VisionWatch: Embedded Driver Safety System

Real-Time Fatigue Monitoring with Raspberry Pi and Computer Vision

Project Type: Open-source prototype / Personal research project

Developed By:

Name	**LinkedIn link**
J. Srikanth	LinkedIn
Kunwar Vikramaaditya Singh Katheria	LinkedIn
Navya Pathak	LinkedIn
Pronay Dutta	LinkedIn
P. Sai Teja	LinkedIn

Table of Contents

Abstract

Driver fatigue is a major contributor to road accidents, especially during long journeys or nighttime driving. In India, such incidents are alarmingly common, with many drivers relying on cruise control and falling asleep behind the wheel, assuming the vehicle will manage itself. A study by the Central Road Research Institute (CRRI) found that nearly 40% of highway accidents are caused by exhausted drivers dozing off while driving. On the Agra–Lucknow Expressway alone, over 54% of the 7,024 recorded accidents between 2021 and 2025 were linked to drowsy driving. The Government of India has acknowledged this issue and is working to reintroduce “driver drowsiness” as a distinct cause in national accident databases like e-DAR and iRAD.

To address this problem, this project presents a real-time Driver Drowsiness Detection System using a laptop webcam and Raspberry Pi. The system monitors the driver’s eye activity using computer vision techniques and calculates the Eye Aspect Ratio (EAR) from facial landmarks. If the driver’s eyes remain closed for more than 5 seconds, the laptop sends an alert to the Raspberry Pi, which then initiates a multi-stage alert mechanism: a voice warning, followed by a voice announcement, and finally simulating emergency contact if the driver remains unresponsive. The solution integrates OpenCV, dlib, Flask, and Raspberry Pi to demonstrate a cost-effective, scalable, and universally compatible embedded safety application that enhances road safety through proactive intervention.

Objectives

The primary objective of this project is to design and implement a real-time driver drowsiness detection and alert system using affordable and accessible embedded technologies. The specific goals are:

- Detect driver drowsiness by monitoring eye activity using a webcam and calculating the Eye Aspect Ratio (EAR) through facial landmark detection.
- Establish communication between a laptop (detection unit) and a Raspberry Pi (alert unit) using HTTP-based messaging via Flask and the requests library.
- Trigger a multi-stage alert mechanism on the Raspberry Pi when drowsiness is detected:
  - Stage 1: Issue a voice warning to alert the driver.
  - Stage 2: Provide a second voice announcement if the driver remains unresponsive.
  - Stage 3: Simulate emergency contact initiation if drowsiness continues.
- Display real-time feedback on the laptop screen, including EAR values and driver status (awake, drowsy, alert sent).
- Provide a cost-effective and scalable solution that can be implemented in any vehicle, regardless of existing driver assistance technologies.
- Enhance road safety by proactively preventing accidents caused by driver fatigue, particularly on highways and long-distance routes.

System Architecture

The Driver Drowsiness Detection System is designed as a simple and affordable embedded solution that combines computer vision and network communication. It consists of two main components: a Laptop (Sender Unit) and a Raspberry Pi (Receiver Unit), connected via a local Wi-Fi network. The system is lightweight, scalable, and suitable for integration into any vehicle.

1. Laptop (Sender Unit)

- Uses a webcam to monitor the driver’s face.
- Detects whether the driver’s eyes are open or closed.
- Calculates how long the eyes stay closed using Eye Aspect Ratio (EAR).
- If eyes are closed for more than 5 seconds, sends a warning signal to the Raspberry Pi.
- Displays real-time messages like “Drowsy” or “Awake” on the laptop screen.

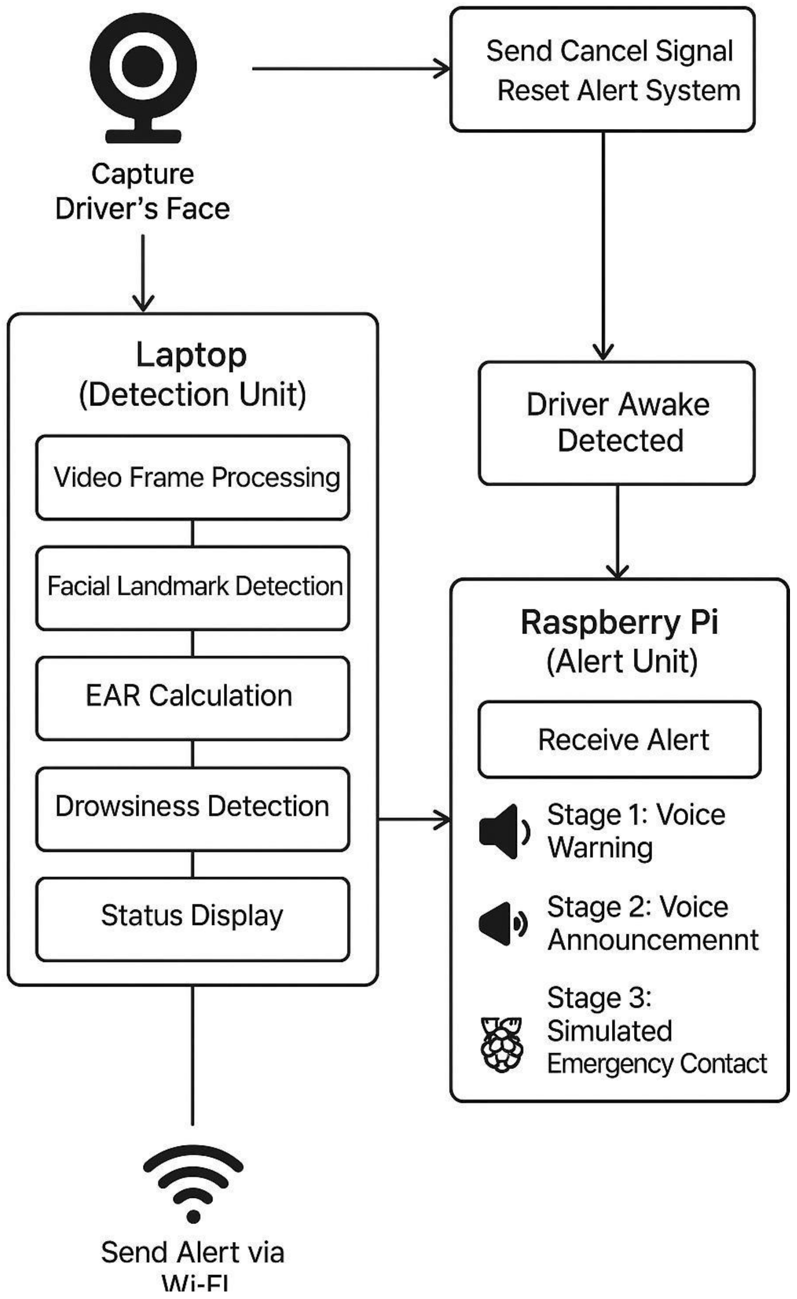
2. Raspberry Pi (Receiver Unit)

- Receives alert signals from the laptop over Wi-Fi.
- Starts a step-by-step voice alert process when drowsiness is detected.
- First gives a voice warning to wake the driver.
- Then gives a second voice announcement if the driver remains unresponsive.
- If the driver still doesn’t wake up, it simulates contacting emergency help.
- Cancels the alert and resets the system if the driver wakes up.

3. Communication Protocol

- The laptop and Raspberry Pi communicate using Wi-Fi.
- The laptop sends simple web requests to the Raspberry Pi using the Flask and requests libraries.
- These messages indicate whether the driver is drowsy or awake.

4. System Workflow



Hardware and Software Requirements

Hardware Requirements

- Laptop with Webcam
  Used to monitor the driver’s face and detect drowsiness using computer vision.

- Raspberry Pi (Model 3B or 4)
  Acts as the alert unit, receiving signals and generating voice warnings.

- Speaker (connected via AUX cable)
  Outputs voice alerts from the Raspberry Pi.

- Wi-Fi Network
  Enables communication between the laptop and Raspberry Pi.

Software Requirements

- Raspberry Pi OS
  Operating system installed on the Raspberry Pi.

- Python 3.x
  Programming language used for both detection and alert logic.

- OpenCV
  Used on the laptop to capture video and process frames.

- dlib
  Used for facial landmark detection and EAR calculation.

- Flask
  Runs a lightweight server on the Raspberry Pi to receive alerts.

- pyttsx3 or espeak
  Text-to-speech engine used to generate voice alerts.

- requests
  Python library used on the laptop to send alert messages to the Raspberry Pi.

Methodology

This project follows a structured, real-time workflow to detect driver drowsiness and trigger voice-based alerts using a laptop and Raspberry Pi. The system is designed to be lightweight, responsive, and easy to deploy in any vehicle.

Step 1: Continuous Monitoring

- The laptop uses its webcam to continuously capture video of the driver’s face.
- Each frame is analysed using computer vision to detect eye landmarks.
- The Eye Aspect Ratio (EAR) is calculated to measure how open or closed the eyes are.

Step 2: Drowsiness Detection Logic

- If the EAR remains below a set threshold for more than 5 seconds, the system identifies the driver as drowsy.
- This threshold helps avoid false alarms from normal blinking or brief eye closure.

Step 3: Alert Transmission

- Once drowsiness is detected, the laptop sends an alert to the Raspberry Pi using an HTTP POST request.
- The message includes the alert type and timestamp, ensuring accurate response timing.

Step 4: Voice-Based Alert Response

- The Raspberry Pi receives the alert and begins a staged voice response:
  - ➢ Stage 1: A short voice warning is played to grab the driver’s attention.
  - ➢ Stage 2: A louder and more urgent voice message is played if the driver remains unresponsive.
  - ➢ Stage 3: A final message simulates contacting emergency help.
- These stages escalate gradually to ensure the driver has multiple chances to respond.

Step 5: System Reset

- If the laptop detects that the driver has awakened (EAR returns to normal), it sends a cancel signal.
- The Raspberry Pi stops the alert and resets the system to standby mode.

This methodology ensures that the system operates in real time, minimizes false positives, and provides a clear escalation path to prevent accidents caused by fatigue.

Code

- Laptop (Detection Unit)

```
import cv2
import dlib
import numpy as np
from scipy.spatial import distance as dist
import requests
import time

RASPBERRY_PI_IP = "192.168.43.154"
RASPBERRY_PI_PORT = 5000
EAR_THRESHOLD = 0.25

class DrowsinessDetector:
    def _init_(self):
        self.detector = dlib.get_frontal_face_detector()
        self.predictor =
dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")
        self.LEFT_EYE = list(range(36, 42))
        self.RIGHT_EYE = list(range(42, 48))
        self.eye_closed_start_time = None
        self.alert_sent = False

    def eye_aspect_ratio(self, 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 get_landmarks(self, frame, rect):
        shape = self.predictor(frame, rect)
        coords = np.zeros((68, 2), dtype=int)
        for i in range(68):
            coords[i] = (shape.part(i).x, shape.part(i).y)
        return coords

    def send_alert_to_pi(self, alert_type, duration):
        try:
            url = f"http://{RASPBERRY_PI_IP}:{RASPBERRY_PI_PORT}/alert"
            data = {"type": alert_type, "duration": duration}
            response = requests.post(url, json=data, timeout=2)
            return response.status_code == 200
        except Exception as e:
            print(f"Error sending alert: {e}")
            return False

    def process_frame(self, frame):
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        faces = self.detector(gray, 0)
        status = "ALERT"
        ear = 0.0
        if len(faces) > 0:
            face = faces[0]
            landmarks = self.get_landmarks(gray, face)
            left_eye = landmarks[self.LEFT_EYE]
            right_eye = landmarks[self.RIGHT_EYE]
            left_ear = self.eye_aspect_ratio(left_eye)
            right_ear = self.eye_aspect_ratio(right_eye)
            ear = (left_ear + right_ear) / 2.0

            if ear < EAR_THRESHOLD:
                if self.eye_closed_start_time is None:
                    self.eye_closed_start_time = time.time()
                elapsed = time.time() - self.eye_closed_start_time
                if elapsed >= 5 and not self.alert_sent:
                    self.send_alert_to_pi("drowsy", elapsed)
                    self.alert_sent = True
                    status = "DROWSY - ALERT SENT"
                elif elapsed >= 5:
                    status = f"DROWSY - {elapsed:.1f}s"
                else:
                    status = f"EYES CLOSING - {elapsed:.1f}s"
            else:
                if self.alert_sent:
                    self.send_alert_to_pi("awake", 0)
                self.eye_closed_start_time = None
                self.alert_sent = False
                status = "AWAKE"
        else:
            status = "NO FACE DETECTED"
            self.eye_closed_start_time = None
            self.alert_sent = False

        cv2.putText(frame, f"Status: {status}", (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)
        cv2.putText(frame, f"EAR: {ear:.2f}", (10, 60),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)
        return frame

def main():
    detector = DrowsinessDetector()
    cap = cv2.VideoCapture(0)
    if not cap.isOpened():
        print("ERROR: Could not open webcam")
        return

    while True:
        ret, frame = cap.read()
        if not ret:
            break
        processed_frame = detector.process_frame(frame)
```

```
                cv2.imshow("Driver Drowsiness Detection",
                processed_frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()

if _name___ == "_main_":
    main()
```

- Raspberry Pi (Alert Unit)

```
from flask import Flask, request, jsonify
import threading
import time
import RPi.GPIO as GPIO
import pyttsx3

HAZARD_LIGHT_PIN = 17
BRAKE_PIN = 27

app = Flask(_name_)
tts_engine = pyttsx3.init()

alert_active = False
alert_thread = None
alert_stage = 0
alert_start_time = None

GPIO.setmode(GPIO.BCM)
GPIO.setup(HAZARD_LIGHT_PIN, GPIO.OUT)
GPIO.setup(BRAKE_PIN, GPIO.OUT)
GPIO.output(HAZARD_LIGHT_PIN, GPIO.LOW)
GPIO.output(BRAKE_PIN, GPIO.LOW)

def speak(text):
    try:
        tts_engine.say(text)
        tts_engine.runAndWait()
    except Exception as e:
        print(f"TTS Error: {e}")

def blink_hazard_lights():
    for _ in range(10):
        GPIO.output(HAZARD_LIGHT_PIN, GPIO.HIGH)
        time.sleep(0.5)
        GPIO.output(HAZARD_LIGHT_PIN, GPIO.LOW)
        time.sleep(0.5)

def alert_sequence():
    global alert_active, alert_stage, alert_start_time
```

```python
    alert_start_time = time.time()
    alert_stage = 1

    speak("Warning! Driver drowsiness detected. Please wake up
immediately!")
    for i in range(10):
        if not alert_active:
            return
        time.sleep(1)

    alert_stage = 2
    speak("Driver still unresponsive. Activating hazard lights.")
    hazard_thread = threading.Thread(target=blink_hazard_lights)
    hazard_thread.start()
    for i in range(10):
        if not alert_active:
            GPIO.output(HAZARD_LIGHT_PIN, GPIO.LOW)
            return
        time.sleep(1)

    alert_stage = 3
    GPIO.output(HAZARD_LIGHT_PIN, GPIO.LOW)
    GPIO.output(BRAKE_PIN, GPIO.HIGH)
    speak("Emergency braking initiated. Contacting emergency services.")
    time.sleep(2)
    speak("Emergency services have been notified. Vehicle location
transmitted.")

    while alert_active:
        time.sleep(1)

    GPIO.output(BRAKE_PIN, GPIO.LOW)
    GPIO.output(HAZARD_LIGHT_PIN, GPIO.LOW)

@app.route('/alert', methods=['POST'])
def receive_alert():
    global alert_active, alert_thread, alert_stage
    data = request.json
    alert_type = data.get('type', 'unknown')
    duration = data.get('duration', 0)

    if alert_type == 'drowsy' and not alert_active:
        alert_active = True
        alert_thread = threading.Thread(target=alert_sequence)
        alert_thread.start()
        return jsonify({"status": "alert_started"}), 200

    elif alert_type == 'awake':
        alert_active = False
        GPIO.output(HAZARD_LIGHT_PIN, GPIO.LOW)
        GPIO.output(BRAKE_PIN, GPIO.LOW)
        speak("Driver is awake. Alert cancelled.")
```

```
      if alert_thread:
            alert_thread.join(timeout=2)
        alert_stage = 0
        return jsonify({"status": "alert_cancelled"}), 200

    return jsonify({"status": "acknowledged", "current_stage":
alert_stage}), 200

@app.route('/status', methods=['GET'])
def get_status():
    return jsonify({
        "alert_active": alert_active,
        "alert_stage": alert_stage,
        "time_elapsed": time.time() - alert_start_time if
alert_start_time else 0
    }), 200

def cleanup():
    GPIO.cleanup()

if __name__ == '__main__':
    try:
        app.run(host='0.0.0.0', port=5000, debug=False)
    except KeyboardInterrupt:
        cleanup()
```

Testing and Results

To validate the system, the hardware was assembled as shown below. The laptop (detection unit) was positioned to capture the driver's face via webcam, while the Raspberry Pi (alert unit) was connected to a speaker via AUX cable and placed nearby for voice alerts.

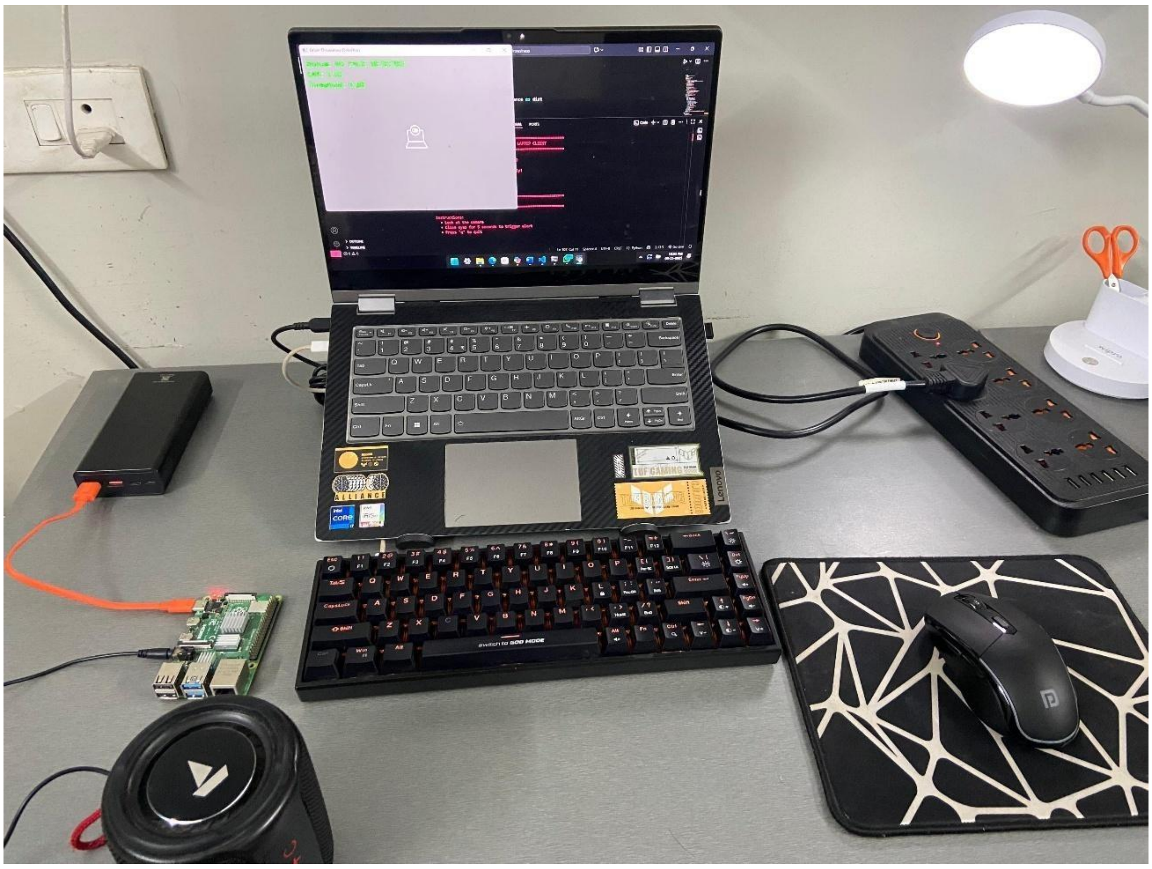


Figure 1: Hardware setup showing laptop and Raspberry Pi with speaker

Testing Procedure

To validate the functionality of the Driver Drowsiness Detection System, the following tests were conducted:

1. Facial Detection and EAR Monitoring

- Setup: Laptop webcam focused on the user's face.
- Action: User looked at the camera with eyes open and then closed eyes for more than 5 seconds.
- Expected Result: EAR drops below threshold; system detects drowsiness.

2. Alert Transmission to Raspberry Pi

- Setup: Laptop and Raspberry Pi connected to same Wi-Fi network.
- Action: Laptop sends HTTP POST request when drowsiness is detected.
- Expected Result: Raspberry Pi receives alert and initiates voice response.

3. Voice Alert Stages

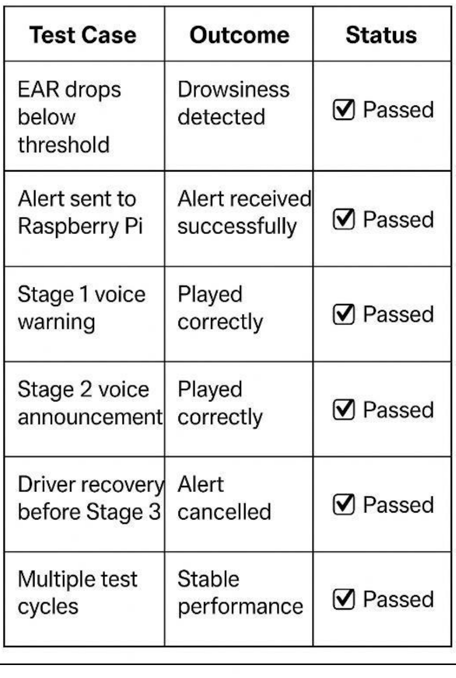
- Stage 1: Raspberry Pi plays initial warning message.
- Stage 2: Raspberry Pi plays a second, more urgent message.
- Stage 3: Raspberry Pi simulates emergency contact with a final voice message.

4. Recovery Detection

- Action: User opens eyes before Stage 3.
- Expected Result: Laptop sends "awake" signal; Raspberry Pi cancels alert and resets.

5. System Stability

- Tested: Multiple cycles of drowsiness and recovery.
- Result: System consistently detected drowsiness and responded correctly without crashes or false positives.



Test Case	Outcome	Status
EAR drops below threshold	Drowsiness detected	☑ Passed
Alert sent to Raspberry Pi	Alert received successfully	☑ Passed
Stage 1 voice warning	Played correctly	☑ Passed
Stage 2 voice announcement	Played correctly	☑ Passed
Driver recovery before Stage 3	Alert cancelled	☑ Passed
Multiple test cycles	Stable performance	☑ Passed

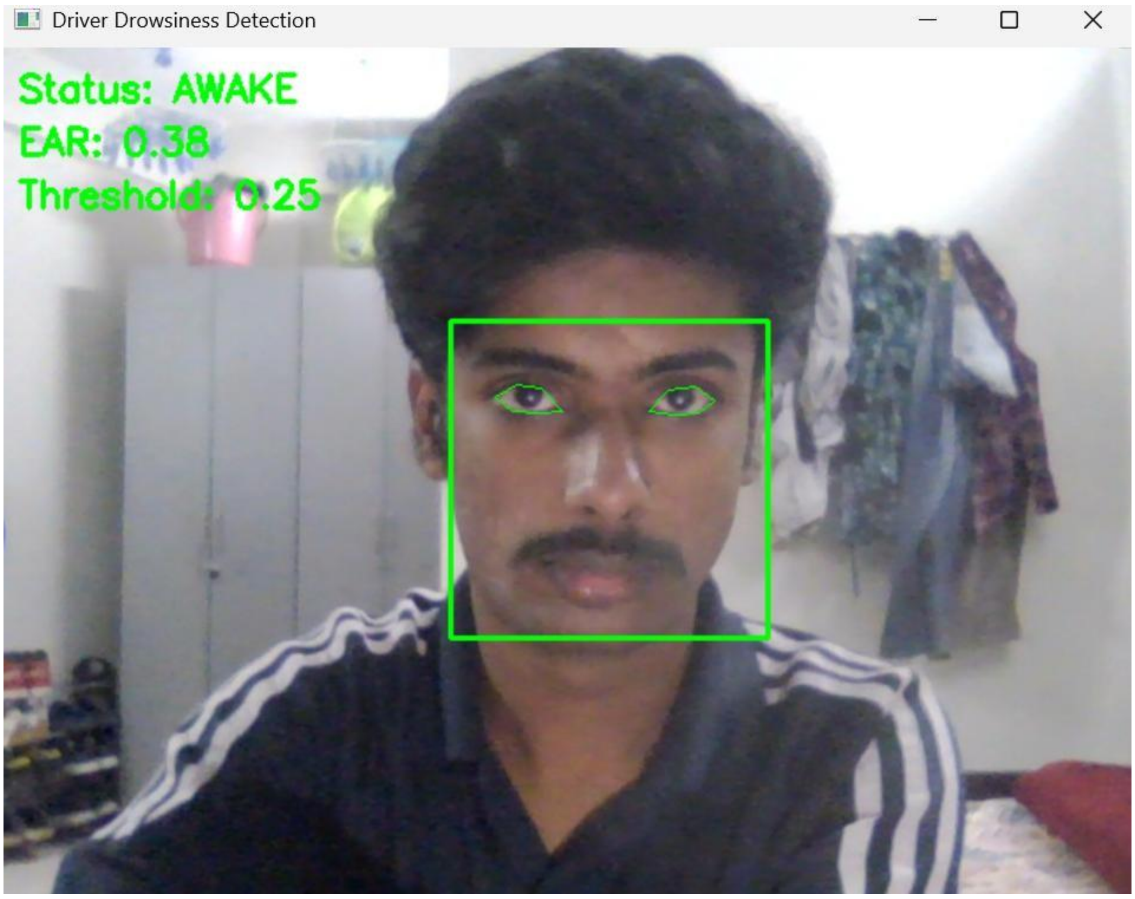
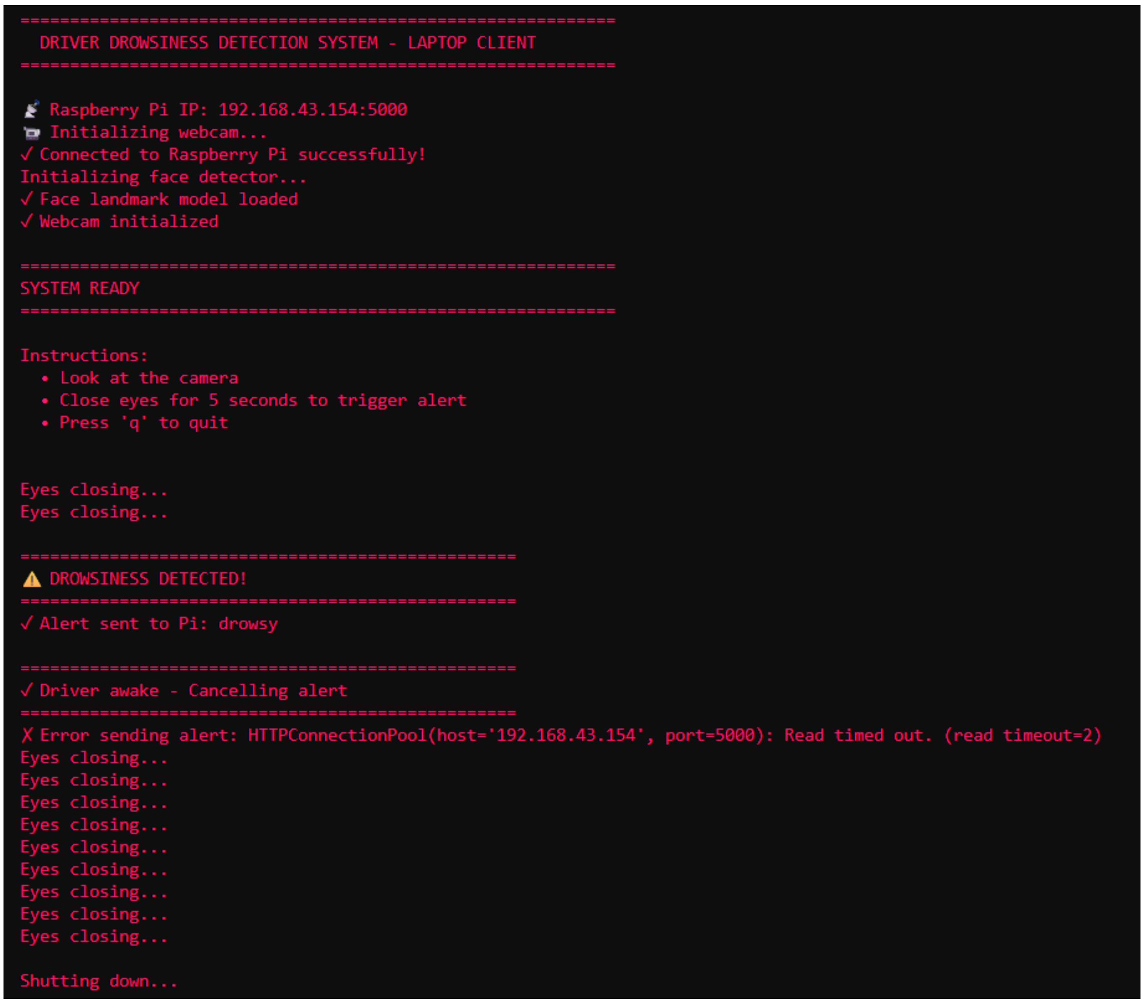
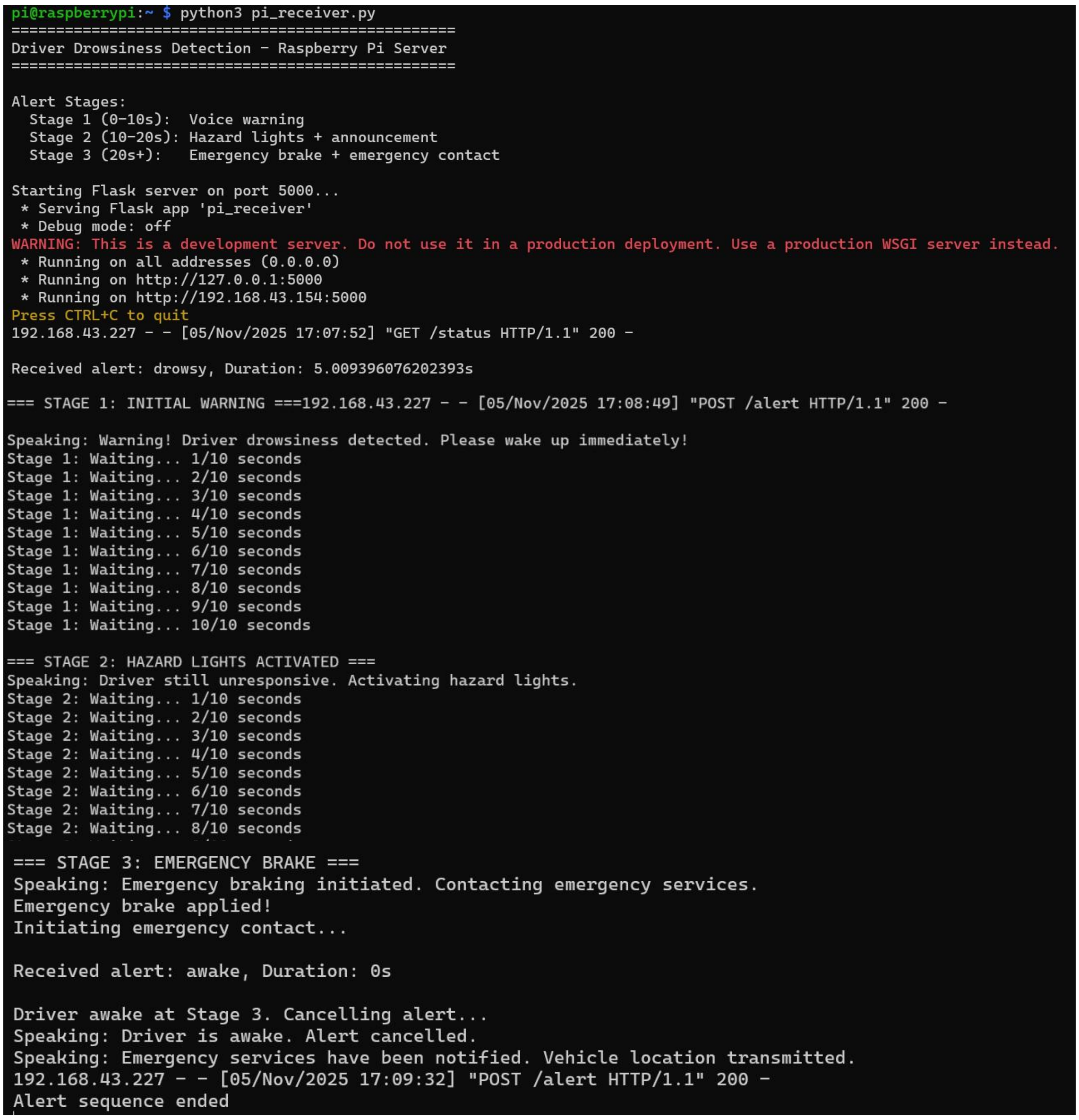


Figure 2: Webcam feed capturing driver's face for real-time monitoring



```
============================================================
  DRIVER DROWSINESS DETECTION SYSTEM - LAPTOP CLIENT
============================================================

📡 Raspberry Pi IP: 192.168.43.154:5000
📷 Initializing webcam...
✓ Connected to Raspberry Pi successfully!
Initializing face detector...
✓ Face landmark model loaded
✓ Webcam initialized

============================================================
SYSTEM READY
============================================================

Instructions:
  • Look at the camera
  • Close eyes for 5 seconds to trigger alert
  • Press 'q' to quit


Eyes closing...
Eyes closing...

==================================================
⚠ DROWSINESS DETECTED!
==================================================
✓ Alert sent to Pi: drowsy

==================================================
✓ Driver awake - Cancelling alert
==================================================
✗ Error sending alert: HTTPConnectionPool(host='192.168.43.154', port=5000): Read timed out. (read timeout=2)
Eyes closing...
Eyes closing...
Eyes closing...
Eyes closing...
Eyes closing...
Eyes closing...
Eyes closing...
Eyes closing...
Eyes closing...

Shutting down...
```

Figure 3: EAR detection and drowsiness alert on laptop



```
pi@raspberrypi:~ $ python3 pi_receiver.py
==================================================
Driver Drowsiness Detection - Raspberry Pi Server
==================================================

Alert Stages:
  Stage 1 (0-10s):  Voice warning
  Stage 2 (10-20s): Hazard lights + announcement
  Stage 3 (20s+):   Emergency brake + emergency contact

Starting Flask server on port 5000...
 * Serving Flask app 'pi_receiver'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://192.168.43.154:5000
Press CTRL+C to quit
192.168.43.227 - - [05/Nov/2025 17:07:52] "GET /status HTTP/1.1" 200 -

Received alert: drowsy, Duration: 5.009396076202393s

=== STAGE 1: INITIAL WARNING ===192.168.43.227 - - [05/Nov/2025 17:08:49] "POST /alert HTTP/1.1" 200 -

Speaking: Warning! Driver drowsiness detected. Please wake up immediately!
Stage 1: Waiting... 1/10 seconds
Stage 1: Waiting... 2/10 seconds
Stage 1: Waiting... 3/10 seconds
Stage 1: Waiting... 4/10 seconds
Stage 1: Waiting... 5/10 seconds
Stage 1: Waiting... 6/10 seconds
Stage 1: Waiting... 7/10 seconds
Stage 1: Waiting... 8/10 seconds
Stage 1: Waiting... 9/10 seconds
Stage 1: Waiting... 10/10 seconds

=== STAGE 2: HAZARD LIGHTS ACTIVATED ===
Speaking: Driver still unresponsive. Activating hazard lights.
Stage 2: Waiting... 1/10 seconds
Stage 2: Waiting... 2/10 seconds
Stage 2: Waiting... 3/10 seconds
Stage 2: Waiting... 4/10 seconds
Stage 2: Waiting... 5/10 seconds
Stage 2: Waiting... 6/10 seconds
Stage 2: Waiting... 7/10 seconds
Stage 2: Waiting... 8/10 seconds
=== STAGE 3: EMERGENCY BRAKE ===
Speaking: Emergency braking initiated. Contacting emergency services.
Emergency brake applied!
Initiating emergency contact...

Received alert: awake, Duration: 0s

Driver awake at Stage 3. Cancelling alert...
Speaking: Driver is awake. Alert cancelled.
Speaking: Emergency services have been notified. Vehicle location transmitted.
192.168.43.227 - - [05/Nov/2025 17:09:32] "POST /alert HTTP/1.1" 200 -
Alert sequence ended
```

Figure 4: Raspberry Pi receiving alert and triggering voice response

Applications and Future Scope

Applications
This system has strong potential for real-world deployment in various domains:

- Personal Vehicles
  Helps individual drivers stay alert during long journeys, especially at night.
- Commercial Transport
  Can be integrated into trucks, buses, and delivery vehicles to reduce fatigue-related accidents.
- Fleet Management Systems
  Offers centralized monitoring and alerting for logistics companies managing multiple drivers.
- Driving Schools and Training Centers
  Useful for educating new drivers about the dangers of fatigue and demonstrating safety technology.
- Public Safety Campaigns
  Can be showcased in awareness programs to promote road safety and responsible driving.

Future Scope
To enhance the system's capabilities and impact, the following improvements are proposed:

- GPS Integration
  Automatically transmit location data during emergency alerts for faster response.
- Mobile App Extension
  Allow alerts and status updates to be sent to a connected smartphone for remote monitoring.
- Cloud-Based Logging
  Store drowsiness events and system performance data for long-term analysis and reporting.
- Multi-Factor Detection
  Combine EAR with yawning, head tilt, and blink rate for more accurate drowsiness detection.
- Vehicle Control Integration
  Interface with vehicle systems to trigger automatic braking or lane correction in critical cases.
- Voice Customization and Multilingual Support
  Enable alerts in regional languages to improve accessibility and user comfort.

Conclusion

This project successfully demonstrates a real-time Driver Drowsiness Detection System using a laptop and Raspberry Pi. By monitoring eye activity and triggering staged voice alerts, the system helps prevent fatigue related accidents in a cost-effective and scalable way.

Testing confirmed reliable detection, smooth communication between devices, and consistent alert responses. With future enhancements like GPS, mobile integration, and cloud logging, this system can evolve into a powerful safety tool for both personal and commercial vehicles.