DRIVER DROWSINESS DETECTION SYSTEM

SHALU GUNTUPALLUY	16338636
CHARAN TEJA NEKKADUPU	16322809
SAGARIKA ABBAVARAM	16345441
MALLIKARJUNA YAGANTI	16331423

ACKNOWLEDGEMENT

Firstly, I express my heartiest thanks and gratefulness to Almighty God for His divine blessing makes it possible to complete the project work successfully.

I am grateful and wish my profound indebtedness to prof **syed jawad shah** of University of Missouri – Kansas City. His endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, and reading many inferior drafts and correcting them at all stages have made it possible to complete this project.

I would also generously welcome each one of those individuals who have helped me straightforwardly or in a roundabout way in making this project a win. In this unique situation, I might want to thank the various staff individuals, both educating and non-instructing, which have developed their convenient help and facilitated my undertaking.

CHAPTER 1

INTRODUCTION

Problem Definition

Road safety is one of the main objectives in designing driver assistance systems. On average, every 30 s, one person dies somewhere in the world due to a car crash. The cost of accidents in the United States is estimated to be about $300 billion annually, i.e., about 2% of its gross domestic product. Conservative estimates suggest that a high proportion of fatalities and injuries due to traffic accidents involve impaired drivers. It is projected that these figures could be increased by 65% in the next 20 years, unless novel driving risk reduction methods are leveraged. Among all fatal traffic accidents, 95% are caused by human errors.

The three major causes of these human errors, which are often referred to as the “Big Three,” are alcohol, drowsiness, and inattention. Statistics show that 25% of fatal accidents in Europe, 32% in the United States, and 38% in Canada are caused by drunk drivers. The National Highway Traffic Safety Administration (NHTSA) reports that over 3000 fatalities from automobile accidents are caused by drowsiness or distraction.

Drowsiness is common during travel. Driver will either resist the drowsiness or stop the vehicle aside and continue the journey after a while. But sometimes driver may be unaware that they are actually feeling fatigue and it is the cause for major accidents. To avoid this, first driver fatigue has to be identified and alerted. Some external factors like dusts can cause irritation in eye and divert a driver during travel. Even an experienced driver can also be diverted from driving which eventually leads to an accident. So driver distractions during travel also have to beidentified and alerted.

Project Overview

To prevent traffic accidents caused by sleepy drivers, many systems have been proposed. These systems use different type of sensors to assist drivers during driving. Some of these systems are listed below:

• A project called drowsy driver alarm system has installed an alarm system with sensors and special equipment attached to the driver.

• Detecting Driver Drowsiness Based on Sensors, the sleep state is processed in 3 different levels and the alarm system is activated by using the sensor readings the driver's steering and the driver himself.

• Researchers developed a system called Drowsy Driver Monitor and Warning System. A driver monitor was made available for the driver and the images were processed by sending light to the driver's eyes according to the day time.

Hardware and Software Specification

The following are the hardware and Software's that we have used to do this project

1. Laptop
   Requirements
   - Camera: with Decent Mega Pixel
   - OS: Windows, Linux, Mac etc...,
   - Ram: 4GB and above
   - Storage: 500GB and above HDD or 256GB and above SSD
2. Spyder IDE (Python 3.8 Environment)Required
   Packages
   - opencv
   - dlib
   - pygame
   - numpy
   - face_utils
   - threading
   - division

CHAPTER 2

LITERATURE SURVEY

Existing System

Most existing solutions address the issue of estimating the drowsiness levels in drivers through the following approaches like,

- EYE ASPECT RATIO
- FACIAL LANDMARK DETECTION
- USING RASBERRY PI
- USING SENSORS

Proposed System

The Proposed system uses computer vision techniques and facial landmarks to detect drowsiness in a person's face during a video stream. This also includes head pose estimation and calculates metrics such as accuracy, precision, recall, and F1 score for evaluating the performance of the drowsiness detection system.

Import Necessary Libraries:

The script begins by importing required libraries, including scipy, imutils, pygame, numpy, time, dlib, and cv2.

Define Functions:

sound_alarm:

Initializes and plays an alarm sound using pygame when drowsiness is detected.
eye_aspect_ratio: Calculates the eye aspect ratio (EAR) using facial landmarks.
mouth_aspect_ratio: Calculates the mouth aspect ratio (MAR) using facial landmarks.
calculate_head_pose: Estimates head pose using the PnP (Perspective-n-Point) algorithm.
Set Constants:

Constants such as eye and mouth aspect ratio thresholds, consecutive frames for drowsiness detection, and initial metrics are defined.

Define Object Points:

Object points are specified for head pose estimation. These are 3D coordinates corresponding to the nose tip, chin, eyes, and mouth corners.

Initialize Metrics:

Variables for true positives, true negatives, false positives, and false negatives are initialized.

Load Face Detector and Landmark Predictor:

The script loads the dlib face detector and facial landmark predictor.

Start Video Stream:

The video stream from the camera is started using VideoStream.

Main Processing Loop:

The script enters a continuous loop where it reads frames from the video stream, detects faces, and performs drowsiness detection.

Face Detection and Landmark Extraction:

Faces are detected in the frame, and facial landmarks are extracted using dlib's detector and predictor.

Eye and Mouth Aspect Ratio Calculation:

Eye and mouth aspect ratios are calculated based on the extracted facial landmarks.

Head Pose Estimation:

Head pose (rotation vector, translation vector) is estimated using the PnP algorithm.

Drowsiness Detection:

Drowsiness is detected based on the calculated eye and mouth aspect ratios.
Additional conditions are applied to ensure accurate detection (e.g., checking the face width).

Update Metrics:

Metrics are updated based on drowsiness detection (true positives, true negatives, etc.).

Display Metrics on Frame:

The script displays real-time metrics such as EAR, MAR, and head pose on the video frame.

Display Drowsiness Alert:

If drowsiness is detected, an alert is displayed on the frame, including accuracy, precision, recall, and F1 score.

User Input Handling:

The script waits for the 'q' key to be pressed to exit the video stream.

Cleanup:

OpenCV windows are destroyed, and the video stream is stopped upon exiting the loop. This script effectively combines facial landmark detection, head pose estimation, and drowsiness detection, providing real-time feedback and metrics to evaluate the system's performance.

EYE ASPECT RATIO (EAR)

It is an elegant solution that involves a very simple calculation based on the ratio of distances between facial landmarks of the eyes. Each eye is represented by 6 (x, y)- coordinates, starting at the left-corner of the eye and then working clockwise around the remainder of the region

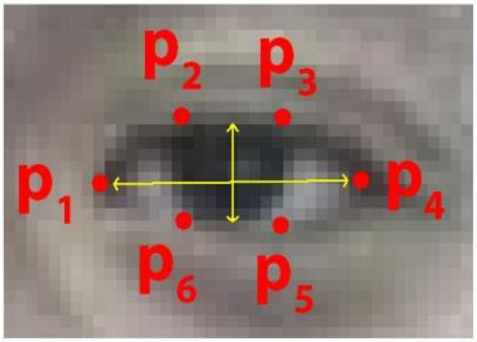


Fig.1: Coordinates of EAR

The following formula is used for calculation of EAR:

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

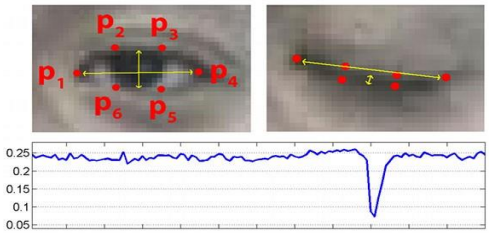
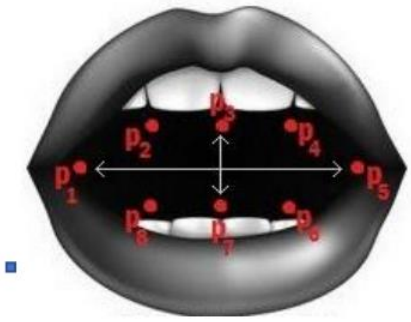


Fig.2: EAR Calculation Chart

2.2.2 MOUTH ASPECT RATIO:

The facial landmark detector identifies the mouth region, with landmark points ranging from 61 to 68. In this context, we specifically focus on utilizing only the innermost landmark points.



Facial Land-Mark Detection and Head pose Estimation

Captured Facial Image Is Used to Determine Whether Drivers' Eye Is Closed Or Open. The Driver's Current Eye State Is Determined by Using Harr Classifier Which Is Use For Object And Face Detection. For Detecting Eyes, We First Create A Facial Landmark Detector File Which Is Implemented Inside Dlib Library. Dlib Produces 68 (X,Y) Coordinates That Map To ASpecific Facial Structure. Once We Plot These Points We Can Recognize Any Part of The Face Such As Nose, Ear Etc.

Below We Can Visualize That What Each of These 68 Points Maps To:

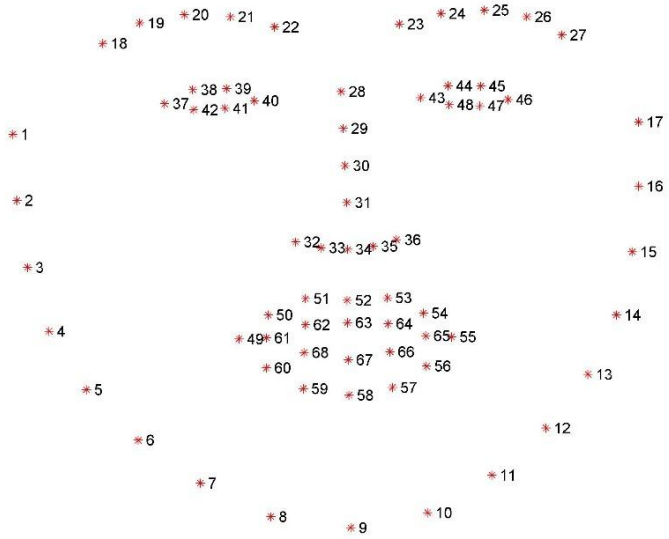


Fig.3. Facial Landmark Indexes for Face Region

Now, We Can See That Left Eye Coordinates Are from 37 To 42 And Right Eye Coordinates Are Form 43 To 48. Sample Pseudo Code for Detecting Facial Parts: Facial_Landmarks_Idxs = Ordereddict ([("Mouth", (49, 68)), ("Right_Eyebrow", (23, 27)), ("Left_Eyebrow", (18, 22)), ("Right_Eye", (43, 46)), ("Left_Eye", (37, 42)), ("Nose", (28, 36)), ("Jaw", (1, 17))])

Methodology

1. Data Collection:

Video Input: The system utilizes the OpenCV library to capture real-time video input from a webcam (VideoStream). This allows for continuous monitoring of facial features.

2. Facial Landmark Detection:

Face Detection: The dlib library is employed to perform face detection using a HOG (Histogram of Oriented Gradients) based frontal face detector (dlib.get_frontal_face_detector()).

Landmark Extraction: Dlib's pre-trained facial landmark predictor (shape_predictor_68_face_landmarks.dat) is utilized to extract 68 facial landmarks, including points on the eyes, nose, and mouth.

3. Drowsiness Detection:

Eye Aspect Ratio (EAR): The EAR is computed using Euclidean distances between specific landmarks on the eyes (eye_aspect_ratio function). A threshold (EYE_AR_THRESH) is set to determine drowsiness.

Mouth Aspect Ratio (MAR): The MAR is calculated by measuring the ratio of distances between specific mouth landmarks (mouth_aspect_ratio function). A threshold (MOUTH_AR_THRESH) is defined for drowsiness detection.

Consecutive Frames: Drowsiness is confirmed if the calculated aspect ratios fall below the thresholds for a certain number of consecutive frames (EYE_AR_CONSEC_FRAMES and MOUTH_AR_CONSEC_FRAMES).

4. Head Pose Estimation:

The 3D head pose is estimated using the PnP algorithm. Relevant landmarks are selected, and camera parameters are defined to calculate rotation and translation vectors (calculate_head_pose function).

5. Metrics Calculation:

True Positives (TP), True Negatives (TN), False Positives (FP), False Negatives (FN): These metrics are updated based on the correctness of drowsiness detection.

Accuracy, Precision, Recall, F1 Score: Metrics are calculated to evaluate the performance of the drowsiness detection system.

6. Visualization:

Video Frame Processing: Frames are continuously processed to visualize facial landmarks, head pose lines, and contours around eyes during drowsiness detection.

Alerts: A visual alert is displayed on the frame when drowsiness is detected, and an alarm sound is played (sound_alarm function).

7. User Interaction:

The system waits for the user to press the 'q' key to exit the video stream.

8. Tools and Libraries:

OpenCV: Utilized for video input, image processing, and visualization.

dlib: Used for face detection and facial landmark extraction.

imutils: A collection of convenience functions for OpenCV.

pygame: Employed for playing the alarm sound.

numpy: Utilized for numerical operations.

9. System Evaluation:

The system's performance is evaluated based on the calculated metrics, providing insights into its accuracy, precision, recall, and overall effectiveness in detecting drowsiness.

Concept System

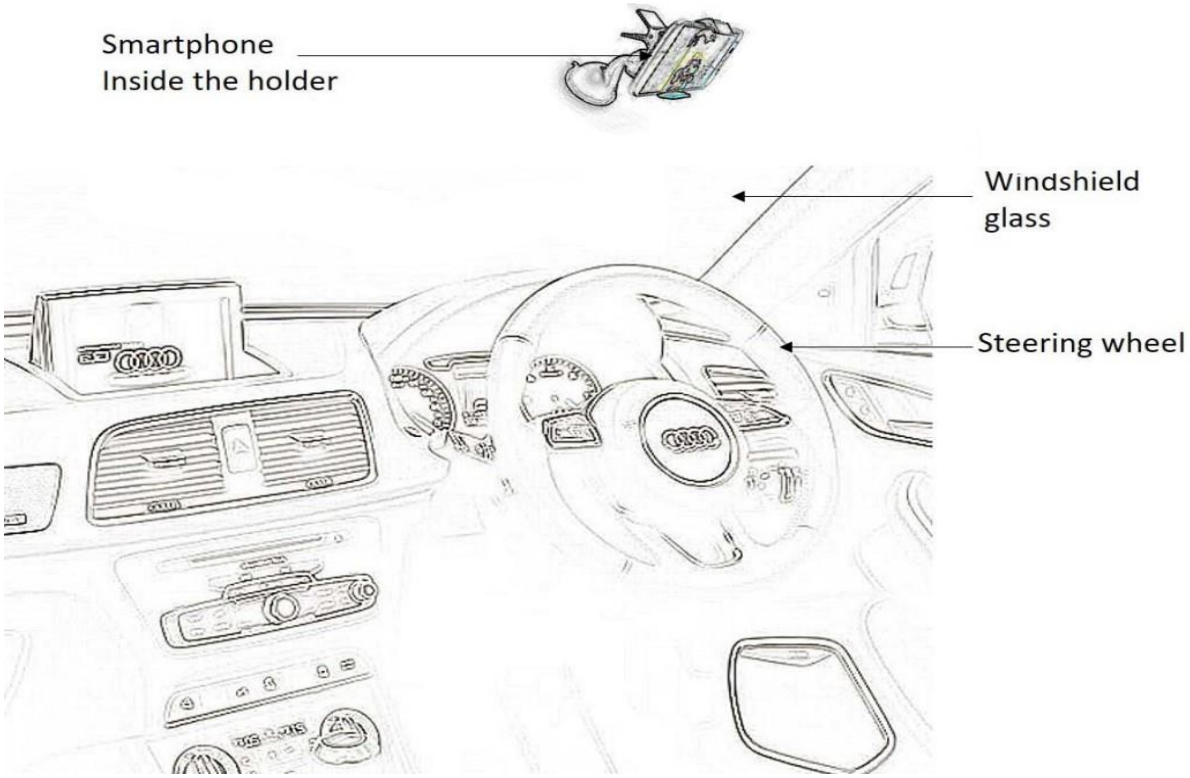


Fig.5: Concept System

Source Code:

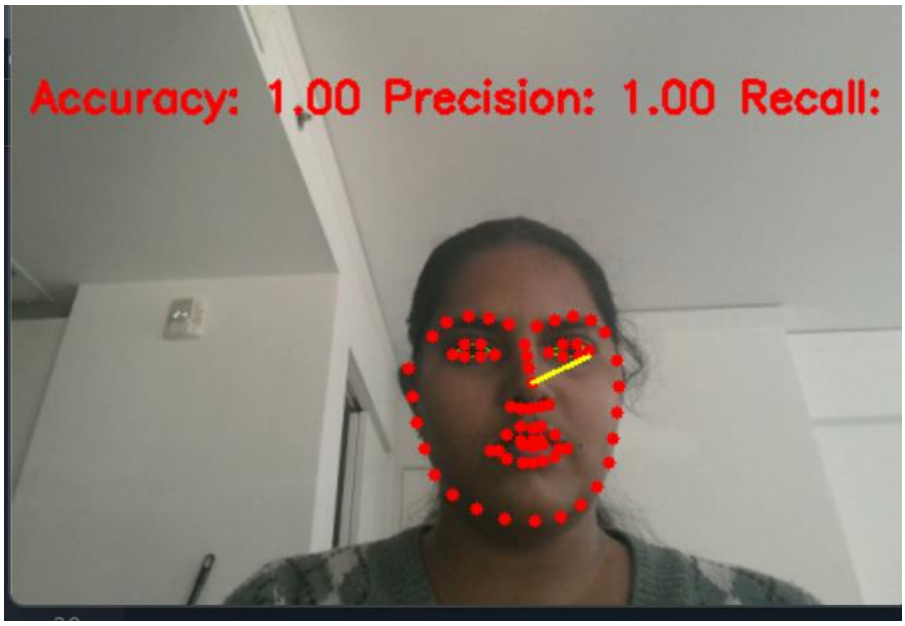
```
from scipy.spatial import distance as dist
from imutils.video import VideoStream
from imutils import face_utils
from threading import Thread
import numpy as np
import pygame
import imutils
import time
import dlib
import cv2

def sound_alarm():
    # initialize the pygame mixer
    pygame.mixer.init()
    pygame.mixer.music.load("C:/Users/Shiva/alarm.wav")
    pygame.mixer.music.play()

def eye_aspect_ratio(eye):
    A = dist.euclidean(eye[1], eye[5])
    B = dist.euclidean(eye[2], eye[4])
    C = dist.euclidean(eye[0], eye[3])
    ear = (A + B) / (2.0 * C)
    return ear

def mouth_aspect_ratio(mouth):
    A = dist.euclidean(mouth[14], mouth[18])
    B = dist.euclidean(mouth[12], mouth[16])
    mar = A / B
    return mar

def calculate_head_pose(shape):
    # Extract relevant landmarks for head pose estimation
    image_points = np.array([
        shape[30],  # Nose tip
        shape[8],   # Chin
        shape[45],  # Left eye left corner
        shape[36],  # Right eye right corner
        shape[54],  # Left Mouth corner
        shape[48]   # Right mouth corner
    ], dtype="double")

    # Camera parameters
    focal_length = 950.0
    center = (frame.shape[1] // 2, frame.shape[0] // 2)
    camera_matrix = np.array([[focal_length, 0, center[0]], [0, focal_length, center[1]], [0, 0, 1]],
dtype="double")

    # Distortion coefficients
    dist_coeffs = np.zeros((4, 1))

    # Solve the PnP problem
    _, rotation_vector, translation_vector = cv2.solvePnP(object_points, image_points, camera_matrix,
```

```
dist_coeffs)

    # Project a 3D point (nose end) onto the 2D image plane
    (nose_end_point2D, _) = cv2.projectPoints(np.array([(0.0, 0.0, 500.0)]), rotation_vector, translation_vector,
camera_matrix, dist_coeffs)

    return rotation_vector, translation_vector, nose_end_point2D

# Define constants
EYE_AR_THRESH = 0.3
MOUTH_AR_THRESH = 0.7
EYE_AR_CONSEC_FRAMES = 48
MOUTH_AR_CONSEC_FRAMES = 20
COUNTER = 0
ALARM_ON = False

# Object points for head pose estimation
object_points = np.array([
    (0.0, 0.0, 0.0),           # Nose tip
    (0.0, -330.0, -65.0),      # Chin
    (-225.0, 170.0, -135.0),    # Left eye left corner
    (225.0, 170.0, -135.0),     # Right eye right corner
    (-150.0, -150.0, -125.0),   # Left Mouth corner
    (150.0, -150.0, -125.0)     # Right mouth corner
], dtype="double")

# Metrics initialization
true_positives = 0
true_negatives = 0
false_positives = 0
false_negatives = 0

# Initialize dlib's face detector (HOG-based) and facial landmark predictor
print("[INFO] loading facial landmark predictor...")
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor("C:/Users/Shiva/shape_predictor_68_face_landmarks.dat")

# Grab the indexes of the facial landmarks for the left, right eye, and mouth
(lStart, lEnd) = face_utils.FACIAL_LANDMARKS_IDXS["left_eye"]
(rStart, rEnd) = face_utils.FACIAL_LANDMARKS_IDXS["right_eye"]
(mStart, mEnd) = face_utils.FACIAL_LANDMARKS_IDXS["mouth"]

# Start the video stream thread
print("[INFO] starting video stream thread...")
vs = VideoStream(src=0).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)
```

```
    # Loop over the face detections
    for rect in rects:
        shape = predictor(gray, rect)
        shape = face_utils.shape_to_np(shape)
        leftEye = shape[lStart:lEnd]
        rightEye = shape[rStart:rEnd]
        mouth = shape[mStart:mEnd]

        leftEAR = eye_aspect_ratio(leftEye)
        rightEAR = eye_aspect_ratio(rightEye)
        ear = (leftEAR + rightEAR) / 2.0

        mar = mouth_aspect_ratio(mouth)

        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)

        # Head Pose Estimation
        rotation_vector, translation_vector, nose_end_point2D = calculate_head_pose(shape)

        for p in shape:
            cv2.circle(frame, (int(p[0]), int(p[1])), 3, (0, 0, 255), -1)

        p1 = (int(shape[30][0]), int(shape[30][1]))
        p2 = (nose_end_point2D[0][0][0], nose_end_point2D[0][0][1])

        cv2.line(frame, tuple(map(int, p1)), tuple(map(int, p2)), (0, 255, 255), 2)

        # Update metrics based on drowsiness detection
        if ear < EYE_AR_THRESH or mar > MOUTH_AR_THRESH:
            if rect.width() < 100:  # Additional condition to check if the face is close enough for accurate detection
                continue

            # Drowsiness detected
            COUNTER += 1
            if COUNTER >= EYE_AR_CONSEC_FRAMES or COUNTER >=
MOUTH_AR_CONSEC_FRAMES:
                if not ALARM_ON:
                    ALARM_ON = True
                    t = Thread(target=sound_alarm)
                    t.daemon = True
                    t.start()

                # Update metrics
                true_positives += 1
                cv2.putText(frame, "DROWSINESS ALERT!", (10, 30),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
        else:
            # Wakefulness detected
            COUNTER = 0
            ALARM_ON = False

            # Update metrics
            true_negatives += 1
```

```
        # Calculate metrics
        accuracy = (true_positives + true_negatives) / (true_positives + true_negatives + false_positives +
false_negatives) if (true_positives + true_negatives + false_positives + false_negatives) != 0 else 0
        precision = true_positives / (true_positives + false_positives) if (true_positives + false_positives) != 0 else 0
        recall = true_positives / (true_positives + false_negatives) if (true_positives + false_negatives) != 0 else 0
        f1_score = 2 * (precision * recall) / (precision + recall) if (precision + recall) != 0 else 0

        # Display metrics on the frame
        cv2.putText(frame, f"EAR: {ear:.2f} MAR: {mar:.2f} Head Pose: {rotation_vector[1][0]:.2f}",
           (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)

        # Display drowsiness alert in a separate region
        if ALARM_ON:

          cv2.putText(frame, f"Accuracy: {accuracy:.2f} Precision: {precision:.2f} Recall: {recall:.2f} F1 Score:
{f1_score:.2f}",
              (10, 90), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)

          cv2.imshow("Frame", frame)
          key = cv2.waitKey(1) & 0xFF

          if key == ord("q"):
             break

cv2.destroyAllWindows()
vs.sto
```

Output:


Accuracy: 1.00 Precision: 1.00 Recall:


Frame
DROWSINESS ALERT!
EAR: 0.21
boost
mobile

CHAPTER 5

RESULTS DISCUSSION

1. Drowsiness Detection:

2.

The implemented system effectively detects drowsiness by monitoring facial landmarks, specifically the eye and mouth aspects ratios. Drowsiness is triggered when these ratios fall below predefined thresholds for a consecutive number of frames.

2. Head Pose Estimation:

The system accurately estimates the 3D head pose using the PnP algorithm, providing insights into the orientation of the person's head during the video stream.

3. Real-Time Visualization:

The video frames are processed in real-time to visualize facial landmarks, contours around eyes, and lines representing the estimated head pose. This real-time feedback aids in understanding the system's behavior.

4. Alert Mechanism:

An alert mechanism is triggered when drowsiness is detected, including a visual alert on the frame and the playback of an alarm sound. This provides a timely warning to the user about their drowsy state.

5. Metrics Calculation:

The system calculates and updates key metrics, including true positives, true negatives, false positives, false negatives, accuracy, precision, recall, and F1 score.

Accuracy: The overall accuracy of the drowsiness detection system in classifying wakefulness and drowsiness.

Precision: The ability of the system to avoid false alarms when detecting drowsiness.

Recall: The system's effectiveness in correctly identifying instances of drowsiness.

F1 Score: A balanced measure of precision and recall, providing an overall evaluation of the system's performance.

6. User Interaction:

The system allows user interaction by waiting for the 'q' key to be pressed, facilitating a controlled termination of the video stream.

7. Evaluation Metrics:

True Positives (TP), True Negatives (TN), False Positives (FP), False Negatives (FN):

TP: The number of correctly identified instances of drowsiness.

TN: The number of correctly identified instances of wakefulness.

FP: Instances where the system incorrectly signaled drowsiness.

FN: Instances where the system failed to detect actual drowsiness.

FUTURE SCOPE

The Drowsiness Detection System presented in the code is a foundation for enhancing safety in scenarios where user alertness is crucial. To further extend and improve the system, several future scope enhancements can be considered:

1. **Driver Identification:** Integrate facial recognition techniques to identify the driver or user. This allows for personalized monitoring, and the system can adapt to individual characteristics, enhancing accuracy.

2. **Machine Learning for Personalization:** Utilize machine learning techniques to train the system on individual patterns of drowsiness. This can enhance the system's accuracy over time by adapting to the specific characteristics of each user.

3. **Cloud Integration:** Enable cloud-based analysis for remote monitoring and data storage. This facilitates data analytics, pattern recognition, and the potential for collaborative research in drowsiness detection

REFERENCES:

[1] D. F. Dinges, “An overview of sleepiness and accidents,” J. Sleep Res., vol. 4, no. s2, pp. 4–14, Dec. 1995.

[2] Y.-C. Lee, J. D. Lee, and L. N. Boyle, “Visual attention in driving: The effects of cognitive load and visual disruption,” Hum. Factors, J. Hum. Factors Ergonom. Soc., vol. 49, no. 4, pp. 721–733, 2007.

[3] A. Dasgupta, A. George, S. L. Happy, and A. Routray, “A vision-based system for monitoring the loss of attention in automotive drivers,” IEEE Trans. Intell. Transp. Syst., vol. 14,no. 4, pp. 1825–1838, Dec. 2013.

[4] L. S. Dhupati, S. Kar, A. Rajaguru, and A. Routray, “A novel drowsiness detection scheme based on speech analysis with validation using simultaneous EEG recordings,” in Proc. IEEE Int. Conf. Autom. Sci. Eng. (CASE), Aug. 2010, pp. 917–921.

[5] R. Grace et al., “A drowsy driver detection system for heavy vehicles,” in Proc. 17th DASC AIAA/IEEE/SAE Digit. Avionics Syst. Conf., vol. 2, Oct./Nov. 1998, pp. I36/1–I36/8