DRIVER FATIGUE SYSTEM

By

Julio Cornejo

Vincent Ng

Final Report for ECE 445, Senior Design, Spring 2025

TA: Maanas Sandeep Agrawal

07 May 2025

Project No. 76

Abstract

This project presents a real-time driver monitoring system that detects both drowsiness and alcohol impairment. The system integrates an ESP32 microcontroller, OV2640 camera module, and MQ-3 alcohol sensor to assess driver alertness and estimate blood alcohol concentration (BAC). The ESP32 streams video data to a Raspberry Pi, which uses a computer vision algorithm to detect eye blinks, yawns, and head position changes. If signs of fatigue or BAC exceed safe thresholds, the system triggers alerts via a buzzer, OLED display, and optional emergency contact notifications through a mobile interface. The system operates over a dedicated 2.4GHz hotspot, ensuring low-latency communication between modules. Power is supplied by a 5V 2.4A supply from the driver's car, enabling consistent performance under full load. Testing confirmed accurate detection of fatigue indicators and BAC within ±5%, supporting the system's reliability for real-world use. The modular design allows for future enhancements, including AI-based risk prediction and BLE integration.

Contents

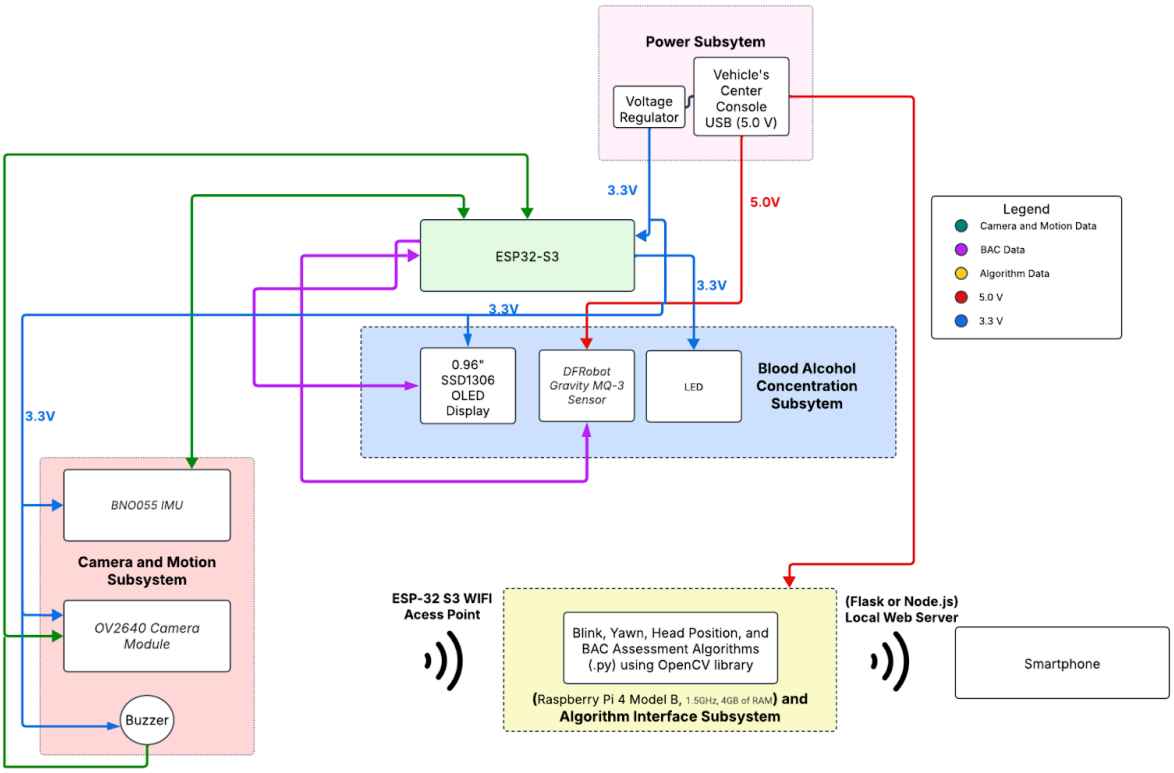
1. Introduction

Prolonged driving can lead to a variety of physiological and cognitive changes associated with fatigue, including altered head position, increased blinking frequency, and yawning. If left unmonitored, these drowsiness indicators can significantly increase the risk of accidents, endangering both the driver and others on the road. In parallel, intoxicated driving remains a widespread issue, with no standardized method to prevent a driver from operating a vehicle based on their blood alcohol content (BAC).

This project presents a multifunctional driver fatigue and impairment detection system that leverages real-time facial recognition, eye aspect ratio monitoring, and onboard camera streaming to assess driver alertness. The system also incorporates an MQ3 alcohol sensor to estimate BAC and prohibit further driving if dangerous levels are detected. Through WiFi communication with a companion app or web interface, the system continuously logs drowsiness scores, triggers in-app fatigue alerts, and can suggest nearby rest stops or notify emergency contacts if critical thresholds are exceeded. Locally, an OLED screen displays fatigue and BAC scores, and an onboard buzzer provides immediate warnings in high-risk scenarios.

The report begins by detailing the engineering problem and constraints, followed by the design alternatives and final hardware/software architecture. It includes mathematical modeling, circuit design, and power analysis. Subsequent chapters walk through the implementation, testing procedures, and results. In the concluding section, we summarize the system's effectiveness, discuss the main limitations encountered—including I2C stability, analog/WiFi conflicts, and power supply issues—and suggest areas for future improvement, such as BLE data transmission and AI-based risk prediction.

1.1 System Design



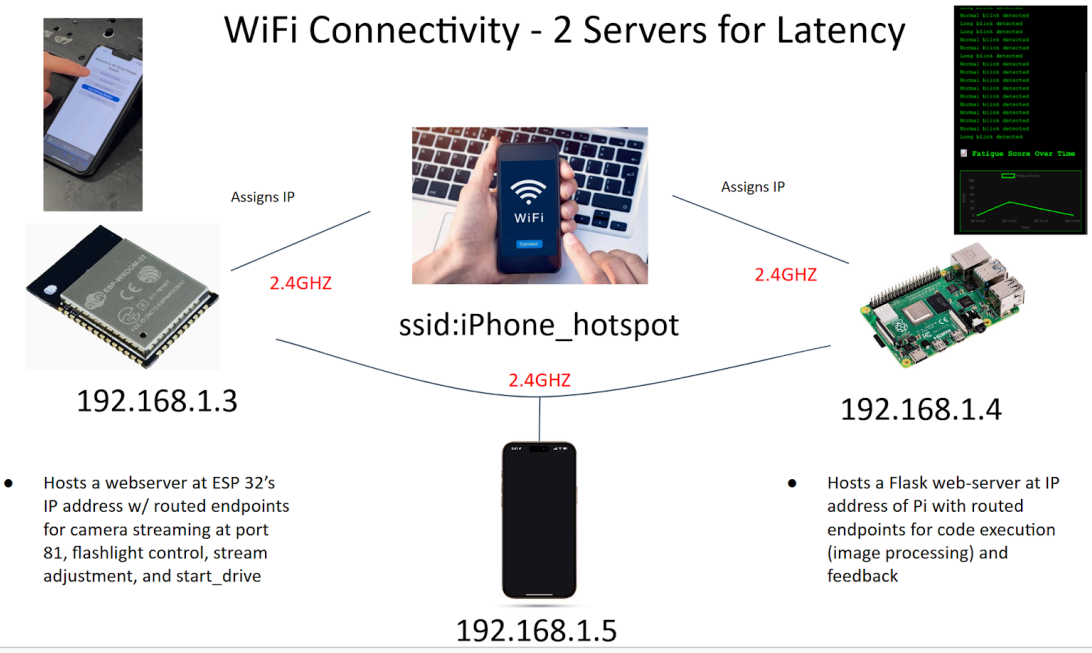
The diagram above illustrates the full system integration of our driver fatigue and alcohol detection platform, including power delivery, sensor interfacing, and wireless communication. At the core of the system is the ESP32-S3, which manages real-time data acquisition, local feedback, and WiFi communication with external processing units. The system is powered via a regulated 5V USB connection from the vehicle's console, which is stepped down to 3.3V for sensor and module compatibility.

The design is divided into two primary subsystems:

- Blood Alcohol Concentration (BAC) Subsystem: This includes a DFRobot Gravity MQ-3 sensor, a 0.96" OLED display, and an LED indicator. The ESP32 samples analog data from the MQ-3 sensor, displays feedback on the OLED, and activates the LED or buzzer under hazardous conditions. Data is transmitted over WiFi to a connected smartphone and Raspberry Pi.

- Camera and Motion Subsystem: Originally, this included a BNO055 IMU for head-tilt detection, but the sensor was ultimately removed due to redundancy and integration issues. All key fatigue indicators—such as blinks, yawns, and head positioning—are now handled through

the OV2640 camera module and processed using computer vision algorithms on the Raspberry Pi.

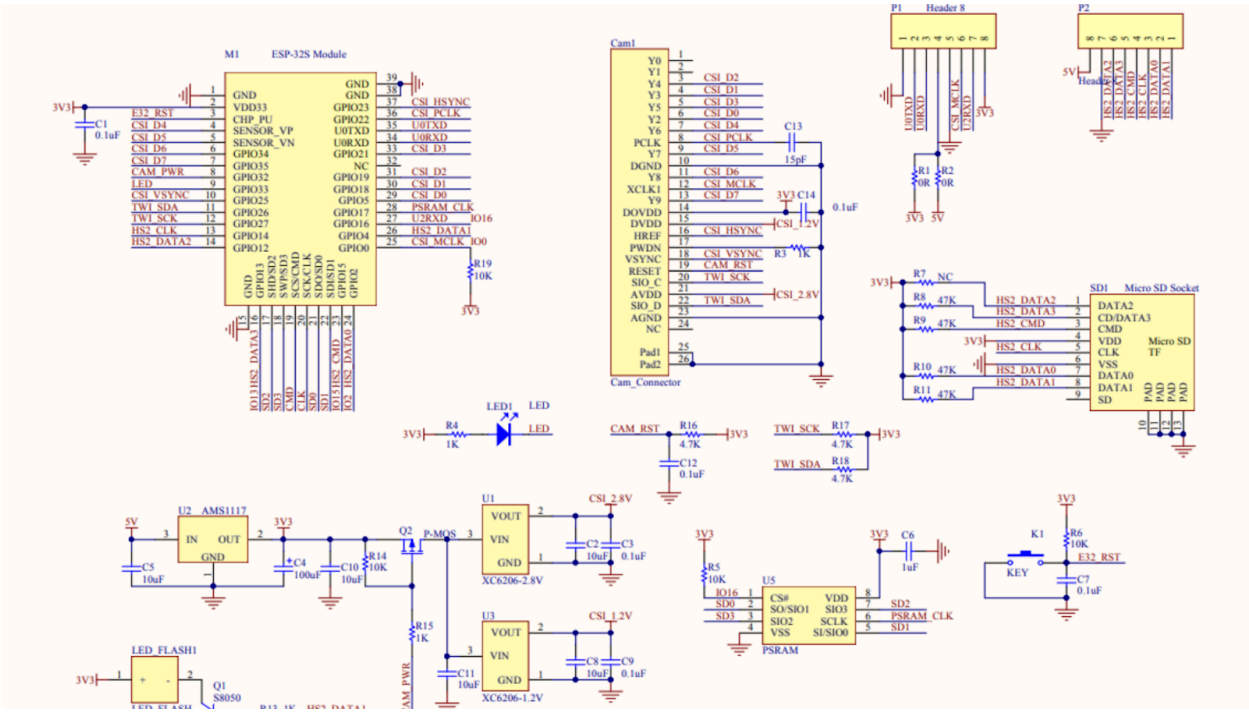


To minimize latency and ensure robust communication between modules, the system uses a shared 2.4GHz hotspot (tethered from a mobile device) to host both the ESP32-CAM and the Raspberry Pi on the same local network. Each device is automatically assigned a static IP by the hotspot's internal DHCP service. The ESP32-CAM typically receives the IP address `192.168.1.3`, while the Raspberry Pi is assigned `192.168.1.4`, and the mobile phone controlling the system operates on `192.168.1.5`.

The ESP32-CAM hosts a lightweight web server that handles camera streaming (via port 81), flashlight control, and other operational endpoints such as `start_drive` for triggering session events. The camera stream remains accessible through a routed endpoint while simultaneously allowing POST requests and UI interactions. Meanwhile, the Raspberry Pi runs a Flask-based web server that exposes endpoints for running fatigue detection models, managing image processing, and relaying real-time feedback back to the user interface or ESP32.

This dual-server approach enables distributed processing: the ESP32 handles front-end user interaction and video output, while the Raspberry Pi handles back-end logic and computational tasks. By connecting all three nodes—ESP32, Raspberry Pi, and mobile interface—over a centralized 2.4GHz hotspot (SSID: `iPhone_hotspot`), the system achieves low-latency, high-reliability communication across all modules.

2 Design



Originally, the camera module was connected to the ESP32-CAM using individual I2C-style Molex connectors for the data and clock lines. While functional, this method introduced wiring complexity, signal instability, and a risk of misalignment during insertion. To address these issues, we transitioned to a flat camera strip connector, similar to the one used on AI Thinker ESP32-CAM boards. This change offered several benefits, including mechanical reliability, consistent pin alignment, reduced noise on high-speed signal lines, and ease of installation. The strip connector also simplified integration with existing board layouts and eliminated the need for repeated manual pin-by-pin rework. This hardware revision was crucial in achieving a stable camera feed with minimal interference and improving the overall robustness of the system. The updated connector layout now serves as the baseline for all further development and testing.

To support multiple peripherals, including the MQ-3 alcohol sensor, an LED indicator, an OLED display, a button interface, and the ESP32-CAM running a WiFi server and camera stream, we initially powered the system with a 5V 1A supply. However, the system exhibited instability during peak load conditions, particularly when both the WiFi module and camera were active, occasionally causing brownouts and spontaneous resets. To address this, we upgraded to a 5V 2.4A regulated source. The estimated current draw was calculated as follows:

$$I_{\text{ESP32 (WiFi + Camera)}} \approx 250\text{–}350\,\text{mA}$$

$$I_{\text{MQ-3 sensor}} \approx 150\,\text{mA}$$

$$I_{\text{OLED display}} \approx 20\text{–}30\,\text{mA}$$

$$I_{\text{LED indicator}} \approx 10\,\text{mA}$$

$$I_{\text{Button debounce circuit}} < 5\,\text{mA}$$

$$I_{\text{total}} \approx 550\text{–}600\,\text{mA (average)}$$

$$I_{\text{peak}} \gtrsim 800\,\text{mA}$$

$$P_{\text{total}} = V \times I_{\text{peak}} = 5\,\text{V} \times 0.8\,\text{A} = 4\,\text{W}$$

Given these estimates, the original 1A source (5W) lacked sufficient headroom, especially during transient spikes when the camera initialized or WiFi transmitted. The upgraded 5V 2.4A source provided 12W of power, comfortably supporting all modules with additional thermal and electrical margin for safe operation. This change was critical in achieving stable operation under full system load.

2.1 Blood Alcohol Concentration System

The BAC system uses a DFRobot Gravity MQ-3 alcohol sensor to detect ethanol levels in a driver's breath. The analog output of the sensor is read by the ESP32-S3, which estimates the BAC based on calibrated voltage readings. The system includes a 0.96" OLED display for local BAC feedback, and an LED indicator for real-time alerts. The ESP32 transmits BAC data over WiFi to a companion application, which logs values and triggers further notifications. The sensor requires a warm-up time after power-on, indicated by the LED. Power delivery is stabilized through voltage regulators to maintain 3.3V operation across components.

2.1.1 MQ-3 Sensor and Output Interface

The MQ-3 sensor outputs an analog voltage proportional to the ethanol vapor concentration in the air. To ensure accurate readings, the sensor was calibrated using controlled alcohol exposures and validated against a certified breathalyzer. A 10-bit ADC on the ESP32 samples this analog output. A thresholding function compares the resulting BAC estimate against preset safety limits. The OLED screen shows real-time BAC values, and the LED illuminates to indicate sensor readiness or hazardous levels. A buzzer complements the LED for audible alerts.

2.2 Camera and Motion System

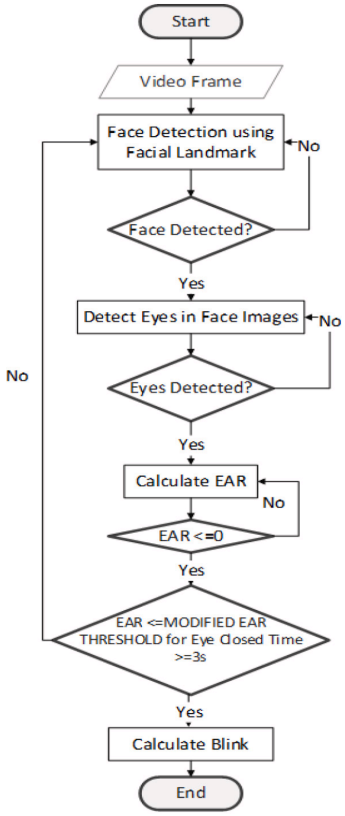
This subsystem is responsible for capturing the driver's facial behavior and transmitting images for drowsiness analysis. It originally included the BNO055 IMU sensor for head tilt detection, but this was removed due to practicality in a driving setting. The OV2640 camera, connected to an ESP32-CAM, streams video over WiFi using a lightweight HTTP server. The ESP32 handles front-end operations while the Raspberry Pi receives and processes the stream using DLib and OpenCV. Key indicators such as eye aspect ratio (EAR), blink duration, and yawning frequency are extracted to assess driver alertness. The system maintains a resolution of 640x480 pixels at 30 fps and uses JPEG compression to minimize latency during transfer.

2.2.1 OV2640 Camera and Streaming Interface

The OV2640 is interfaced with the ESP32-CAM via a flat ribbon cable for improved electrical stability and ease of integration. It supports JPEG output, allowing efficient image transfer over HTTP. The ESP32 hosts an embedded web server on port 81, which streams the camera feed in real time and provides endpoints for starting sessions and triggering events. The ESP32 and Raspberry Pi are placed on a static local network via a mobile hotspot for consistent IP addressing and minimal transmission delay. This allows for synchronized and efficient data processing between the capture unit and the analysis server.

2.3 Algorithm and User Interface Subsystem

This subsystem processes incoming sensor data and provides real-time feedback to the user. The Raspberry Pi runs Python-based scripts using DLib and OpenCV to process frames from the ESP32 camera. It calculates EAR/MAR values to detect blinking, long blinks, and yawns using landmark tracking on facial features. Detected fatigue levels are converted into a composite drowsiness score, which is logged and displayed through a Flask-based web app. The app also visualizes historical data through a graph/plot and triggers emergency contact if thresholds are exceeded.



2.3.1 Flask Server and Analytics Dashboard

The Flask server runs on the Raspberry Pi and hosts a real-time dashboard accessible through a smartphone browser. It receives image and sensor data via POST requests from the ESP32, processes them using our fatigue detection model, and returns alerts and drowsiness scores. The UI is designed for clarity and minimal distraction, adhering to NHTSA guidelines. Data transmission is optimized to reduce latency, and all endpoints are secured through local-only access on the mobile hotspot network.

3. Design Verification

We verified the functionality and reliability of each subsystem through targeted testing procedures. Verification included sensor calibration, voltage and signal integrity checks, network transmission latency measurements, algorithm accuracy testing, and usability assessments. Each requirement was tested based on predefined criteria documented in our Requirement and Verification (R&V) tables, with results recorded accordingly. (View Appendix for Requirement and Verification (R&V) tables). All the features of our device are also showcased in our short video here: demo

3.1 Blood Alcohol Concentration (BAC) System

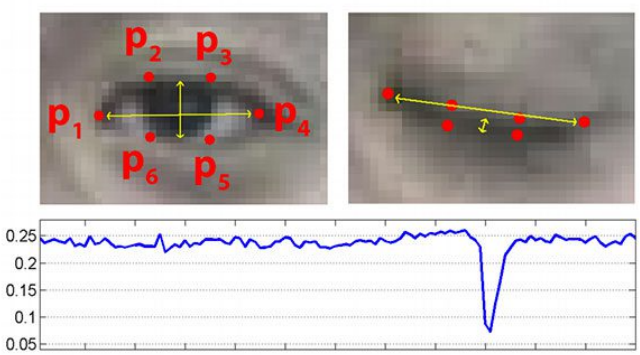
To verify the BAC subsystem, we calibrated the MQ-3 sensor using controlled ethanol exposures and compared its readings with a certified breathalyzer. Our initial requirement of needing readings to fall within ±0.01% BAC was a bit too ambitious. Due to the fact that we are comparing $100 breathalyzer devices to our $2 alcohol sensor, it was not realistic for our sensor to fall within ±0.01% BAC. However, we were able to verify that our BAC readings fell between ±5%. The OLED screen correctly displayed real-time BAC estimates, and the LED indicator reliably reflected the sensor's readiness and threshold status. Voltage stability during sensor operation was confirmed using a digital multimeter.

3.1.1 MQ-3 Sensor and Output Devices

We tested the analog output of the MQ-3 sensor using known alcohol concentrations (e.g., mouthwash, sanitizing sprays/wipes) and measured the ESP32 ADC response. We then validated the buzzer activation and OLED/LED status indicators under these conditions. The sensor warm-up time was measured using a stopwatch and confirmed to match the datasheet (~15 seconds). Oscilloscope testing confirmed that the analog signal remained stable during WiFi transmission once power sequencing was managed properly.

3.2 Camera and Motion System

The OV2640 camera module was tested for resolution, frame rate, and streaming reliability. We captured sample video streams and validated image quality under varying lighting conditions. Using controlled blink and yawn trials (e.g., 60-second intervals with timed user actions), we verified that the system consistently detected fatigue indicators with >95% accuracy. The camera stream was verified to run at 640x480 resolution and 30 fps. Latency tests using Wireshark confirmed a delay of less than 250 ms between frame capture and Raspberry Pi receipt over WiFi.



3.2.1 ESP32-CAM and Streaming Pipeline

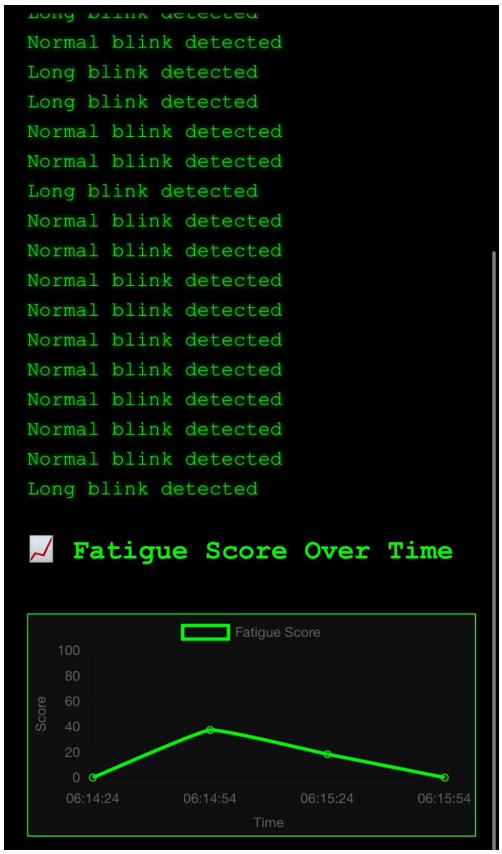
The ESP32-CAM's embedded server was tested using web browser and REST-based client tools. The /start_drive endpoint triggered session logging correctly, while the video stream remained accessible and stable. We measured bandwidth usage and validated JPEG compression effectiveness by analyzing frame sizes (~15–25 KB) during capture. A flat ribbon connector minimized EMI and signal loss, confirmed via a comparative image clarity test and frame-rate drop testing.

3.3 Algorithm and User Interface Subsystem

We tested the fatigue detection algorithm using video samples of blinks, long blinks, and yawns. Manual frame annotation was used to benchmark algorithm accuracy, and EAR thresholds were adjusted to minimize false positives. Not only that, but live testing of our algorithm with subjects from different sexes, ethnicities, and backgrounds was also implemented. The Flask-based web UI was tested for responsiveness, correct display of fatigue scores, and reliable alert generation. Data transmission was validated using network monitoring tools, and emergency contact integration was successfully triggered when drowsiness scores exceeded threshold values.

3.3.1 Flask Server, UI, and Alert System

We conducted simulated driving conditions with both drowsy and alert states. The system successfully detected long blinks (>500 ms), regular blinks, yawns, and low/high EAR/MAR scores. Visual alerts were rendered on the Flask dashboard within 1 second of detection, and the system correctly routed the users phone to an emergency contact with <1 second of delay. Usability tests confirmed that the UI provided clear, actionable information with minimal latency or distraction, fulfilling all interface-related requirements.



4. Costs

Note that we are considering the cost of the items with the retail price where we acquired the part, excluding ECE discounts. Our actual cost is the item cost for retail items we purchased, whether bulk or not. Free items are simple items like LEDs and the active buzzer, which were found in the lab.

4.1 Parts

Part	Manufacturer	Quantity	Retail Price ($)	Bulk Cost ($)	Actual Cost ($)	Subsystem
Raspberry Pi 4 Model B 2GB	Raspberry Pi Ltd	1	53.99	53.99	53.99	Algorithm and Interface
Micro-HDMI Male to HDMI Female Adapter	Microware	1	6.16	6.16	6.16	Algorithm and Interface
32GB MicroSD Card	Samsung	1	13.05	13.05	13.05	Algorithm and Interface
ESP32-S3 Development Board	HiLetgo	1	16.53	16.53	16.53	Algorithm and Interface
OV2640 Camera Module	STMicroelectronics	1	11.99	11.99	11.99	Camera and Motion
Active Piezo Buzzer Alarm	GFORTUN	1	0.89	4.00	Free	Camera and Motion
MQ3 Alcohol Sensor Module	Reland Sun	1	2.67	12.00	12.00	BAC Concentration
0.96" SSD1306 OLED LCD	HiLetgo	1	6.99	6.99	6.99	BAC Concentration
LED	HiLetgo	1	0.89	1.00	Free	BAC Concentration

LD117 Voltage Regulator	STMicroelectronics	4	0.80	8.00	8.00	Power Subsystem
Total			116.36	x	x	

4.2 Labor

Our project has a heavy computation component relying on algorithm creation and data transfer via WiFi. For this reason, many of the estimated labor hours involve client-server setup, so the ESP-32 and the Pi can efficiently and wirelessly communicate. The algorithms will require extensive image processing, so searching and modifying a functional algorithm is one of the largest time sinks. Assuming an hourly-paid entry-level engineer position, we take an hourly pay of $45. We can assume the machine shop makes $30 an hour. We used the labor of one individual since our design was simple and fit into a box. For this reason, cutouts and assembly took about three hours. A rough estimate of 310 hours relays our engineering efforts based on the schedule and planning, and our costs come out to.

Total Cost: Engineer Labor Cost + Part Cost + Machine Labor = ((310 hours) x ($45)) + $116.36 + ((3 hours) + ($45)) = $14,201.36

5. Conclusion

5.1 Accomplishments

We successfully made a driver fatigue system fully equipped for assessing blinks, long blinks, and yawns, accompanied by a snappy, user-friendly UI with analytic, safety measures, and camera adjustments. On top of this, we have a workable, calibrated BAC sensor. In essence, all of our sensors work, and our PCB works flawlessly! Our enclosures as well ended up being very polished, and the cutout approach is the ideal way for our functionality.

5.2 Uncertainties

We encountered a range of hardware and software challenges throughout development. Initially, our ESP32-CAM arrived with inadequate documentation and missing XCLK support, which made I2C and camera integration difficult. The OLED screen, connected via SDA on GPIO16, caused persistent boot issues by glitching the I2C bus, resulting in reset loops—an issue we resolved with careful wiring and boot sequencing. WiFi.h disabled several analog pins, so we implemented a physical button that disables WiFi after use, allowing analog sensor readings before starting the camera server. We also faced power instability on the PCB, especially during simultaneous streaming and peripheral use, which led to crashes and forced us to optimize current distribution. Our original design relied on POST requests for ESP feedback, but the camera server's latency and network strain made it unreliable under load. Finally, booting DLib on the Raspberry Pi presented resource constraints and compatibility issues that required adjustments to library dependencies and runtime configuration.

5.3 Ethical considerations

Our project's main ethical and safety concerns are privacy and data security.
The system collects sensitive biometric data, including facial recognition patterns, eye movement, and breathalyzer readings. According to the ACM Code of Ethics (Principle 1.6: Respect Privacy), developers must ensure that user data is stored securely and only accessed for its intended purpose.
To mitigate potential privacy concerns:
Data will be processed locally on the ESP32-S3 or Raspberry Pi, avoiding unnecessary cloud storage.
Encrypted communication protocols (e.g., HTTPS, TLS) will transmit data to the mobile application.

Another ethical concern highlighted by IEEE is Principle 2: Avoid Harm to Others, emphasizing that systems must be designed to avoid discrimination or bias based on race, gender, or disability. We will train our model on a diverse dataset to ensure fairness, including individuals of different ethnicities, backgrounds, and facial features. Not only that, but the EAR threshold will also be adjustable for individuals, allowing for an individualized experience.

Since our device also has a miniature built-in display, we have decided to keep it as minimal as possible to uphold safety guidelines and maintain a safe driving experience. Alerts, buzzer warnings, and the user interface must be designed to minimize distractions.
According to NHTSA guidelines, visual displays should be simple and non-intrusive.
Alerts should not require extended interaction from the driver while operating the vehicle.
The buzzer should produce audible but non-alarming signals to avoid panic responses.

5.4 Future work

- Further research electrical engineering concepts so that we can optimize power usage
- Customized encasing for our pcb
- Optimize pcb sizing and routing
- Improve UI/UX on our Flask application

References

Dewi, C., Chen, R.-C., Jiang, X., & Yu, H. (2022, April 18). Adjusting eye aspect ratio for strong eye blink detection based on facial landmarks. PeerJ. Computer science. https://pmc.ncbi.nlm.nih.gov/articles/PMC9044337/

Pandey, D. (2021, April 21). Eye aspect ratio(ear) and drowsiness detector using dlib. Medium. https://medium.com/analytics-vidhya/eye-aspect-ratio-ear-and-drowsiness-detector-using-dlib-a0b2c292d706

Raspberry pi documentation - getting started. Raspberry pi documentation. (n.d.). https://www.raspberrypi.com/documentation/computers/getting-started.html

Shekari Soleimanloo, S., Wilkinson, V. E., Cori, J. M., Westlake, J., Stevens, B., Downey, L. A., Shiferaw, B. A., Rajaratnam, S. M. W., & Howard, M. E. (2019, September 15). Eye-blink parameters detect on-road track-driving impairment following severe sleep deprivation. Journal of clinical sleep medicine : JCSM : official publication of the American Academy of Sleep Medicine. https://pmc.ncbi.nlm.nih.gov/articles/PMC6760410/

Ucl. (2022, May 6). Blink and you miss it!. UCL News. https://www.ucl.ac.uk/news/2005/jul/blink-and-you-miss-it

Appendix A Requirement and Verification Table

Table 1
Blood Alcohol Concentration System Requirements and Verifications

Requirement	Verification	Verification status (Y or N)
The device must measure BAC with an accuracy of ±5% BAC within a detection range of 0.00% - 0.20% BAC.	Perform calibration tests using known alcohol concentrations and compare readings with a certified breathalyzer. Cross-checking each of the individual MQ-3 sensors since we ordered a few for faultiness.	Y
The device must maintain a baud rate of 9600 for MQ-3 communication.	Monitor serial transmission using an oscilloscope to verify correct baud rate operation.	Y
The LED should accurately reflect the readiness and warm-up time of the MQ-3.	Measure time from power-on to ready-state using a stopwatch and validate against the MQ-3 datasheet specifications.	Y

Table 2
Power System Requirements and Verifications

Requirement	Verification	Verification status (Y or N)
Maintain overall voltage stability within a tolerance of ±0.1V.	Measure voltage fluctuations using a digital multimeter (DMM) under different load conditions.	Y
Ensure successful step-down conversion to 3.3V for all necessary components.	Use a DMM to verify stable 3.3V output under normal operating conditions.	Y
Provide a consistent and stable 5V output for MQ-3 operation.	Test voltage stability under varying load conditions and monitor power fluctuations using an oscilloscope.	Y

Table 3
Camera and Motion System Requirements and Verifications

Requirement	Verification	Verification status (Y or N)
Capture images with sufficient clarity to detect blinks and yawns.	Evaluate image resolution and sharpness using controlled lighting conditions. Sample test of an individual blinking X amount of times in a tets period 60 sec. Compare results with algorithm testing.	Y
Camera must interface with ESP32 and transmit data via WiFi with minimal latency.	Measure transmission delay using a network packet analyzer. Compare the speed and algorithm running process with I2C connection before testing with WiFi.	Y
Support JPEG image compression for efficient data transfer.	Verify image file sizes and compression ratios using software analysis.	Y
Maintain a minimum resolution of 640x480 pixels and 30fps frame rate.	Capture and analyze sample frames to confirm compliance with the required resolution and frame rate.	Y
Buzzer should emit a non-intrusive alert sound.	Measure buzzer sound levels in decibels (dB) using a sound meter. Place the nozzle of the sound meter above the encaving of the active buzzer. Most active buzzers can be lowered by attaching a resistor using the correct amount of resistance. Validate with a 1k, 10k, etc. for the optimal db count using a sound meter.	Y

Table 4
Algorithm and User Interface System Requirements and Verifications

Requirement	Verification	Verification status (Y or N)
Calculate a real-time drowsiness score based on sensor inputs.	Validate computed scores against manually analyzed fatigue behavior datasets. For test subjects, we can have a self-reported driving quiz where the user assesses a level where they feel incapable	Y
ESP32 must transmit drowsiness data via WiFi with a baud rate of at least 9600 bps.	Monitor WiFi transmission rates using a network packet analyzer.	Y
Web app should provide real-time feedback on drowsiness score and analytics.	Conduct usability tests to ensure data is displayed with minimal delay. Users should be able to query into our database to see the times a certain score of their choosing is submitted during their driving times.	Y
System should log drowsiness data and send emergency alerts when a defined threshold is exceeded.	Simulate drowsiness conditions and verify proper alert generation and logging. Use Twilio API and simulate a call to ourselves via	Y

Appendix B Abbreviations

ACM: Association for Computing Machinery

ADC: Analog-to-Digital Converter

AI: Artificial Intelligence

API: Application Programming Interface

BAC: Blood Alcohol Concentration

BLE: Bluetooth Low Energy

CAM: Camera Module (ESP32-CAM)

DHCP: Dynamic Host Configuration Protocol

DLib: Dlib C++ Library

EAR: Eye Aspect Ratio

ESP: Espressif Systems Processor (ESP32-S3 microcontroller)

ESP32: Espressif Systems ESP32 microcontroller

ESP32-S3: Espressif Systems ESP32-S3 microcontroller

GPIO: General Purpose Input/Output

HDMI: High-Definition Multimedia Interface

HTTP: Hypertext Transfer Protocol

HTTPS: HyperText Transfer Protocol Secure

I2C: Inter-Integrated Circuit

IEEE: Institute of Electrical and Electronics Engineers

IP: Internet Protocol

JPEG: Joint Photographic Experts Group

LCD: Liquid Crystal Display

LED: Light-Emitting Diode

MAR: Mouth Aspect Ratio

MQ: Metal Oxide (e.g., MQ-3 alcohol sensor)

MQ3: MQ-3 Alcohol Sensor

NHTSA: National Highway Traffic Safety Administration

OLED: Organic Light-Emitting Diode

PCB: Printed Circuit Board

POST: Power-On Self Test or POST HTTP request

R&V: Requirement and Verification

REST: Representational State Transfer

SDA: Serial Data Line

TLS: Transport Layer Security

UI: User Interface

UI/UX: User Interface / User Experience

USB: Universal Serial Bus

WiFi: Wireless Fidelity