B.Tech. IoT Programming
Project Report

On

Real-Time Driver Fatigue and Distraction Monitoring System

By

Group Leader:

Raghav Dolyar (124102019) 124102019@nitkkr.ac.in 7988226170

Other Team Members:

Pranav Patidar (124102035) 124102035@nitkkr.ac.in 86430 78304
Yogishwer Kumar (124102025) 124102025@nitkkr.ac.in 9103854571
Harkesh (124102028) 124102028@nitkkr.ac.in 9518437456
Suleman Khan (124102031) 124102031@nitkkr.ac.in 8467066883

Branch: CS – A – 02
Subject: IoT Programming (CSPC 209)

Abstract

With the increasing number of vehicles on roads worldwide, driver fatigue and distraction have become leading contributors to traffic accidents. A brief lapse of attention can lead to severe consequences, yet many vehicles lack the required safety tools to monitor and alert a driver who is physically or cognitively impaired. This study presents the design and development of a real-time Driver Drowsiness and Distraction Detection System aimed at enhancing road safety through continuous driver monitoring and an automated, escalating alert system.

The proposed system is built around a Raspberry Pi 4 microcontroller, which processes a live video feed from a Raspberry Pi 5MP Camera Module. It leverages computer vision libraries, primarily OpenCV for image processing and Dlib for machine learning-based facial analysis. A key feature of this system is its ability to detect drowsiness by calculating the Eye Aspect Ratio (EAR) to identify prolonged eye closure and the Mouth Aspect Ratio (MAR) to detect yawning. These metrics are derived from a pre-trained model that predicts 68 facial landmarks in real-time. Distraction is detected when the driver's face is not visible in the camera's frame for a set duration.

Overall, the proposed Driver Drowsiness and Distraction Detection System enhances driver safety through real-time monitoring, intelligent computer vision analysis, and IoT-connected alert escalation.

The accuracy of our system is nearly 90%. The formula used for calculating accuracy is: -
Accuracy = (Total Times it Correctly Alerted) / (Total Attempts)

Keywords: Internet of Things (IoT), Real-Time Monitoring, Driver Drowsiness Detection, Computer Vision, Raspberry Pi, Eye Aspect Ratio (EAR), Dlib, Facial Landmarks, Twilio.

1. Introduction

Imagine you've had a long, exhausting day at work, and now you're driving home on a busy Indian highway in the evening. Your body feels heavy with tiredness and a big yawn slips out. Suddenly, the road ahead turns into a scary risk. For thousands of drivers in our country, this quick moment of sleepiness can lead to a horrible accident that can ruin lives. But what if a simple smart system in your car could wake you up instantly, turning a possible disaster into a safe trip home?

Safe driving needs sharp focus and quick reactions, but drowsiness takes both away, making even normal trips dangerous. Long hours on the road, especially on India's long national highways or during night shifts for truck drivers, make the risk even worse. This leads to veering off the road, slow stops, and terrible crashes. Our project, "IoT-Based Drowsiness and Yawn Detection System," acts as a low-cost, real-time guard.

According to the latest 2022 data from the Ministry of Road Transport and Highways (MoRTH)—the most recent full report as of 2025—India had a shocking 4,61,312 road accidents. These killed 1,68,491 people and injured 4,43,366 others. Drowsiness and driver tiredness contributes 10-20% of these crashes, mainly on the fast highways. That means about 16,800-33,700 deaths and thousands more injuries each year [1]. The World Health Organization (WHO) says the same worldwide: fatigue plays a role in up to 20% of road deaths. This shows the big need for active safety tools in India, where road accidents are the ninth top cause of death.

In the last 10 years, tech has changed how we drive, with the Internet of Things (IoT) adding smart features to vehicles for things like maps and music. High-end systems like Advanced Driver Assistance Systems (ADAS) in cars such as Tesla's Autopilot use cameras and AI to spot tiredness, but they cost too much for most people in India. Simpler ideas, like phone apps for directions or wearables that check fatigue, are a good start. But they don't give the quick, car-built response needed in real dangers.

Our project takes affordable car tech to the next level. It combines a Raspberry Pi 4 with a basic camera to spot eye blinks and yawns which are the clear signs of tired driving. It also uses IoT for instant alerts. This setup warns the driver with buzzers, phone messages and calls. It helps everyday drivers, long-haul truckers, and fleet owners in India's busy and varied roads.

[1] Ministry of Road Transport and Highways (MoRTH), "Road Accidents in India 2022."

Unique Safety Features: -

1. **Driver Authentication (Face Match):** The system first uses face recognition to confirm the identity of the person in the driver's seat. This ensures that the safety features and alerts are personalized to the correct registered driver.

2. **Immediate Mobile Alert:** The very first warning includes a loud, high-priority notification sent directly to the driver's phone, which works alongside the in-car light and buzzer to maximize the chance of a wake-up response.

3. **Automatic Emergency Call & Location Share:** If the driver doesn't responds for 20 seconds, the system initiates a hands-free call to the registered emergency contact, simultaneously sending a text with a Google Maps location link for immediate dispatch of help.

2. Related Work in Project

Title	Authors	Key Feature/ Methods	Year and Link	Drawbacks/ Limitations
Real-Time Driver-Drowsiness Detection System Using Facial Features	Ruoxue Wu, Wanghua Deng	• Convolutional neural network • Fatigue detection • Feature location • Face tracking	2019 [1]	1)Relies only on camera-based facial cues, so it struggles when drivers wear sunglasses, turn their heads, or drive in very low light. 2)The "real-time" results were shown on standard hardware — the paper doesn't profile latency or power on typical in-car or low-power units.
Driver Drowsiness Detection by Applying Deep Learning Techniques to Sequences of Images	Elena Magan , M. Paz Sesmaro , Juan Manuel Alonso-Weber, Araceli Sanchis	. Eye state (open/closed) . Yawning behavior . Blink duration/frequency . Head movement / position . Temporal drowsiness pattern	2022 [2]	1) Dependent on Video Quality 2) Only ~65% on training data and ~60% on test data.

Driver Drowsiness Prediction Based on Multiple Aspects Using Image Processing Techniques	V.Uma Maheswa ri, Rajanikanth Aluvalu	. Eye Aspect Ratio (EAR) for eye status detection. . Mouth Aspect Ratio (MAR) for yawning detection. . Face Aspect Ratio (FAR) for hand-occluded or micro-gestures	2022 [3]	Needs high processing power to run. Limited or biased data can reduce real-world accuracy. Accuracy depends on how well key details are detected.
Real-Time Driver Drowsiness Detection Using Facial Analysis and Machine Learning Techniques	Siham Essahraou, Ismail Lamaaka, Yassine Maleh, Osama Alfarraj	. Uses facial analysis (eyes, mouth, head movements) for drowsiness detection . Employs machine learning (ML) and deep learning (DL) techniques (CNN, BiLSTM, ViT, etc.) . Can utilize physiological signals (EEG, ECG, EMG, HRV, EDA) for multimodal detection . Supports real-time monitoring and proactive detection	2025 [4]	. Accuracy may vary across individuals due to diverse facial features and expressions . Some models are computationally heavy, limiting deployment on low-power devices . Video-based and temporal models need more processing power and storage

3. Project

This project details the design and implementation of an advanced, real-time Driver Assistance System (ADAS) built on a Raspberry Pi 4. The primary objective is to enhance driver safety by actively monitoring for signs of fatigue and distraction, and to execute a 2-stage alert protocol in response to detected signs of risks.

The system uses computer vision and machine learning to analyze a live video feed of the driver (in multiple frames). It is designed to be a standalone, embedded system capable of functioning within a vehicle.

Technical Details:

The system's functionality is divided into three core modules:

1. **Sensing Module:** Utilizes a Pi Camera to capture a continuous video stream of the driver.
2. **Processing Module:** Runs on the Raspberry Pi and uses a Python script with specialized libraries (OpenCV, dlib, face_recognition) to analyze each frame in real-time.
3. **Alerting Module:** Triggers a series of local and remote alerts based on the analysis.

Key Features Implemented:

- **Drowsiness Detection:** Measures the Eye Aspect Ratio (EAR) to detect prolonged eye closure.
- **Yawn Detection:** Measures the Mouth Aspect Ratio (MAR) to identify yawns as a secondary indicator of fatigue.
- **Distraction Detection:** A code logic that triggers an alert if the driver's face is not detected by the camera for a specified duration, indicating they are not looking forward.
- **Driver Authentication:** Utilizes facial recognition to distinguish between authorized drivers and "Unknown" guest drivers, tailoring the alert protocol accordingly.

- **Two-Stage Alert System:**
    - **Stage 1 (Immediate Alert):** An instant local alert (LED and speaker) combined with a high-priority notification sent to the driver's smartphone via the Pushover API.
    - **Stage 2 (Emergency Escalation):** If an alert condition persists for a pre-defined period (20 seconds), the system escalates the response. The Raspberry Pi initiates an automated voice call to a designated emergency contact via the Twilio API, while the driver's phone (using Macrodroid) sends a follow-up SMS containing a Google Maps link with the vehicle's current GPS location.

Algorithm of Project Work:

1. **Initialization:**
    - Initialize all hardware components (Camera, GPIO pins for LED/Speaker).
    - Load API credentials for Pushover and Twilio.
    - Load the dlib facial landmark prediction model.
    - Load the facial encodings of authorized drivers from the known_faces/ directory.

2. **Main Loop (for each frame):**
    - Capture a frame from the camera.
    - Convert the frame to grayscale for faster processing.
    - Attempt to detect a face in the frame.
    - **If no face is found:**
        - Increment the no_face counter.
        - If the counter exceeds NO_FACE_CONSEC_FRAMES, set the alert_condition_met flag to True.
    - **If a face is found:**
        - Reset the no_face counter.

        - Perform facial recognition to identify the driver as "Authorized" or "Unknown."
        - If the driver is "Unknown," send a one-time security notification to the owner.
        - Detect the 68 facial landmarks on the face.
        - Calculate the Eye Aspect Ratio (EAR) and Mouth Aspect Ratio (MAR).
        - Check EAR against EYE_AR_THRESH. If below, increment the eye_flag counter.
        - Check MAR against YAWN_MAR_THRESH. If above, increment the yawn_flag counter.
        - If either eye_flag or yawn_flag exceeds its respective frame threshold, set the alert_condition_met flag to True.

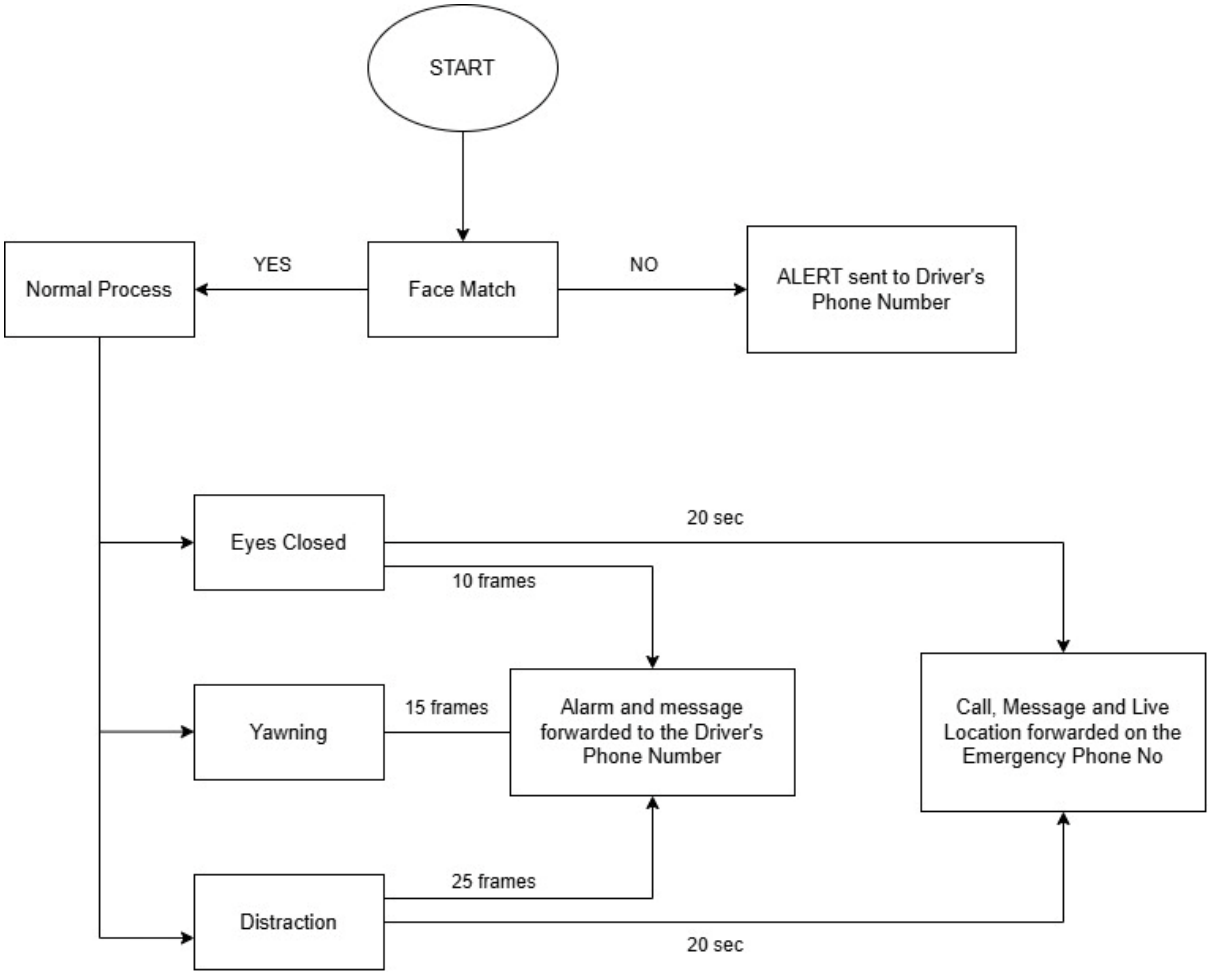
3. **Alert Logic (checked every loop):**
    - **If alert_condition_met is True:**
        - Turn on the local LED and Speaker.
        - If this is the start of a new event, start a 20-second timer and send the initial Pushover alarm to the driver's phone.
        - If the timer exceeds 20 seconds, trigger the emergency voice call from the Pi. (The phone's Macrodroid macro simultaneously sends the location SMS).
    - **If alert_condition_met is False:**
        - Turn off the local LED and Speaker.
        - Reset all timers and flags.

4. **Display:**
    - Show the processed video frame with all overlays (facial landmarks, alert text) in a window.
    - Listen for a 'q' key press to exit.

5. **Cleanup:**
    - When the program exits, release all resources (camera, GPIO pins).

3.1 Conceptual Design Diagram

The architecture of the system follows a logical flow from data acquisition to action, involving both the Raspberry Pi and the driver's smartphone in a coordinated manner.



Used Code: -

```
# ------------------ Import Required Libraries ------------------
import cv2
from scipy.spatial import distance
import RPi.GPIO as GPIO
from imutils import face_utils
import numpy as np
import time
import dlib
from picamera2 import Picamera2
from pushover import Client as PushoverClient
from twilio.rest import Client as TwilioClient
import face_recognition
import os

# ------------------ Cloud Service Credentials ------------------
PUSHOVER_USER_KEY = "uq6oortiezaz5rosym6g7tt19z7hq7"
PUSHOVER_API_TOKEN = "a85fe75dsawtnncqwursg289y5z25f"

TWILIO_ACCOUNT_SID = "ACa6ac7e61dc6be192fc7b0a77b91207d1"
TWILIO_AUTH_TOKEN = "42ccfcd29e0b1a57a25016b1ce028cdc"
TWILIO_PHONE_NUMBER = "+18145272077"
EMERGENCY_CONTACT_NUMBER = "+918643078304"

# ------------------ Initialize Cloud Clients ------------------
pushover_client = PushoverClient(PUSHOVER_USER_KEY, api_token=PUSHOVER_API_TOKEN)
twilio_client = TwilioClient(TWILIO_ACCOUNT_SID, TWILIO_AUTH_TOKEN)



# ------------------ Cloud Alert Functions ------------------
def send_phone_alarm(message, title="Driver Alert!", priority=1):
    """Send push notification to mobile using Pushover app."""
    print(f"Sending Pushover alert: {title}")
    try:
        pushover_client.send_message(message, title=title, priority=priority, sound='siren')
    except Exception as e:
        print(f"Failed to send Pushover alert: {e}")
```

```
def make_emergency_call():
    """Make emergency phone call using Twilio when driver doesn’t respond."""
    print("Making EMERGENCY CALL via Twilio...")
    try:
        twiml_url = "https://handler.twilio.com/twiml/EH97f61d52ddbe0a01a8d11ad4428b8f72"
        call = twilio_client.calls.create(
            url=twiml_url,
            to=EMERGENCY_CONTACT_NUMBER,
            from_=TWILIO_PHONE_NUMBER
        )
        print(f"Emergency call initiated with SID: {call.sid}")
    except Exception as e:
        print(f"Failed to make Twilio call: {e}")



# ------------------ GPIO Setup for Local Alerts ------------------
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
SPEAKER_PIN = 21
LED_PIN = 17
GPIO.setup(SPEAKER_PIN, GPIO.OUT)
GPIO.setup(LED_PIN, GPIO.OUT)
pwm = GPIO.PWM(SPEAKER_PIN, 440)



# ------------------ Helper Functions ------------------
def eye_aspect_ratio(eye):
    A = distance.euclidean(eye[1], eye[5])
    B = distance.euclidean(eye[2], eye[4])
    C = distance.euclidean(eye[0], eye[3])
    return (A + B) / (2.0 * C)



def mouth_aspect_ratio(mouth):
    A = distance.euclidean(mouth[2], mouth[10])
    B = distance.euclidean(mouth[4], mouth[8])
    C = distance.euclidean(mouth[0], mouth[6])
    return (A + B) / (2.0 * C)
```

```
# ------------------ Load Known Driver Faces ------------------
print("Loading known faces...")
known_face_encodings = []
known_face_names = []

for filename in os.listdir("known_faces"):
    if filename.endswith((".jpg", ".png", ".jpeg")):
        try:
            image = face_recognition.load_image_file(f"known_faces/{filename}")
            encoding = face_recognition.face_encodings(image)[0]
            known_face_encodings.append(encoding)
            known_face_names.append(os.path.splitext(filename)[0])
        except IndexError:
            print(f"WARNING: No face found in {filename}. Skipping.")

print(f"Loaded {len(known_face_names)} authorized drivers: {known_face_names}")



# ------------------ Thresholds ------------------
EYE_AR_THRESH = 0.25
EYE_AR_CONSEC_FRAMES = 10
YAWN_MAR_THRESH = 0.5
YAWN_CONSEC_FRAMES = 15
NO_FACE_CONSEC_FRAMES = 25



# ------------------ State Variables ------------------
eye_flag, yawn_flag, no_face_flag = 0, 0, 0
drowsiness_start_time = None
call_made_for_event = False



# ------------------ Initialize Dlib & Camera ------------------
detect = dlib.get_frontal_face_detector()
predict = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")
(lStart, lEnd) = face_utils.FACIAL_LANDMARKS_68_IDXS["left_eye"]
(rStart, rEnd) = face_utils.FACIAL_LANDMARKS_68_IDXS["right_eye"]
(mStart, mEnd) = face_utils.FACIAL_LANDMARKS_68_IDXS["mouth"]
```

```
current_driver = "Authenticating..."
guest_alert_sent = False

picam2 = Picamera2()
picam2.configure(picam2.create_preview_configuration(main={"format": 'XRGB8888', "size": (640, 480)}))
picam2.start()
print("Camera started. Press 'q' to quit.")

cv2.namedWindow("Frame", cv2.WINDOW_NORMAL)



# ------------------ Main Loop ------------------
while True:
    frame = picam2.capture_array()
    frame = cv2.cvtColor(frame, cv2.COLOR_BGRA2BGR)
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    subjects = detect(gray, 0)
    alert_condition_met = False
    face_found = len(subjects) > 0

    # ---- Case 1: No face detected (Driver distracted) ----
    if not face_found:
        no_face_flag += 1
        if no_face_flag >= NO_FACE_CONSEC_FRAMES:
            alert_condition_met = True
            cv2.putText(frame, "DISTRACTION ALERT!", (10, 30),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)

    else:
        no_face_flag = 0
        subject = subjects[0]

        # ---- Driver Authentication ----
        face_locations = [(subject.top(), subject.right(), subject.bottom(), subject.left())]
        rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        face_encodings = face_recognition.face_encodings(rgb_frame, face_locations)

        if len(face_encodings) > 0:
```

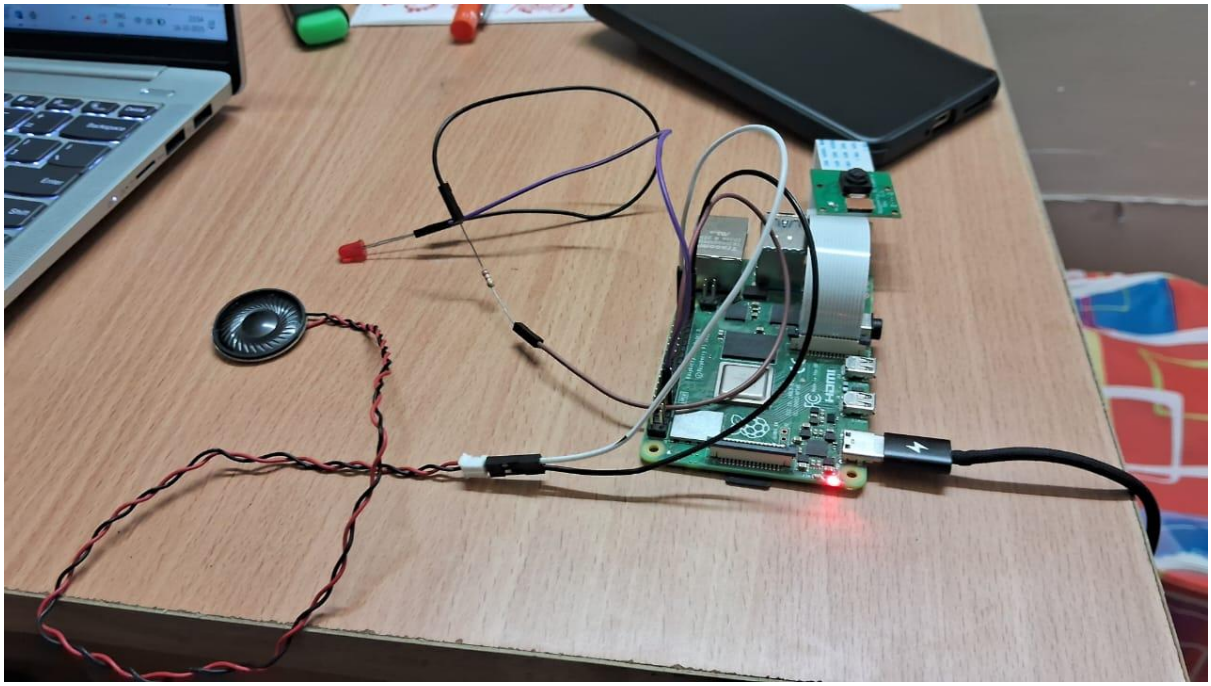
```
    matches = face_recognition.compare_faces(known_face_encodings, face_encodings[0])
    name = "Unknown"
    if True in matches:
        name = known_face_names[matches.index(True)]

    if name != current_driver:
        current_driver = name
        print(f"Driver identified as: {current_driver}")
        guest_alert_sent = False

cv2.putText(frame, f"Driver: {current_driver}", (10, 30),
            cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)

# ---- Alert if Unauthorized Driver ----
if current_driver == "Unknown" and not guest_alert_sent:
    send_phone_alarm("An unauthorized driver is operating the vehicle.",
                     title="Security Alert", priority=0)
    guest_alert_sent = True

# ---- Detect Drowsiness ----
shape = predict(gray, subject)
shape = face_utils.shape_to_np(shape)

leftEye = shape[lStart:lEnd]
rightEye = shape[rStart:rEnd]
ear = (eye_aspect_ratio(leftEye) + eye_aspect_ratio(rightEye)) / 2.0

mouth = shape[mStart:mEnd]
mar = mouth_aspect_ratio(mouth)

leftEyeHull = cv2.convexHull(leftEye)
rightEyeHull = cv2.convexHull(rightEye)
mouthHull = cv2.convexHull(mouth)

cv2.drawContours(frame, [leftEyeHull], -1, (0, 255, 0), 1)
cv2.drawContours(frame, [rightEyeHull], -1, (0, 255, 0), 1)
cv2.drawContours(frame, [mouthHull], -1, (0, 255, 0), 1)

# ---- Eye Closure ----
```

```
    if ear < EYE_AR_THRESH:
        eye_flag += 1
        if eye_flag >= EYE_AR_CONSEC_FRAMES:
            alert_condition_met = True
            cv2.putText(frame, "DROWSINESS ALERT!", (10, 60),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
    else:
        eye_flag = 0

    # ---- Yawning ----
    if mar > YAWN_MAR_THRESH:
        yawn_flag += 1
        if yawn_flag >= YAWN_CONSEC_FRAMES:
            alert_condition_met = True
            cv2.putText(frame, "YAWN ALERT!", (10, 90),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
    else:
        yawn_flag = 0

# ------------------ Handle Alerts ------------------
if alert_condition_met:
    GPIO.output(LED_PIN, GPIO.HIGH)
    pwm.start(50)

    if drowsiness_start_time is None:
        print("Alert event started. Sending phone alarm.")
        drowsiness_start_time = time.time()
        send_phone_alarm("DROWSINESS ALERT! Please respond.")  # triggers cloud alert

    elapsed_time = time.time() - drowsiness_start_time
    if elapsed_time > 20 and not call_made_for_event:
        make_emergency_call()
        call_made_for_event = True

else:
    GPIO.output(LED_PIN, GPIO.LOW)
    pwm.stop()
    if drowsiness_start_time is not None:
        print("Alert event ended.")
```

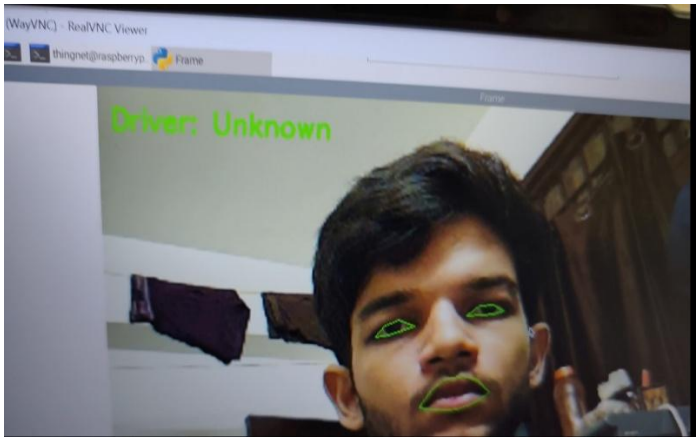
```
        drowsiness_start_time = None
        call_made_for_event = False

    # ------------------ Display ------------------
    cv2.imshow("Frame", frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cv2.destroyAllWindows()
picam2.stop()
pwm.stop()
GPIO.cleanup()
```

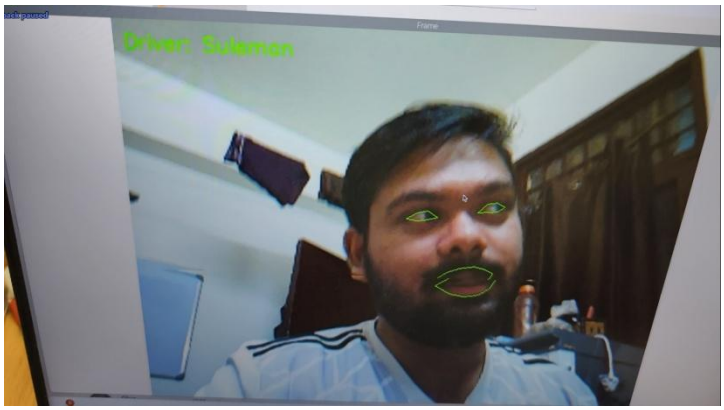
3.2 Working Example

1. The Complete Hardware Setup

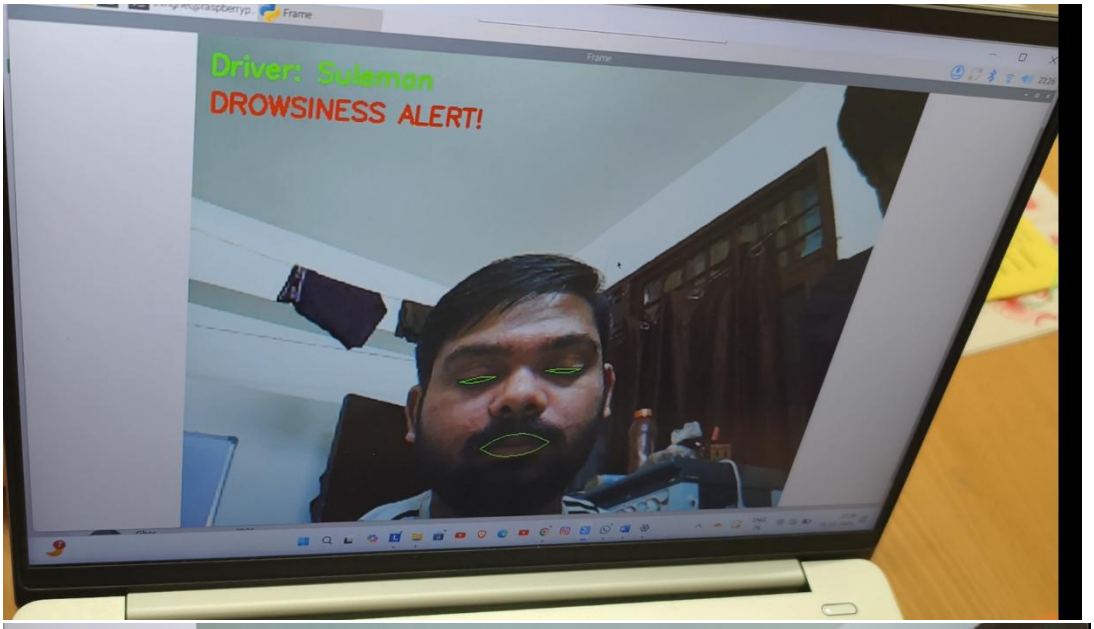


2. Driver Authentication in Progress

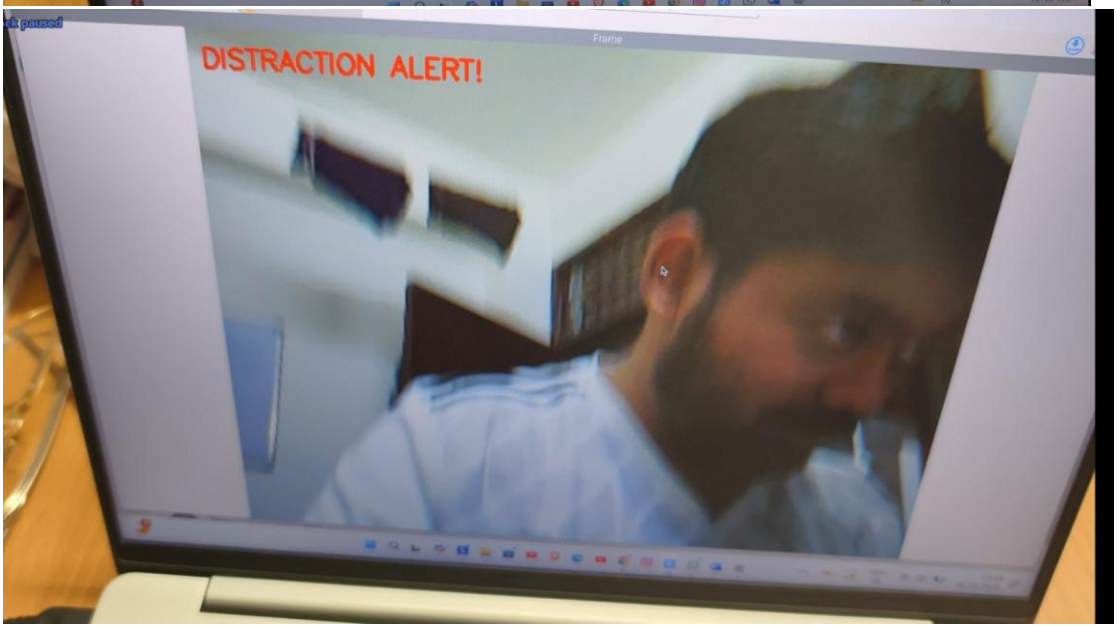


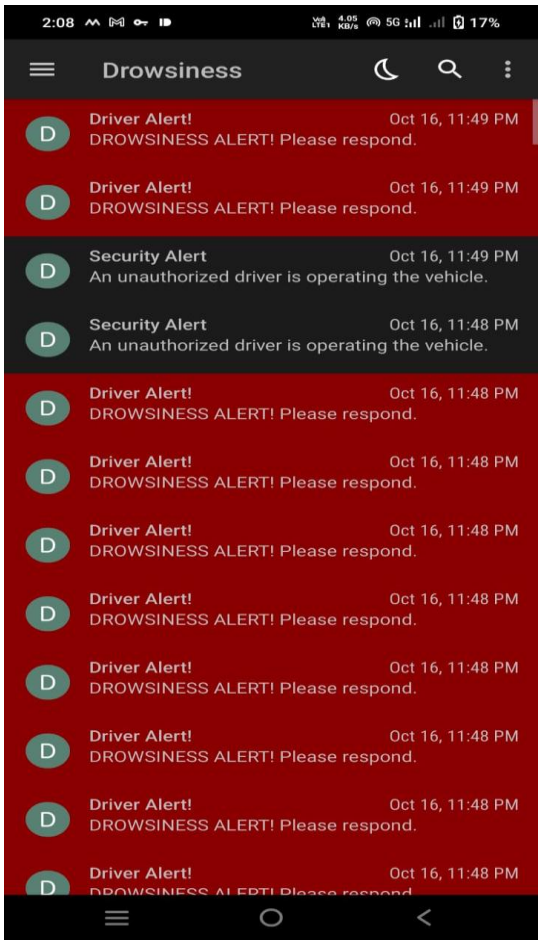


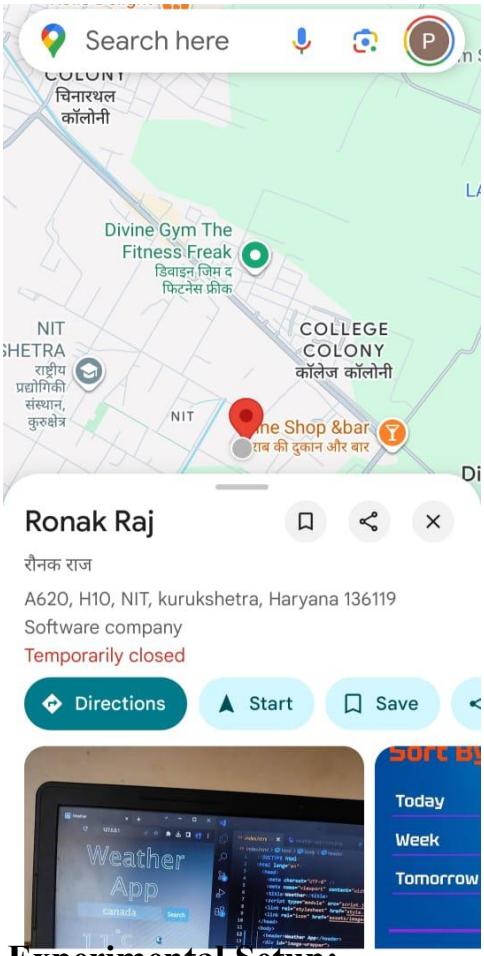
3.Drowsiness Alert Triggered

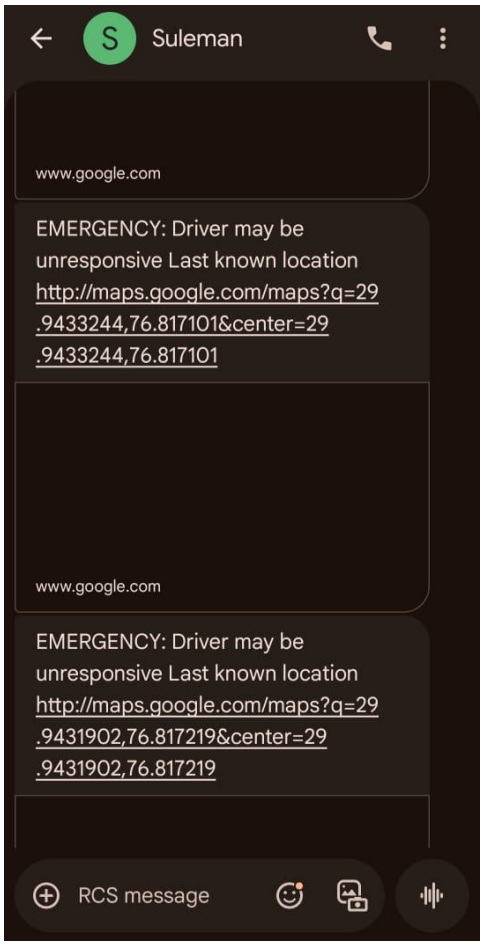

Driver: Suleman
DROWSINESS ALERT!


Driver: Suleman
DROWSINESS ALERT!


DISTRACTION ALERT!







4. Experimental Setup: -

4.1 Implementation Details

The project was implemented using a combination of specific hardware and software components, each chosen for its suitability for a real-time embedded computer vision task.

Table 1: Implementation Details

Component Category	Item	Role/Purpose
Hardware	Raspberry Pi 4 (4GB RAM)	Main processing unit for all computer vision algorithms and system logic.
	Raspberry Pi Camera Module	5MP camera used for capturing the 640x480 video stream of the driver.
	5mm Red LED	Provides an immediate visual alert to the driver.
	Passive Speaker (16Ω 0.25W)	Provides an immediate audible alert, driven by PWM to produce a tone.
	220Ω Resistor	Limits current to the LED to prevent burnout and protect the Pi's GPIO pin.
Software/Libraries	Python 3.11	The primary programming language for the project.
	OpenCV	Used for essential image pre-processing (color conversion) and for displaying the final output.
	dlib	Used for its high-accuracy frontal face detector and the 68-point facial landmark prediction model.
	face_recognition	A high-level library used for the driver authentication feature, comparing the current driver to known faces.
	picamera2	The official library for interfacing with the Pi's camera, ensuring a stable video stream.
	RPi.GPIO	Python library used to control the Raspberry Pi's GPIO pins to activate the LED and speaker.

External Services	Pushover	A cloud service used to send high-priority, instant alarm notifications to the driver's smartphone.
	Twilio	A cloud communications platform used to make the automated emergency voice call to a third party.
	Macrodroid (Android App)	An automation app on the driver's phone, used to listen for the Pushover alert and send the location SMS.

Hyper-parameters Used:

The system's sensitivity is controlled by several key parameters within the Python script. These were fine-tuned through experimentation to provide a balance between responsiveness and avoiding false positives.

Parameter Name	Value	Description
EYE_AR_THRESH	0.25	The Eye Aspect Ratio below which the eye is considered closed.
EYE_AR_CONSEC_FRAMES	10f	Number of consecutive frames the EAR must be below the threshold for a drowsy alert.
YAWN_MAR_THRESH	0.5	The Mouth Aspect Ratio above which the mouth is considered to be yawning.
YAWN_CONSEC_FRAMES	15f	Number of consecutive frames the MAR must be above the threshold for a yawn alert.
NO_FACE_CONSEC_FRAMES	25f	Number of consecutive frames without a face detected to trigger a distraction alert.
Emergency Timer	20s	The duration an alert condition must persist before the emergency call is made.

4.2 Experimental Results

The system was tested under various simulated conditions to validate the logic of the multi-feature detection and the two-stage alert protocol. The results confirmed that the system responds correctly to each defined state.

Results of System State Testing: -

Tested Condition	Driver Action	System Response
Normal Driving	Face visible, eyes open, mouth closed.	No alerts. Green overlays are drawn on the face. Driver is identified as "Authorized" or "Unknown".
Drowsiness Event	Eyes remain closed for more than the threshold time.	**Stage 1:** Local LED and speaker activate. Pushover alarm sent to driver's phone. **Stage 2 (after 20s):** Twilio voice call is made to emergency contact, and Macrodroid sends a location SMS.
Yawning Event	Mouth remains open for more than the threshold time.	**Stage 1:** Local LED and speaker activate. Pushover alarm sent to driver's phone. **Stage 2 (after 20s):** Twilio voice call is made to emergency contact, and Macrodroid sends a location SMS.
Distraction Event	Driver looks away, making their face undetectable.	**Stage 1:** Local LED and speaker activate. Pushover alarm sent to driver's phone. **Stage 2 (after 20s):** Twilio voice call is made to emergency contact, and Macrodroid sends a location SMS.
Unauthorized Driver	An unknown face is detected by the system.	A one-time, non-emergency "Security Alert" notification is sent to the owner's phone. All safety monitoring features (drowsiness, yawn, distraction) remain fully active for the unknown driver to ensure their safety.

6. Comparison with the Existing Work

Standard driver drowsiness systems often rely on a single metric, typically just the Eye Aspect Ratio (EAR) for detecting eye closure. While effective to a

degree, these single-metric systems can be prone to false positives and lack a sophisticated response mechanism.

This project represents a significant enhancement over such basic systems in several key areas:

- **Multi-Factor Detection:** By combining three distinct indicators—eye closure (drowsiness), mouth opening (yawning), and face presence (distraction)—the system achieves a much higher level of confidence before triggering an alert. This approach reduces false positives that might occur from a simple glance away or a brief, non-fatigued eye closure.
- **Two-Stage Escalation Protocol:** A basic system might only feature a local buzzer. This project implements a sophisticated two-stage alert. The initial local and phone alarms serve to re-engage the driver, while the 20-second escalation to an automated voice call and location SMS provides a critical safety net in the event of a true emergency where the driver is unresponsive.
- **Driver Authentication:** The addition of facial recognition moves the system beyond a simple detector into a personalized security device. The "Guest Mode" logic, which alerts the owner to an unauthorized driver while still providing safety monitoring, is a feature not commonly found in simple open-source projects.

7. Conclusion and Future Work

Conclusion:

This project successfully demonstrates the development of a comprehensive, multi-featured driver safety system using Raspberry Pi 4. By integrating drowsiness, yawn, and distraction detection with a two-stage alert protocol that includes driver authentication and remote emergency notifications, the system provides a significant improvement over basic single-metric detector. The final implementation, which coordinates actions between the Raspberry Pi and the driver's smartphone, showcases a modern IoT approach for solving a critical real-world problem. The project successfully balances immediate driver feedback with an emergency failsafe, creating a reliable and intelligent safety companion.

Future Work:

While the current system is highly functional, several avenues exist for future enhancement:

1. **Blink Rate Monitoring and Fatigue Score:** A more proactive approach could be implemented by monitoring the driver's blink rate over time. A decreasing blink rate is a strong indicator of the onset of fatigue. This data could be used to calculate a "Fatigue Score," allowing the system to issue a "Take a break" warning *before* a dangerous micro-sleep event occurs.
2. **System Integrity Checks:** To improve reliability, a self-diagnostic feature could be added. This would involve checking for camera obstruction at startup and periodically verifying the internet connection required for the emergency alerts. The system could then provide specific warnings if a key component is non-functional.
3. **Dedicated Hardware:** For a permanent in-car installation, the components could be integrated into a custom 3D-printed enclosure, and a dedicated GPS module could be added to the Raspberry Pi to remove the dependency on the driver's phone for location data.

References

1) https://ieeexplore.ieee.org/document/8808931

2) https://doi.org/10.3390/app12031145

3) https://ieeexplore.ieee.org/document/9777952

4) https://www.mdpi.com/1424-8220/25/3/812