Safe Sight – Driver Safety Device & App

Matthew Carvajal, Albert Lougedo, Mahyar Mahtabfar

University of Central Florida, College of Engineering and Computer Science, Orlando, Florida, 32816

***Abstract* — This paper presents the design methodology utilized to develop the Safe Sight system, an intelligent driver assistance prototype integrating computer vision and radar sensing. The safety functions of Safe Sight fall into three primary categories: (1) Traffic Signal Detection through real-time video processing; (2) Human Obstacle Distance Measurement using radar sensing; and (3) Alert generation for driver attentiveness and safety feedback. This paper also discusses the implementation of embedded software algorithms enabling reliable, low-cost performance on a Raspberry Pi and ESP32 platform.**

***Index Terms* — Computer vision, driver assistance systems, driver monitoring, embedded systems, intelligent transportation systems, object detection, radar systems, real-time processing.**

I. Introduction

The *Safe Sight* system was developed to demonstrate how embedded intelligence and real-time sensing can improve driver attentiveness and traffic light awareness. Designed as a practical application of computer vision, accelerometer, and radar technologies, the system integrates two independent processing platforms—the ESP32 microcontroller and the Raspberry Pi—to create a coordinated driver assistance prototype. The ESP32 serves as the central control unit, handling all logic operations, radar/accelerometer data collection, and alert generation, while the Raspberry Pi performs visual and traffic light recognition while managing application-level communication.

With two USB webcams connected to the Raspberry Pi, the system continuously captures video data from the driver and the front of the vehicle, which is analyzed using OpenCV-based algorithms and YOLO object detection models to identify traffic lights, obstacles, and driver attentiveness. The extracted information is sent to the ESP32 for processing, where logic routines determine appropriate responses based on environmental conditions and detection results. The microcontroller also synchronizes the radar and accelerometer readings to avoid pedestrians while confirming the change in inertia.

The system's hardware architecture includes a regulated DC 12V power supply from the vehicle that distributes stable voltage levels to each component. A 5.2V regulator powers the Raspberry Pi and peripheral cameras, while a 3.3V regulator provides consistent current to the ESP32, accelerometer, and radar sensor modules. This configuration ensures electrical isolation, stable operation, and protection from voltage fluctuations during extended use.

The innovation of *Safe Sight* lies in its distributed processing design and cost-effective implementation using accessible hardware. By combining visual and sensor readings, the system provides a robust foundation for intelligent driver assistance, scalable to larger transportation safety applications.

II. System Overview

The system is best presented in terms of system components. Whether purchased or designed, these components are the ones that created the final product. This is a semi-technical introduction to each of the components included in this section.

A. Microcontroller

The ESP32 microcontroller serves as the central processing unit for the *Safe Sight* system, responsible for executing all core logic, decision-making algorithms, and sensor coordination tasks. Equipped with dual-core processing, integrated Wi-Fi and Bluetooth modules, and multiple GPIO interfaces, the ESP32 provides both computational performance and communication flexibility within a compact, low-power architecture. It receives processed vision data from the Raspberry Pi via UART, interprets traffic signals, driver awareness along with human presence, and initiates corresponding responses such as alert generation and data logging. In addition to logic control, the ESP32 interfaces directly with the radar and accelerometer sensor to collect distance and motion data, which are used to verify and enhance environmental awareness. The microcontroller also manages serial communication between system modules and ensures synchronized operation across all sensing and feedback components. The system is programmed and debugged using the Arduino IDE and its built-in compiler (GNU C++ compiler), making developing super efficient and user-friendly.

B. Power regulators (3.3V, 5.2V)

The application of the power regulators was used to verify the voltage being delivered to the components from a 12V source. The 12V - 3.3V regulator used for this design was the LM2576S, the values used for the components on this board matched directly with the values used for the original Texas Instruments design of this board as it is a constant output, not varying.

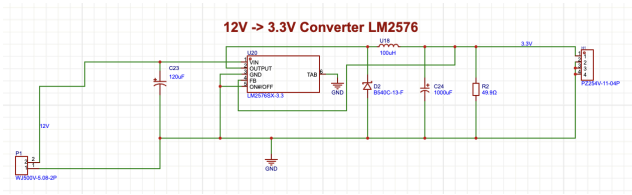

Fig 1. 12-3.3V Schematic

The design for the 5.2V converter was similar, however a bit different. In order to meet the 200mV over the 5V value we are seeking, we needed to adjust the board design in order to house a voltage divider circuit prior to the DC signal reaching the output. The application of a 3.24k Ohm and 1k Ohm resistor helped tune the output voltage to meet our design needs.

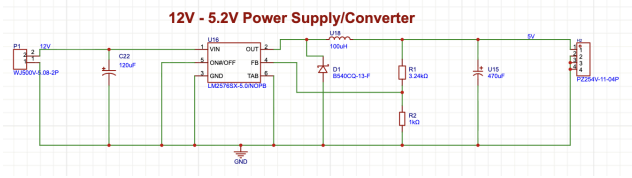

Fig 2. 12-5V Schematic

C. Radar

The radar implemented in this system was the cm4001 24GHz radar from DF Robot. This radar was chosen for the characteristics of its operating frequency being 24GHz, the size of the radar being a single chip, and the distance being projected from the radar which allows for accurate and responsible safety measurements in a dynamic traffic environment. Unlike PIR sensors that detect only motion, the CM4001's operation on the 24GHz millimeter-wave radar and Frequency Modulated Continuous Wave (FMCW) technology detects both static and moving humans. This is essential where an occupant might be unconscious or motionless. The FMCW radar provides real-time high resolution distance measurements, making it more precise and less prone to false positives or missed detections than many other conventional technologies. This sensor also proves robust as it is able to detect humans regardless of lighting conditions, temperature changes, dust, humidity or ambient noise. These conditions make it extremely suitable for vehicles where environmental conditions can be unpredictable, especially in the Florida climate. The device has also demonstrated capability to detect presence through materials like fabric, glass, or even parts of a car interior. This demonstration expands the installation flexibility to give much more freedom to the user. In regards to the integration benefits for the radar, the range being uo to 16 meters for presence detection and a 100° horizontal field view, it can cover a significant portion of the car interior and monitor for human presence around the vehicle. Containing both UART and I2C protocols, simplifying integration with popular microcontrollers such as our ESP32, ideal for this project.

D. Accelerometor

Using an MPU6050 accelerometer in a car safety application such as ours offers many significant benefits for which we were able to utilize. The MPU6050 measures acceleration along three axes, *x, y,* and *z.* This 3-axis plane for reference allowed the detection of sudden changes in velocity with high sensitivity and reliability. For example, when applying acceleration to a vehicle, the inertia taking place on all objects housed within the vehicle was recorded in the negative X direction which directly correlated to the orientation of the device. In order to align our firmware code to our field testing, the orientation of the device would determine the values of acceleration and de-acceleration being either positive or negative. The built-in gyroscope tracks the vehicle's angular movement and tilt, it is especially helpful for detection of vehicle orientation changes as we just described. The combination of a 3-axis accelerometer and a 3-axis gyroscope in one small affordable chip simplified the circuit design and made it ideal for our project. In regards to the field testing, we identified that the accelerometer has adjustable measurement ranges ranging from +/-2g to +/- 16g giving us the ability to fine tune the sensitivity for different vehicles and environments as the force of inertia is felt more on a car that seats the passengers higher from the ground than a car who houses the passengers closer to the surface of the earth. This motion/ tilt threshold adjustment has allowed for us to distinguish between normal driving and idle conditions, minimizing false positives. This digital nature of the sensor ensures that even the small changes in movement are captured.

E. Piezoelectric Buzzer

For the implementation of a signal to the user, we decided to go with an auditory signal to alert the driver to regain their attention to the road, detect pedestrians during a driving state. The piezoelectric buzzer was an ideal decision for this system as it allowed us to operate the buzzer with logic, giving full range to activate the audio signal given our specific conditions, this also included the level of noise being produced for specific scenarios. The functionality of the buzzer is then due to the signals received from the rest of the components in the system

which allows the ESP32 to act on the noise signal accordingly.

F. Raspberry Pi 5

The Raspberry Pi 5 functions as the vision and communication processor within the *Safe Sight* architecture. Leveraging its quad-core ARM CPU, GPU acceleration, and Linux-based operating environment, the Raspberry Pi 5 executes the system's computer vision tasks and manages data exchange between the user interface and the ESP32 controller. Two connected USB webcams provide continuous video input, which is processed through OpenCV libraries and YOLO-based object detection models to identify traffic lights, pedestrians, and driver attentiveness. The Raspberry Pi 5 transmits the interpreted detection data to the ESP32 via UART communication link for further decision-making and system response. In addition to visual processing, the Pi facilitates network communication and application control, enabling system monitoring and configuration through a graphical interface. Its computational resources and versatility make it ideal for handling complex image processing workloads while maintaining synchronization with the embedded control system.

G. USB Cameras

In order to capture all visual data, the system employs two USB camera modules to provide continuous visual data for environmental and driver monitoring. A 1080p camera (Logitech Brio 100) is mounted forward-facing and dedicated to traffic light detection. It delivers high-resolution imagery that enables accurate color and state recognition of traffic signals under varying lighting conditions. A secondary 720p camera (Muhoop USB Webcam) is oriented toward the driver and used for attentiveness monitoring, capturing facial orientation and head movement data to assess focus and potential distraction. Both cameras interface directly with the Raspberry Pi 5 via USB connections, ensuring stable frame capture and consistent synchronization with the computer vision algorithms. The dual-camera configuration allows Safe Sight to simultaneously process environmental and behavioral visual cues, enhancing overall situational awareness and system reliability.

H. Application

To tie all of the data together, an iOS application acts as a perfect tool for the user to access all of their data. Developed in Swift, the app provides a user-friendly interface for performance monitoring and data visualization. The application communicates with the Raspberry Pi 5 through HTTP-based data exchange, where the Pi acts as a hotspot. This enables remote access to the stored trip data and captured images. It features a modular design consisting of three primary views: the Driver Report view presents the number of driver distractions recorded over the ten most recent trips; the Photo Gallery allows users to review images captured during distraction events and traffic light detections; and the Profile Screen displays key metrics such as the user's safety score, total recorded distractions, and application version information. This mobile interface enhances system accessibility by delivering processed data in a clear, intuitive format while maintaining seamless integration with the embedded hardware. Through the iOS application, Safe Sight extends its functionality beyond real-time sensing to include post-trip analysis, improving driver awareness and long-term behavioral feedback.

III. System Concept

The concept of this system is to offer vehicles made before the integration of advanced computers the ability to increase safety confidence in many areas with a device that is capable of being transported from one vehicle to another. This system would also be capable of ensuring driver attention is retained on the road in order to minimize distracted driving on busy roads that inevitably increase the risk of accidents to other automobiles and pedestrians alike. Since the birth of the smartphone, there have been many instances of human attention being taken away from everyday tasks especially in work, schooling, and commuting. Everywhere you go now in modern day society, the substance of human interaction is increasingly being replaced with the integration of technology, enabling its users to constantly turn their attention to the phone even with a couple seconds to spare at a traffic stop. This has been so adopted in the modern day that many people do not consider this to be a "big deal" but when it comes to driving, a couple seconds could be the difference between preventing or causing a tragedy, and this is where the motivation for safe sight originated.

A. Communications Protocol

The Safe Sight system employs UART (Universal Asynchronous Receiver/Transmitter) as the primary communication protocol for robust, low-latency data exchange between the Raspberry Pi 5 (vision processor) and the ESP32 (central logic controller). The Raspberry Pi continuously transmits a concise, 2-byte data packet to the ESP32. The first byte encodes the driver's attentiveness status, and the second byte encodes the traffic light state.

Upon receiving these bytes over the UART bus, the ESP32's firmware initiates its core decision logic. This logic compares the current visual status from the Pi with the simultaneous readings from its local sensors: the MPU6050 accelerometer (indicating vehicle motion/inertia) and the DFRobot C4001 radar (indicating the presence of a pedestrian). By cross-referencing these four critical inputs (Attentiveness, Light State, Motion, and Pedestrian Presence), the ESP32 determines if a dangerous combination exists (e.g. Idle at a green light and driver is distracted) and if so, triggers the appropriate audible warning via the piezoelectric buzzer or sends a capture command back to the Pi to log the event.

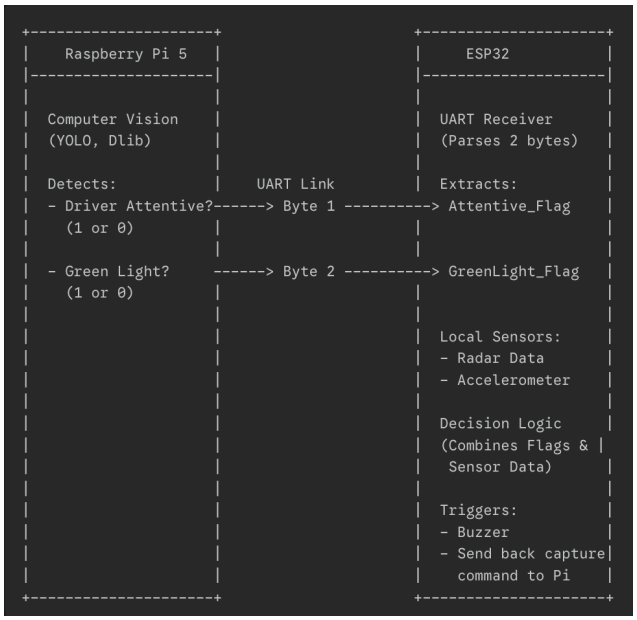

Fig.3 Safe Sight Logic Handling

B. Equations

The Safe Sight system relies on precise measurements from two key sensor types: the DF Robot C4001 mmw Radar for distance sensing and the MPU-6050 Accelerometer for motion sensing. The embedded firmware uses foundational equations from radar theory and kinematics to interpret the raw sensor data into meaningful physical quantities. The DF Robot C4001 mmw Radar Human Presence Detection Radar has firmware that implements a small set of well-known radar signal processing equations plus some detection/thresholding math. Below listed are the key equations you would expect to find in its firmware.

$$f_{tx}(t) = f_0 + \frac{B}{T_s}t \quad (1)$$

$$\tau = \frac{2R}{c} \quad (2)$$

$$f_b = K \bullet \tau \text{ with } K = \frac{B}{T_s} \quad (3)$$

$$R = \frac{cT_s}{2B}f_b = \frac{c}{2K}f_b \quad (4)$$

f_0 = start frequency, B = chirp bandwidth, T_s = chirp (sweep) time, R = Range

The MPU-6050 is a 6-DOF MEMS IMU with a 3-axis accelerometer and 3-axis gyroscope. The accelerometer RAW linear output uses standard kinematics equations for its sensor data interpretations for m/s^2.

$$Raw_{accel\ x,y,z} = \frac{a_{x,y,z}}{S_a} \quad (5)$$

S_a = accelerometer sensitivity (LSB per g)

V. Hardware Detail

The hardware architecture of the safe sight system was designed to power, transmit, and maintain all signal integrity throughout the board in order to properly establish communication in any condition that it may come across in traffic. This execution relied heavily on thickness of traces to reduce resistances and ensure ample power is being delivered to all components. The design also included a ground pour on the PCBs that acted as a shield and reduced interference from other components such as inductors and capacitors used to ground DC signals. The via locations were also thoughtfully placed in order to allow for sufficient heat dissipation in order to not place any of the IC internal temperature in a state that would compromise robustness, ensuring that the system is efficient in any temperature environment. From the power regulator boards to the piezoelectric buzzers, every single component in the system holds a vital role to ensure all of the system communicates effectively to deliver on the goals presented before the system.

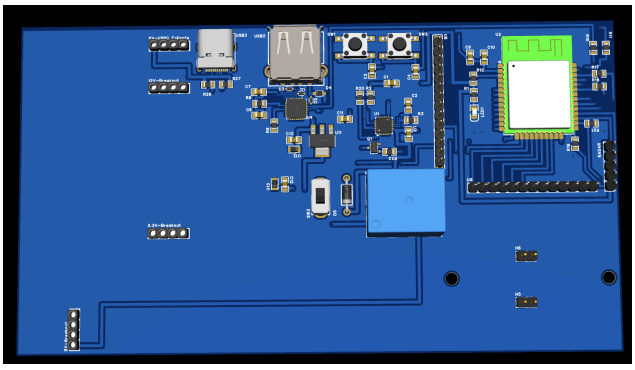

Fig. 4 Main PCB Board

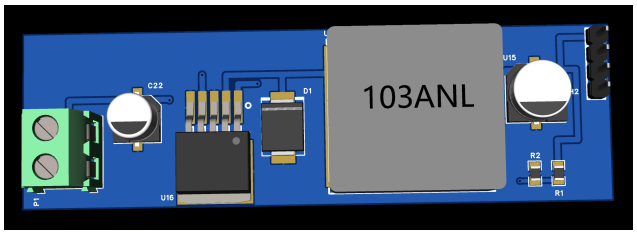

Fig.5 - 5V PCB Breakout Power Board

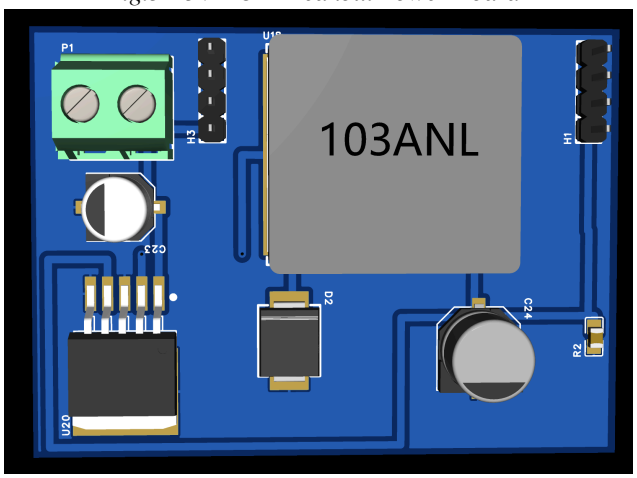

Fig. 6 - 3.3V PCB Breakout Power Board

VI. Software Detail

The software architecture of the Safe Sight system was designed to coordinate real-time sensing, decision-making, and data sharing between multiple embedded platforms. Its operation relies on the seamless interaction between the ESP32 microcontroller, the Raspberry Pi 5, and the iOS mobile application. Each software layer performs a specific role, from data acquisition and analysis to system logic and user feedback.

A. ESP32 Firmware

The ESP32 microcontroller operates as the central logic controller for the Safe Sight system, continuously monitoring vehicle motion, radar data, and driver attentiveness signals from the Raspberry Pi. The firmware is implemented in C++ using the Arduino framework and coordinates three main subsystems: motion detection, radar sensing, and inter-processor communication.

Upon initialization, the firmware configures all communication buses, including I²C for the MPU6050 accelerometer, UART for the DFRobot C4001 radar sensor, and a secondary UART interface for serial communication with the Raspberry Pi. The ESP32 also initializes a digital output pin to control an onboard speaker, which serves as an auditory alert mechanism.

During runtime, the firmware continuously acquires sensor data within the main loop. The MPU6050 accelerometer measures longitudinal acceleration along the X-axis, allowing the system to classify four vehicle motion states: acceleration, deceleration, idle motion, and idle still. These states are determined by comparing current acceleration readings against fixed threshold values. The previous and current motion states are stored to ensure stable transitions and prevent false classifications when the driver is stopped at a light.

```
// ------- Initialize all LOGIC VARIABLES --------
bool attentive, greenLight, pedDetected = false;
enum MotionState{
  ACCELERATION,
  DECELERATION,
  IDLE_MOTION,
  IDLE_STILL
};
// Car State Variables
MotionState carState;
MotionState prevState;
```

Fig. 7 - All logic variable initialization

Simultaneously, the C4001 radar sensor provides range readings used to detect nearby pedestrians. When an object is detected within a specified distance window (typically 1–3 meters), the firmware flags a pedestrian detection event. The radar module operates in speed detection mode, with configurable minimum and maximum range thresholds and internal fretting detection enabled for enhanced stability.

In parallel, the ESP32 receives encoded signals from the Raspberry Pi through the UART2 interface. The Pi transmits single-byte status codes that represent combinations of traffic light color and driver attentiveness. Each code (e.g., 0x11, 0x10, 0x01) is parsed by a switch-case structure to update two logical flags: greenLight and attentive. These flags reflect real-time results from the vision subsystem and directly influence the alert logic.

```
// Pi Serial Data
// Read Serial Data, IF......ELSEif....ELSEif
while (PiSerial.available()) {
  uint8_t c = PiSerial.read();
  switch(c) {
      case 0x11:
          greenLight = true;
          attentive = true;
          Serial.println("Green Light, Attentive");
          break;
      case 0x01:
          greenLight = false;
          attentive = true;
          Serial.println("No Green, Attentive");
          break;
      case 0x10:
          greenLight = true;
          attentive = false;
          Serial.println("Green, Distracted");
          break;
      default:
          greenLight = false;
          attentive = false;
          Serial.println("No green, distracted");
          break;
  }
}
```

Fig. 8 ESP32 reading bytes sent from Pi's CV algorithms.

Once all sensor and communication inputs are processed, the firmware executes its core decision logic. If the vehicle is idle at a green light and the driver is inattentive, the ESP32 activates the speaker and transmits a feedback code to the Raspberry Pi to log the event by taking a picture of the driver. Likewise, if the car is in motion and the driver is distracted—especially when a pedestrian is detected—the system issues an immediate audible warning and records the occurrence. These logic pathways ensure that the driver receives prompt, situation-specific alerts corresponding to dangerous combinations of distraction, motion, and environmental context.

```
//---------- UART TRANSMIT-----------
PiSerial.write(0x01);
Serial.println("Not paying attention while in motion");
```

Fig. 9 ESP32 sends 0x01 to the Pi to tell it to take a picture.

Through this closed-loop design, the ESP32 firmware effectively integrates motion sensing, radar data, and visual feedback into a unified decision-making system. Its modular implementation allows for rapid processing, low-latency communication, and scalable adaptation to additional sensors or external systems.

B. Raspberry Pi 5 Vision and Communication

The Raspberry Pi 5 operates as the Safe Sight system's vision processor and network interface, executing all image analysis and communication tasks. The software, written in Python, integrates OpenCV, Dlib, and Ultralytics YOLOv8 frameworks to detect driver attentiveness and recognize traffic light states in real time. The Raspberry Pi also handles UART-based serial communication with the ESP32 and manages local storage of annotated visual data.

Upon initialization, the program configures two USB webcams—a forward-facing 1080p camera dedicated to traffic light detection and a rear-facing 720p camera directed at the driver for attentiveness monitoring. The YOLO model (best_traffic_nano_yolo.pt) is loaded for frame-by-frame traffic signal recognition, while Dlib's 68-point facial landmark predictor is used for driver pose estimation.

Fig. 10 Tensorflow facial detection points. [1]

The driver monitoring subsystem continuously tracks the driver's head position by detecting key facial landmarks (eyes and nose tip). Using these points, the software computes the vertical displacement between the eye midpoint and nose tip to determine head orientation. Large positive or negative displacements indicate that the driver is looking downward or upward, respectively, while moderate displacement implies forward attention. This information updates the system's attentive flag, which is transmitted to the ESP32 for logic processing.

Simultaneously, the YOLO detection module analyzes each frame from the front camera approximately once per second to locate and classify traffic lights. When a detection occurs, the frame is annotated with bounding boxes and labels indicating the recognized signal color. The images are then timestamped and saved locally in a designated directory. If the YOLO classifier identifies a

green light, a greenLight flag is set; otherwise, it is cleared.

Fig. 11 YOLO green light detected annotations.

The software encodes the system's combined state—traffic light color and driver attentiveness—into a single-byte message that is transmitted to the ESP32 through the serial port. For instance, 0x11 represents a "green light, attentive driver," whereas 0x10 indicates "green light, distracted driver." This efficient communication protocol allows the ESP32 to make rapid, context-aware safety decisions without duplicating computation on the Pi.

```
# --- State signaling ---
if greenLight and attentive:
    bit_to_send = 0x11
elif not greenLight and attentive:
    bit_to_send = 0x01
elif greenLight and not attentive:
    bit_to_send = 0x10
else:
    bit_to_send = 0x00
```

Fig. 12 Pi sending bytes based off of CV algorithms.

In addition to sending data, the Raspberry Pi also listens for UART commands from the ESP32. When a capture command (0x01) is received, the Pi immediately stores current frames from both cameras, preserving visual evidence of distraction events or environmental conditions. The program runs continuously, displaying real-time camera feeds for monitoring and terminating safely on user command.

```
# --- Listen for ESP command ---
byte_in = receive_bit()
if byte_in == 1 and ret1 and ret2:
    filename1 = f"front_{datetime.now().strftime('%Y%m%d_%H%M%S')}.jpg"
    filename2 = f"rear_{datetime.now().strftime('%Y%m%d_%H%M%S')}.jpg"
    cv2.imwrite(os.path.join(save_path, filename1), frame1)
    cv2.imwrite(os.path.join(save_path, filename2), frame2)
    print(f"📸 ESP Triggered capture: {filename1}, {filename2}")
```

Fig. 12 Pi waiting for a byte from ESP to take the picture with annotations.

Overall, the Raspberry Pi 5 software acts as the visual intelligence layer of Safe Sight, combining deep learning–based detection with facial landmark analysis to assess driver attentiveness and environmental signals. Its modular design supports both real-time feedback and offline review, ensuring synchronized operation with the ESP32 firmware and providing high reliability in dynamic driving scenarios.

C. iOS Application

The Safe Sight iOS application serves as the system's user interface and visualization platform, providing real-time access to driver performance data, stored images, and overall safety metrics. Developed in Swift using the SwiftUI framework, the app connects to the Raspberry Pi through an HTTP-based communication interface to retrieve event data and images collected during each trip.

The application's structure is divided into three primary views, implemented within a tab-based navigation layout. Each view is designed to present a specific category of driver information:

Driver Report View: Displays accumulated data for the previous ten trips, including the number of distractions recorded in each session. This section provides an overview of recent driving behavior and allows users to monitor performance trends over time.

Photo Gallery View: Establishes a network connection to the Raspberry Pi's local HTTP server to request image files from the /photos endpoint. The retrieved filenames are asynchronously decoded and displayed as thumbnails using AsyncImage objects. Each image corresponds to either a traffic-light detection or a driver-distraction event recorded by the embedded system.

```
// Global Vars for photo viewing
@State private var photos: [Photos] = []
@State private var piAddress = "http://10.42.0.1:8080"
```

Fig. 13 Photos stored in an array that is filled once the IP is accessed.

Profile View: Summarizes key user metrics, such as total trips, total distractions, calculated safety score, and application version number. The safety score is computed dynamically using a ratio of trip count to distraction count, providing an intuitive quantitative measure of overall attentiveness.

To maintain smooth interaction and fast response, the application performs all network requests asynchronously using Swift Concurrency (async/await) and URLSession for RESTful communication. Data is automatically fetched and refreshed when the user navigates to the photo gallery tab or when new images are available.

In addition to displaying stored media, the application performs local data persistence using Swift's @AppStorage property wrapper, allowing user statistics to remain consistent across sessions. This will allow the user to not lose their data once the device is off.

The Engineers

Matthew Carvajal is a 22 year-old graduating Computer Engineering student who has a passion for iOS development and sales. He is currently looking to land a job as a Sales Engineer or as an iOS Application Engineer.

Albert Lougedo is a 25 year old graduating in Electrical Engineering. His prior work experience is centered around RFIC design and shares a passion for both medical technology and bridging technical concepts to non-technical audiences.

Mahyar Mahtabfar is a 22 year-old who is graduating with a degree in Electric Engineering. Passionate in embedded systems and Firmware development. He has prior work experience in building automation.

References

[1] Foodex Saudi Expo, "Face detection TensorFlow tutorial image recognition," Accessed: Oct. 27, 2025. [Online]. https://www.foodexsaudiexpo.com/?m=111311351101500&mod=f550e0ba&uri=pin.php%3Fid%3D3725959-38%26name%3Dface+detection+tensorflow+tutorial+image+recognition

[2] DF robot specification for radar application. Accessed: Oct. 27, 2025. [online] https://www.robotshop.com/products/dfrobot-gravity-c4001-24ghz-mmwave-human-presence-detection-sensor-12m-i2c-uart

[3] Radar detection physics and engineering. Accessed: Oct. 27, 2025 [Online] https://wiki.dfrobot.com/SKU_SEN0609_C4001_mmWave_Presence_Sensor_25m

[4] InvenSense TDK, "MPU-6050 6-Axis Gyroscope and Accelerometer Sensor Datasheet," Accessed: Oct. 31, 2025. [Online]. Available: https://invensense.tdk.com/products/motion-tracking/6-axis/mpu-6050/

[5] Texas Instruments, "LM2576 Simple Switcher® Step-Down Voltage Regulator Series Datasheet," Accessed: Oct. 31, 2025. [Online]. Available: https://www.ti.com/lit/ds/symlink/lm2576.pdf

[6] Ultralytics, "YOLOv8 Object Detection Model Documentation and Tutorials," Accessed: Oct. 31, 2025. [Online]. Available: https://docs.ultralytics.com/

[7] Apple Developer, "SwiftUI Documentation and HTTP Networking with URLSession," Accessed: Oct. 31, 2025. [Online]. Available: https://developer.apple.com/xcode/swiftui/

[8] Niu C., and Li K., "Traffic Light Detection and Recognition Method Based on YOLOv5s and AlexNet," *Applied Sciences*, vol. 12, no. 21, 10808, 2022. Accessed: Oct. 31, 2025. [Online]. Available: https://www.mdpi.com/2076-3417/12/21/10808

[9] IEEE Standards Association, "IEEE Standard for Serial Communication – Universal Asynchronous Receiver/Transmitter (UART)," Accessed: Oct. 31, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/7493434

[10] Adafruit Industries, "Piezoelectric Buzzer Usage Guide and Application Notes," Accessed: Oct. 31, 2025. [Online]. Available: https://learn.adafruit.com/using-piezo-buzzer

[11]Davis E. King, "Dlib C++ Library – 68 Point Facial Landmark Predictor," Accessed: Oct. 31, 2025. [Online]. Available: https://dlib.net/