR experiences, implementing precise hand gesture recognition, or deploying efficient object detection systems, the combined power of MediaPipe and OpenCV unlocks new possibilities for innovation and creativity in the field of computer vision..

**UTILS**

Incorporating utilities libraries alongside OpenCV enhances the robustness and efficiency of computer vision projects. These utility libraries, often referred to as "utils," offer a variety of helpful tools and functions that complement OpenCV's capabilities. By integrating utils with OpenCV, developers can streamline common tasks such as data loading, preprocessing, and visualization, thereby accelerating the development process. Additionally, utils libraries often provide convenient functions for handling file I/O, logging, and debugging, further enhancing the productivity and maintainability of OpenCV-based projects.

The integration of utils libraries with OpenCV also facilitates code organization and modularity, leading to cleaner and more maintainable implementations. Developers can leverage utils to encapsulate reusable functions and components, promoting code reuse and scalability across different parts of their projects. This modular approach enhances code readability and facilitates collaboration among team members, ultimately contributing to more efficient and sustainable development practices.

Moreover, utils libraries often offer specialized functionalities that address common challenges encountered in computer vision applications. Whether it's geometric transformations, image augmentation, or performance optimization, utils can provide tailored solutions that complement and extend the capabilities of OpenCV. By harnessing the combined power of OpenCV and utils libraries, developers can tackle complex computer vision tasks with confidence, enabling them to build robust and scalable solutions that meet the demands of diverse application domains.

**CHAPTER – 5**

**SYSTEM DESIGN**

5.



**CHAPTER – 6**

**SYSTEM TESTING**

***1. Functional Testing:-***

**Feature Testing:-**Verify that all features of the driver drowsiness detection system, such as face detection, eye tracking, and drowsiness detection algorithms, are functioning correctly.

**Scenario Testing:-** Test the system under various scenarios, including different lighting conditions, driver positions, and distractions, to ensure accurate detection.

**Alarm Trigger Testing:-** Validate that alarms or alerts are triggered appropriately when signs of drowsiness are detected.

***2. Integration Testing:-***

**MediaPipe Integration:-** Verify the seamless integration of MediaPipe components for tasks such as face detection, landmark detection, and facial recognition within the driver drowsiness detection system.

**Sensor Integration:-** Test the integration of external sensors, such as cameras or infrared sensors, with MediaPipe components to ensure accurate data input for drowsiness detection.

***3. Usability Testing:-***

**User Interface Testing:-** Evaluate the user interface of the system to ensure it is intuitive and user-friendly for drivers.

**Response Time Testing:-** Measure the system's response time to ensure it provides timely alerts to prevent accidents effectively.

**Accuracy Testing:-** Assess the accuracy of the system in detecting drowsiness compared to manual observation or other benchmark methods.

***4. Performance Testing:-***

**Load Testing:-** Assess the system's performance under different loads to ensure it can handle concurrent processing of multiple video streams in real-time.

**Resource Utilization Testing:-** Measure the system's resource utilization, including CPU and memory usage, to ensure optimal performance without resource bottlenecks.

**Stability Testing:-** Verify the stability of the system over extended periods of operation to ensure reliability under continuous usage.

***5. Security Testing:-***

**Data Privacy Testing:-** Ensure that the system handles sensitive data, such as facial images or biometric data, securely and complies with privacy regulations.

**Access Control Testing:-** Test the system's access controls to prevent unauthorized access and ensure the integrity of the drowsiness detection process.

6. ***Acceptance Testing:-***

**End-to-End Testing:-** Perform end-to-end testing of the entire system in a simulated real-world environment to validate its effectiveness in preventing driver drowsiness and ensuring road safety.

**User Acceptance Testing**:- Involve actual users or stakeholders to validate that the system meets their requirements and expectations effectively.

***Conclusion:-***

The testing process outlined above ensures that the driver drowsiness detection system using the MediaPipe project is thoroughly evaluated for functionality, integration, usability, performance, security, and acceptance. By following this testing process, potential issues and bugs can be identified and addressed early, resulting in a reliable and effective solution for enhancing road safety.

**CHAPTER – 7**

**MODULES**

**1.Data Capture Module:-**

* Utilize MediaPipe's camera interface to capture video frames from the

camera installed in the car

**2. Preprocessing Module:-**

* Implement image resizing and normalization using MediaPipe's built-in functions.
* Utilize NumPy for additional preprocessing tasks if needed.

**3. Facial Landmark Detection Module:-**

* Utilize MediaPipe's facial landmark detection module to detect facial landmarks, including eye positions.

**4. Drowsiness Detection Module:-**

* Analyze eye closure patterns and other facial features to detect signs of drowsiness.
* Utilize OpenCV and NumPy for image processing and feature extraction tasks if needed.

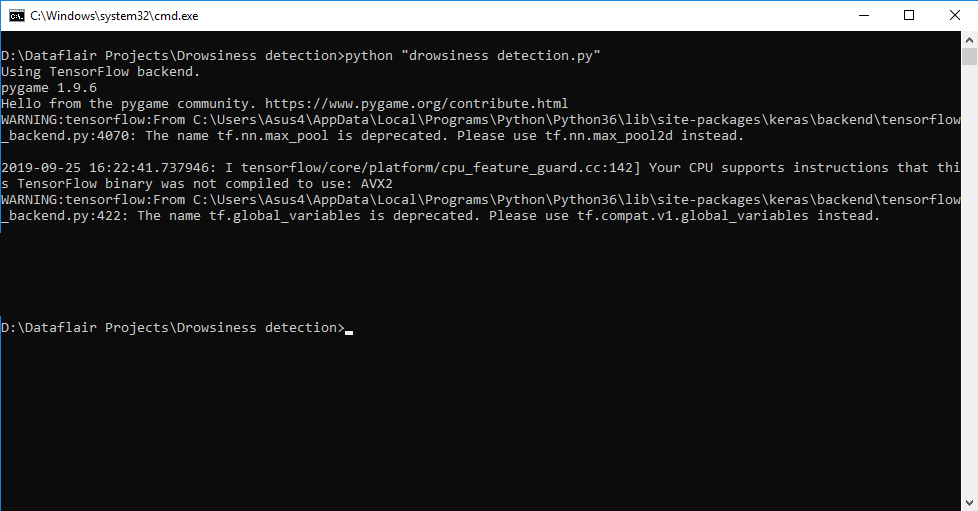
**5. Alerting Module:-**

* Implement sound alerts and visual warnings using MediaPipe's built-in audio and visualization capabilities.
* Utilize playsound library for sound alerts if needed

**CHAPTER – 8**

**OUTPUT DESIGN**

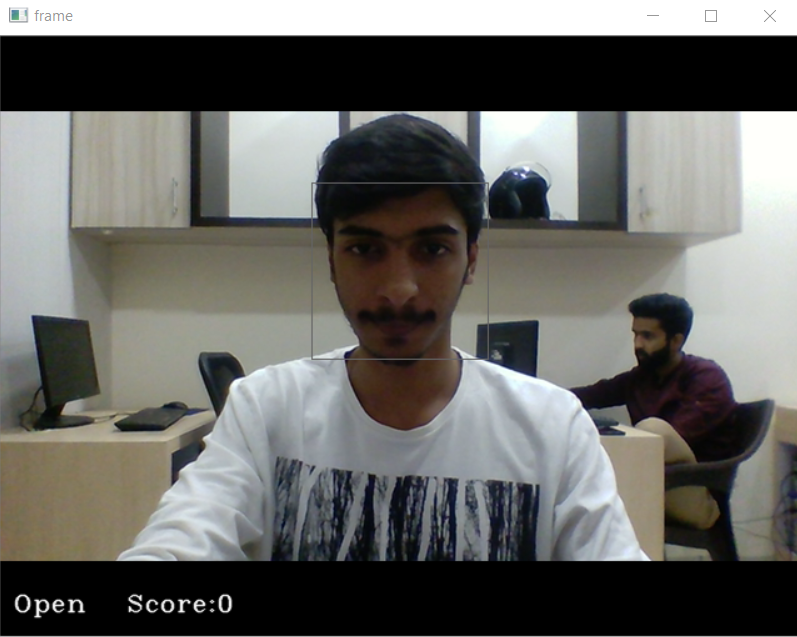
**Example Screenshot:**



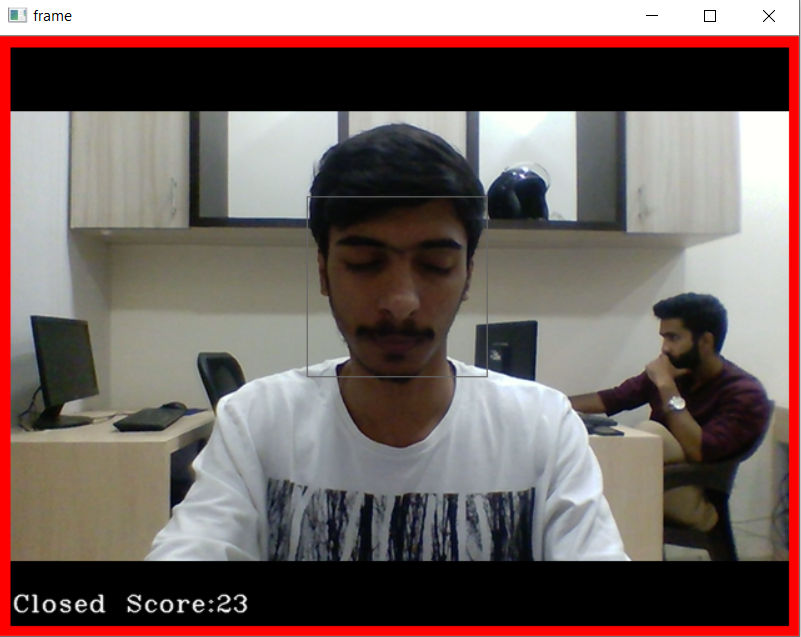


Closed Eye Detection

Open Eyes Detection



Sleep Alert



**CHAPTER – 9**

**PROJECT CODE**

**drowsiness.py**

import cv2 as cv

import mediapipe as mp

import time

import utils, math

import numpy as np

import playsound

# variables

frame\_counter = 0

CEF\_COUNTER = 0

# constants

CLOSED\_EYES\_FRAME = 30

FONTS = cv.FONT\_HERSHEY\_COMPLEX

# Left eyes indices

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

# right eyes indices

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

map\_face\_mesh = mp.solutions.face\_mesh

# camera object

camera = cv.VideoCapture(0)

# landmark detection function

def landmarksDetection(img, results, draw=False):

img\_height, img\_width = img.shape[:2]

mesh\_coord = [(int(point.x \* img\_width), int(point.y \* img\_height)) for point in

results.multi\_face\_landmarks[0].landmark]

if draw:

[cv.circle(img, p, 2, (0, 255, 0), -1) for p in mesh\_coord]

# returning the list of tuples for each landmarks

return mesh\_coord

# Euclaidean distance

def euclaideanDistance(point, point1):

x, y = point

x1, y1 = point1

distance = math.sqrt((x1 - x) \*\* 2 + (y1 - y) \*\* 2)

return distance

# Blinking Ratio

def blinkRatio(img, landmarks, right\_indices, left\_indices):

# Right eyes

# horizontal line

rh\_right = landmarks[right\_indices[0]]

rh\_left = landmarks[right\_indices[8]]

# vertical line

rv\_top = landmarks[right\_indices[12]]

rv\_bottom = landmarks[right\_indices[4]]

# LEFT\_EYE

# horizontal line

lh\_right = landmarks[left\_indices[0]]

lh\_left = landmarks[left\_indices[8]]

# vertical line

lv\_top = landmarks[left\_indices[12]]

lv\_bottom = landmarks[left\_indices[4]]

rhDistance = euclaideanDistance(rh\_right, rh\_left)

rvDistance = euclaideanDistance(rv\_top, rv\_bottom)

lvDistance = euclaideanDistance(lv\_top, lv\_bottom)

lhDistance = euclaideanDistance(lh\_right, lh\_left)

reRatio = rhDistance / rvDistance

leRatio = lhDistance / lvDistance

ratio = (reRatio + leRatio) / 2

return ratio

with map\_face\_mesh.FaceMesh(min\_detection\_confidence=0.5, min\_tracking\_confidence=0.5) as face\_mesh:

# starting time here

start\_time = time.time()

# starting Video loop here.

while True:

frame\_counter += 1 # frame counter

ret, frame = camera.read() # getting frame from camera

if not ret:

break # no more frames break

# resizing frame

frame = cv.resize(frame, None, fx=1.5, fy=1.5, interpolation=cv.INTER\_CUBIC)

frame\_height, frame\_width = frame.shape[:2]

rgb\_frame = cv.cvtColor(frame, cv.COLOR\_RGB2BGR)

results = face\_mesh.process(rgb\_frame)

if results.multi\_face\_landmarks:

mesh\_coords = landmarksDetection(frame, results, False)

ratio = blinkRatio(frame, mesh\_coords, RIGHT\_EYE, LEFT\_EYE)

utils.colorBackgroundText(frame, f'Ratio : {round(ratio, 2)}', FONTS, 0.7, (30, 100), 2, utils.PINK,

utils.YELLOW)

if ratio > 5:

CEF\_COUNTER += 1

if CEF\_COUNTER <= 3:

utils.colorBackgroundText(frame, f'1', FONTS, 5, (430, 200), 6, utils.WHITE, utils.RED, pad\_x=4,

pad\_y=6, )

elif CEF\_COUNTER <= 20:

utils.colorBackgroundText(frame, f'2', FONTS, 5, (430, 200), 6, utils.WHITE, utils.RED, pad\_x=4,

pad\_y=6, )

elif CEF\_COUNTER <= CLOSED\_EYES\_FRAME:

utils.colorBackgroundText(frame, f'3', FONTS, 5, (430, 200), 6, utils.WHITE, utils.RED, pad\_x=4,

pad\_y=6, )

if CEF\_COUNTER > CLOSED\_EYES\_FRAME:

utils.colorBackgroundText(frame, f'Drowsiness Alert...', FONTS, 1.7, (250, 150),

2, utils.YELLOW, pad\_x=4, pad\_y=6, )

playsound.playsound(r'C:\Users\91822\Pictures\MediaPipe\warning.mp3')

else:

CEF\_COUNTER = 0

cv.polylines(frame, [np.array([mesh\_coords[p] for p in LEFT\_EYE], dtype=np.int32)], True, utils.GREEN, 1,

cv.LINE\_AA)

cv.polylines(frame, [np.array([mesh\_coords[p] for p in RIGHT\_EYE], dtype=np.int32)], True, utils.GREEN, 1,

cv.LINE\_AA)

# calculating frame per seconds FPS

end\_time = time.time() - start\_time

fps = frame\_counter / end\_time

frame = utils.textWithBackground(frame, f'FPS: {round(fps, 1)}', FONTS, 1.0, (30, 50), bgOpacity=0.9,

textThickness=2)

frame = cv.resize(frame, (1280, 720))

cv.imshow('frame', frame)

key = cv.waitKey(2)

if key == ord('q') or key == ord('Q'):

break

cv.destroyAllWindows()

camera.release()

Main.py

import cv2 as cv

import mediapipe as mp

import time

import utils, math

import numpy as np

# variables

frame\_counter = 0

CEF\_COUNTER = 0

TOTAL\_BLINKS = 0

# constants

CLOSED\_EYES\_FRAME = 3

FONTS = cv.FONT\_HERSHEY\_COMPLEX

# face bounder indices

FACE\_OVAL = [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]

# lips indices for Landmarks

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]

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

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

# Left eyes indices

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

LEFT\_EYEBROW = [336, 296, 334, 293, 300, 276, 283, 282, 295, 285]

# right eyes indices

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

RIGHT\_EYEBROW = [70, 63, 105, 66, 107, 55, 65, 52, 53, 46]

map\_face\_mesh = mp.solutions.face\_mesh

# camera object

camera = cv.VideoCapture(1)

# landmark detection function

def landmarksDetection(img, results, draw=False):

img\_height, img\_width = img.shape[:2]

# list[(x,y), (x,y)....]

mesh\_coord = [(int(point.x \* img\_width), int(point.y \* img\_height)) for point in

results.multi\_face\_landmarks[0].landmark]

if draw:

[cv.circle(img, p, 2, (0, 255, 0), -1) for p in mesh\_coord]

# returning the list of tuples for each landmarks

return mesh\_coord

# Euclaidean distance

def euclaideanDistance(point, point1):

x, y = point

x1, y1 = point1

distance = math.sqrt((x1 - x) \*\* 2 + (y1 - y) \*\* 2)

return distance

# Blinking Ratio

def blinkRatio(img, landmarks, right\_indices, left\_indices):

# Right eyes

# horizontal line

rh\_right = landmarks[right\_indices[0]]

rh\_left = landmarks[right\_indices[8]]

# vertical line

rv\_top = landmarks[right\_indices[12]]

rv\_bottom = landmarks[right\_indices[4]]

# draw lines on right eyes

# cv.line(img, rh\_right, rh\_left, utils.GREEN, 2)

# cv.line(img, rv\_top, rv\_bottom, utils.WHITE, 2)

# LEFT\_EYE

# horizontal line

lh\_right = landmarks[left\_indices[0]]

lh\_left = landmarks[left\_indices[8]]

# vertical line

lv\_top = landmarks[left\_indices[12]]

lv\_bottom = landmarks[left\_indices[4]]

rhDistance = euclaideanDistance(rh\_right, rh\_left)

rvDistance = euclaideanDistance(rv\_top, rv\_bottom)

lvDistance = euclaideanDistance(lv\_top, lv\_bottom)

lhDistance = euclaideanDistance(lh\_right, lh\_left)

reRatio = rhDistance / rvDistance

leRatio = lhDistance / lvDistance

ratio = (reRatio + leRatio) / 2

return ratio

with map\_face\_mesh.FaceMesh(min\_detection\_confidence=0.5, min\_tracking\_confidence=0.5) as face\_mesh:

# starting time here

start\_time = time.time()

# starting Video loop here.

while True:

frame\_counter += 1 # frame counter

ret, frame = camera.read() # getting frame from camera

if not ret:

break # no more frames break

# resizing frame

frame = cv.resize(frame, None, fx=1.5, fy=1.5, interpolation=cv.INTER\_CUBIC)

frame\_height, frame\_width = frame.shape[:2]

rgb\_frame = cv.cvtColor(frame, cv.COLOR\_RGB2BGR)

results = face\_mesh.process(rgb\_frame)

if results.multi\_face\_landmarks:

mesh\_coords = landmarksDetection(frame, results, False)

ratio = blinkRatio(frame, mesh\_coords, RIGHT\_EYE, LEFT\_EYE)

# cv.putText(frame, f'ratio {ratio}', (100, 100), FONTS, 1.0, utils.GREEN, 2)

utils.colorBackgroundText(frame, f'Ratio : {round(ratio, 2)}', FONTS, 0.7, (30, 100), 2, utils.PINK,

utils.YELLOW)

if ratio > 5.5:

CEF\_COUNTER += 1

# cv.putText(frame, 'Blink', (200, 50), FONTS, 1.3, utils.PINK, 2)

utils.colorBackgroundText(frame, f'Blink', FONTS, 1.7, (int(frame\_height / 2), 100), 2, utils.YELLOW,

pad\_x=6, pad\_y=6, )

else:

if CEF\_COUNTER > CLOSED\_EYES\_FRAME:

TOTAL\_BLINKS += 1

CEF\_COUNTER = 0

# cv.putText(frame, f'Total Blinks: {TOTAL\_BLINKS}', (100, 150), FONTS, 0.6, utils.GREEN, 2)

utils.colorBackgroundText(frame, f'Total Blinks: {TOTAL\_BLINKS}', FONTS, 0.7, (30, 150), 2)

cv.polylines(frame, [np.array([mesh\_coords[p] for p in LEFT\_EYE], dtype=np.int32)], True, utils.GREEN, 1,

cv.LINE\_AA)

cv.polylines(frame, [np.array([mesh\_coords[p] for p in RIGHT\_EYE], dtype=np.int32)], True, utils.GREEN, 1,

cv.LINE\_AA)

# calculating frame per seconds FPS

end\_time = time.time() - start\_time

fps = frame\_counter / end\_time

frame = utils.textWithBackground(frame, f'FPS: {round(fps, 1)}', FONTS, 1.0, (30, 50), bgOpacity=0.9,

textThickness=2)

# writing image for thumbnail drawing shape

# cv.imwrite(f'img/frame\_{frame\_counter}.png', frame)

cv.imshow('frame', frame)

key = cv.waitKey(2)

if key == ord('q') or key == ord('Q'):

break

cv.destroyAllWindows()

camera.release()

Utlis.py

import cv2 as cv

import numpy as np

# colors

# values =(blue, green, red) opencv accepts BGR values not RGB

BLACK = (0, 0, 0)

WHITE = (255, 255, 255)

BLUE = (255, 0, 0)

RED = (0, 0, 255)

CYAN = (255, 255, 0)

YELLOW = (0, 255, 255)

MAGENTA = (255, 0, 255)

GRAY = (128, 128, 128)

GREEN = (0, 255, 0)

PURPLE = (128, 0, 128)

ORANGE = (0, 165, 255)

PINK = (147, 20, 255)

points\_list = [(200, 300), (150, 150), (400, 200)]

def drawColor(img, colors):

x, y = 0, 10

w, h = 20, 30

for color in colors:

x += w + 5

# y += 10

cv.rectangle(img, (x - 6, y - 5), (x + w + 5, y + h + 5), (10, 50, 10), -1)

cv.rectangle(img, (x, y), (x + w, y + h), color, -1)

def colorBackgroundText(img, text, font, fontScale, textPos, textThickness=1, textColor=(0, 255, 0), bgColor=(0, 0, 0),

pad\_x=3, pad\_y=3):

"""

Draws text with background, with control transparency

@param img:(mat) which you want to draw text

@param text: (string) text you want draw

@param font: fonts face, like FONT\_HERSHEY\_COMPLEX, FONT\_HERSHEY\_PLAIN etc.

@param fontScale: (double) the size of text, how big it should be.

@param textPos: tuple(x,y) position where you want to draw text

@param textThickness:(int) fonts weight, how bold it should be

@param textPos: tuple(x,y) position where you want to draw text

@param textThickness:(int) fonts weight, how bold it should be.

@param textColor: tuple(BGR), values -->0 to 255 each

@param bgColor: tuple(BGR), values -->0 to 255 each

@param pad\_x: int(pixels) padding of in x direction

@param pad\_y: int(pixels) 1 to 1.0 (), controls transparency of text background

@return: img(mat) with draw with background

"""

(t\_w, t\_h), \_ = cv.getTextSize(text, font, fontScale, textThickness) # getting the text size

x, y = textPos

cv.rectangle(img, (x - pad\_x, y + pad\_y), (x + t\_w + pad\_x, y - t\_h - pad\_y), bgColor, -1) # draw rectangle

cv.putText(img, text, textPos, font, fontScale, textColor, textThickness) # draw in text

return img

def textWithBackground(img, text, font, fontScale, textPos, textThickness=1, textColor=(0, 255, 0), bgColor=(0, 0, 0),

pad\_x=3, pad\_y=3, bgOpacity=0.5):

"""

Draws text with background, with control transparency

@param img:(mat) which you want to draw text

@param text: (string) text you want draw

@param font: fonts face, like FONT\_HERSHEY\_COMPLEX, FONT\_HERSHEY\_PLAIN etc.

@param fontScale: (double) the size of text, how big it should be.

@param textPos: tuple(x,y) position where you want to draw text

@param textThickness:(int) fonts weight, how bold it should be

@param textPos: tuple(x,y) position where you want to draw text

@param textThickness:(int) fonts weight, how bold it should be.

@param textColor: tuple(BGR), values -->0 to 255 each

@param bgColor: tuple(BGR), values -->0 to 255 each

@param pad\_x: int(pixels) padding of in x direction

@param pad\_y: int(pixels) 1 to 1.0 (), controls transparency of text background

@return: img(mat) with draw with background

"""

(t\_w, t\_h), \_ = cv.getTextSize(text, font, fontScale, textThickness) # getting the text size

x, y = textPos

overlay = img.copy() # coping the image

cv.rectangle(overlay, (x - pad\_x, y + pad\_y), (x + t\_w + pad\_x, y - t\_h - pad\_y), bgColor, -1) # draw rectangle

new\_img = cv.addWeighted(overlay, bgOpacity, img, 1 - bgOpacity, 0) # overlaying the rectangle on the image.

cv.putText(new\_img, text, textPos, font, fontScale, textColor, textThickness) # draw in text

img = new\_img

return img

def textBlurBackground(img, text, font, fontScale, textPos, textThickness=1, textColor=(0, 255, 0), kneral=(33, 33),

pad\_x=3, pad\_y=3):

"""

Draw text with background blured, control the blur value, with kernal(odd, odd)

@param img:(mat) which you want to draw text

@param text: (string) text you want draw

@param font: fonts face, like FONT\_HERSHEY\_COMPLEX, FONT\_HERSHEY\_PLAIN etc.

@param fontScale: (double) the size of text, how big it should be.

@param textPos: tuple(x,y) position where you want to draw text

@param textThickness:(int) fonts weight, how bold it should be.

@param textColor: tuple(BGR), values -->0 to 255 each

@param kneral: tuple(3,3) int as odd number: higher the value, more blurry background would be

@param pad\_x: int(pixels) padding of in x direction

@param pad\_y: int(pixels) padding of in y direction

@return: img mat, with text drawn, with background blured

call the function:

img =textBlurBackground(img, 'Blured Background Text', cv2.FONT\_HERSHEY\_COMPLEX, 0.9, (20, 60),2, (0,255, 0), (49,49), 13, 13 )

"""

(t\_w, t\_h), \_ = cv.getTextSize(text, font, fontScale, textThickness) # getting the text size

x, y = textPos

blur\_roi = img[y - pad\_y - t\_h: y + pad\_y, x - pad\_x:x + t\_w + pad\_x] # croping Text Background

img[y - pad\_y - t\_h: y + pad\_y, x - pad\_x:x + t\_w + pad\_x] = cv.blur(blur\_roi,

kneral) # merging the blured background to img

cv.putText(img, text, textPos, font, fontScale, textColor, textThickness)

# cv.imshow('blur roi', blur\_roi)

# cv.imshow('blured', img)

return img

def fillPolyTrans(img, points, color, opacity):

"""

@param img: (mat) input image, where shape is drawn.

@param points: list [tuples(int, int) these are the points custom shape,FillPoly

@param color: (tuples (int, int, int)

@param opacity: it is transparency of image.

@return: img(mat) image with rectangle draw.

"""

list\_to\_np\_array = np.array(points, dtype=np.int32)

overlay = img.copy() # coping the image

cv.fillPoly(overlay, [list\_to\_np\_array], color)

new\_img = cv.addWeighted(overlay, opacity, img, 1 - opacity, 0)

# print(points\_list)

img = new\_img

cv.polylines(img, [list\_to\_np\_array], True, color, 1, cv.LINE\_AA)

return img

# def pollyLines(img, points, color):

# list\_to\_np\_array = np.array(points, dtype=np.int32)

# cv.polylines(img, [list\_to\_np\_array], True, color,1, cv.LINE\_AA)

# return img

def rectTrans(img, pt1, pt2, color, thickness, opacity):

"""

@param img: (mat) input image, where shape is drawn.

@param pt1: tuple(int,int) it specifies the starting point(x,y) os rectangle

@param pt2: tuple(int,int) it nothing but width and height of rectangle

@param color: (tuples (int, int, int), it tuples of BGR values

@param thickness: it thickness of board line rectangle, if (-1) passed then rectangle will be fulled with color.

@param opacity: it is transparency of image.

@return:

"""

overlay = img.copy()

cv.rectangle(overlay, pt1, pt2, color, thickness)

new\_img = cv.addWeighted(overlay, opacity, img, 1 - opacity, 0) # overlaying the rectangle on the image.

img = new\_img

return img

def main():

cap = cv.VideoCapture('Girl.mp4')

counter = 0

while True:

success, img = cap.read()

# img = np.zeros((1000,1000, 3), dtype=np.uint8)

img = rectTrans(img, pt1=(30, 320), pt2=(160, 260), color=(0, 255, 255), thickness=-1, opacity=0.6)

img = fillPolyTrans(img=img, points=points\_list, color=(0, 255, 0), opacity=.5)

drawColor(img, [BLACK, WHITE, BLUE, RED, CYAN, YELLOW, MAGENTA, GRAY, GREEN, PURPLE, ORANGE, PINK])

textBlurBackground(img, 'Blured Background Text', cv.FONT\_HERSHEY\_COMPLEX, 0.8, (60, 140), 2, YELLOW, (71, 71),

13, 13)

img = textWithBackground(img, 'Colored Background Texts', cv.FONT\_HERSHEY\_SIMPLEX, 0.8, (60, 80),

textThickness=2, bgColor=GREEN, textColor=BLACK, bgOpacity=0.7, pad\_x=6, pad\_y=6)

imgGray = cv.cvtColor(img, cv.COLOR\_BGR2GRAY)

# cv.imwrite('color\_image.png', img)

counter += 1

cv.imshow('img', img)

cv.imwrite(f'image/image\_{counter}.png', img)

if cv.waitKey(1) == ord('q'):

break

if \_\_name\_\_ == "\_\_main\_\_":

main()

**CHAPTER – 9**

**CONCLUSION**

**CONCLUSION**

In conclusion, the development of a driver drowsiness detection system utilizing a combination of MediaPipe, OpenCV, Utils, and PySound represents a significant stride towards enhancing road safety. By harnessing the power of computer vision algorithms and machine learning techniques, coupled with audio alert systems, this integrated solution effectively monitors driver behavior in real-time.

The utilization of MediaPipe and OpenCV offers robust capabilities in facial landmark detection and eye tracking, enabling accurate analysis of crucial indicators of drowsiness such as eye closure and head nodding. This, combined with the versatility and efficiency of Utils for data processing and manipulation, facilitates seamless integration and efficient operation of the system.

Additionally, the incorporation of PySound for auditory alerts enhances the system's responsiveness by providing immediate feedback to the driver upon detection of drowsiness signs. This auditory cue acts as a proactive measure, alerting the driver to take necessary actions to prevent potential accidents due to fatigue.

The collaborative effort of these technologies not only ensures real-time monitoring but also allows for the customization and optimization of parameters to adapt to varying environmental conditions and individual differences in driver behavior.

In summary, the implementation of a driver drowsiness detection system using MediaPipe, OpenCV, Utils, and PySound demonstrates a practical and effective approach to mitigate the risks associated with driver fatigue. By leveraging cutting-edge technology and innovative methodologies, this system contributes significantly to enhancing road safety and ultimately saving lives.

**CHAPTER – 10**

**FUTURE ENHANCEMENT**

Future enhancements of the driver drowsiness detection system developed using MediaPipe, OpenCV, Utils, and PySound, several avenues can be explored to further improve its effectiveness and functionality. One potential enhancement could involve the integration of advanced machine learning models to enhance the accuracy and robustness of drowsiness detection algorithms. By employing deep learning techniques, such as convolutional neural networks (CNNs) or recurrent neural networks (RNNs), the system can learn more complex patterns and nuances in driver behavior, leading to better detection performance even in challenging scenarios.

Another area for improvement could be the incorporation of multi-modal sensing capabilities, such as integrating additional sensors like infrared cameras or wearable devices to complement visual and auditory cues. By combining data from multiple sources, the system can achieve a more comprehensive understanding of the driver's state, allowing for more reliable detection of drowsiness and fatigue.

Furthermore, the implementation of real-time analytics and predictive modeling techniques could enable the system to anticipate drowsiness before it becomes apparent through observable cues. By analyzing trends in driver behavior over time, the system can proactively alert the driver to take breaks or rest, thereby preventing potential accidents before they occur.

Moreover, enhancing the system's user interface and interaction design could improve user experience and acceptance. Providing customizable alert settings, adaptive feedback mechanisms, and integrating features for driver engagement, such as gamification elements or personalized coaching, could encourage drivers to actively use and benefit from the system.

Additionally, exploring opportunities for cloud connectivity and data sharing could enable collaborative efforts in analyzing aggregated driver behavior data to identify broader trends and insights related to drowsiness detection and road safety.

Overall, by embracing advancements in technology and human-computer interaction, future enhancements to the driver drowsiness detection system hold the potential to further enhance road safety and reduce the incidence of accidents caused by driver fatigue.