The University of Texas Rio Grande Valley

College of Engineering and Computer Science

Department of Electrical and Computer Engineering

Senior Design II

Report

Project: AI Autonomous Security Mobile Robot (A-ASMR)

Team member 1: Rolando Garza **Major:** BSEE
Team member 2: Yahir Jasso **Major:** BSEE
Team member 3: Eduardo Salinas **Major:** BSCE

April 2025

Edinburg, TX (USA)

CERTIFICATE

This is to certify that, Rolando Garza, Yahir Jasso, Eduardo Salinas, UID number: 20513384, 20581691, 20520279, have been working under my supervision on the project titled AI Autonomous Security Mobile Robot towards the completion of their Senior Design project. The work satisfies the requirements of the Senior Design I/II (ELEE 4361/CMPE 4373, ELEE 4362/CMPE 4374) course.

Advisor: Dr Kumar Signature: SK Date: 5-9-25

Course Instructor: _____ Signature: _____ Date: _____

Table of Contents

List of Figures

List of Tables

Technical Abstract

The AI Autonomous Security Mobile Robot (A-ASMR) is designed for passive autonomous night-time surveillance. It uses LiDAR for real-time environmental mapping and autonomous navigation, ensuring effective coverage and obstacle avoidance. A webcam integrated with YOLOv8 enables object detection, while facial recognition is used exclusively for authentication. Environmental sensors monitor temperature and CO2 levels to help identify potential safety risks. A web-based interface provides users with live video feeds, facial recognition results, and system notifications for real-time monitoring. The software is developed using Python and C++ within the ROS 2 Humble framework, with a modular design supporting scalability and maintainability. AWS is used for cloud connectivity, allowing for secure data access and remote operation. Development follows Agile methodologies to support continuous iteration and rapid deployment of new features. A-ASMR provides a reliable and flexible solution for enhancing overnight security through continuous monitoring and autonomous operation.

Non-technical Abstract

A recent survey by BOS Security, Inc. in April 2024 found that nearly 30 percent of night shift security guards reported falling asleep while on duty [1]. This highlights a serious vulnerability in traditional security practices, particularly during overnight hours. To address this issue, this project presents an AI-powered autonomous security robot capable of providing uninterrupted nighttime surveillance. Equipped with advanced sensors such as LiDAR, the robot can map its surroundings, navigate autonomously, and detect potential intruders using real-time object recognition software. Security alerts, live video feeds, and environmental data including temperature and CO2 levels are transmitted to administrators through a web application. By reducing dependence on human personnel, this solution offers a more reliable, cost-effective, and scalable approach to enhancing safety in overnight security operations.

1. Introduction

The growing demand for enhanced security solutions has led to the development of autonomous security robots, each designed to address specific needs within various environments. Current designs, however, present limitations in adaptability, affordability, or comprehensive functionality. Through a background search, notable examples like Cobalt Robots, Knightscope Robots, and SMP Robots highlight these challenges and inspire innovation.

Cobalt security robots stand out for their autonomous navigation, safety inspections, and facility management capabilities, providing human-like oversight through machine learning. These robots integrate advanced self-driving technology with real-time anomaly detection, offering a solution that can adapt to a wide variety of environments, from office spaces to warehouses. Their ability to carry out routine tasks such as automated inspections, access control, and emergency response reduces human workload while increasing operational efficiency. Despite their impressive capabilities, the high subscription costs remain a key barrier to widespread adoption.

Cobalt Robots excel in indoor environments, employing LiDAR and 3D cameras to detect anomalies like unauthorized access, unusual sounds, or open doors. These robots feature advanced mapping, elevator operation, and environmental monitoring, but their high monthly cost of $6,000 limits accessibility. Conversely, Knightscope Robots, at a more affordable $545 per month, primarily serve outdoor spaces with features such as 360-degree cameras, thermal sensors, and license plate recognition. While they offer AI-based suspicious behavior detection and multi-terrain capability, their effectiveness in indoor settings is limited. Similarly, SMP Robots, priced at $1,500 per month, are optimized for outdoor industrial applications, employing

GPS, thermal imaging, and motion sensors. However, their focus on perimeter patrol leaves a gap in applications that require versatility across both indoor and outdoor environments.

Figure 1: Cobalt Security Robot [2016] [2]

Knightscope's K5 robot stands out with its combination of affordable pricing and advanced security features. It is designed for both indoor and outdoor use, with capabilities like 360° ultra-HD cameras, thermal detection, people detection, and license plate recognition, making it well-suited for environments such as corporate campuses, shopping malls, parking structures, and airports. Knightscope robots also integrate mobile physical deterrence, workplace violence protection, and real-time situational awareness, offering both physical and digital monitoring capabilities. While effective for a range of applications, these robots, like others in

the industry, still face challenges related to high operational costs, often driven by subscription services and data storage fees.

Figure 2: Knightscope Security Robot [2023] [3]

The SMP Robotics S5.2 IR Argus is an advanced autonomous security robot designed for outdoor perimeter and fence line surveillance, offering a robust solution for commercial and industrial facilities. Equipped with a dual-spectrum PTZ camera, the Argus can perform 360-degree thermal video surveillance, providing reliable detection of people and vehicles, even in low-light or no-light conditions. The robot's high-resolution visible spectrum camera complements its thermal imaging, enhancing its ability to capture detailed images for human detection and face recognition. With the ability to automatically patrol pre-programmed routes, the Argus can detect and track individuals at distances of up to 200 meters using thermal imaging, while its six high-resolution cameras ensure continuous 360-degree surveillance of its

immediate surroundings. Additionally, its intelligent video analytics system, powered by deep learning algorithms, minimizes false alarms and ensures precise detection of objects. The Argus is ideal for night-time security operations in areas such as oil, gas, chemical plants, airports, and border patrols. It autonomously avoids obstacles, returns to its charging station when necessary, and transmits real-time video and status updates via 4G or Wi-Fi, integrating seamlessly with VMS systems supporting ONVIF protocol. Designed for rugged environments, the Argus operates effectively on uneven surfaces and extreme weather conditions, making it a valuable asset for enhancing perimeter security.

Table 1: SMP Security Features [5]

Features	5.2 ARGUS	5.3 ARGUS	MP S5.2-PTZ-IS PICARD
PTZ IR/IS THERMAL AND IP camera 1080P	✓	✓	
360-degree video surveillance by six (6) 720P panoramic cameras	✓	✓	✓
High sensitivity microphone	✓	✓	✓
Human and face detection on PTZ and panoramic cameras	✓	✓	✓
Two (2) embedded computers T9	✓	✓	✓
Edge recording H.264	✓	✓	✓
ONVIF	✓	✓	✓
Sounds notification	✓	✓	✓
IP intercom	✓	✓	✓
Rear obstacle avoidance visual system	✓	✓	✓
Laser Road lighting	✓	✓	✓
Alarm siren	✓	✓	✓
Horn loudspeaker	✓	✓	✓
Front and rear top stereo cameras with obstacle avoidance visual system		✓	
Distant object motion detector		✓	

Figure 3: SMP Security Robot [2020] [4]

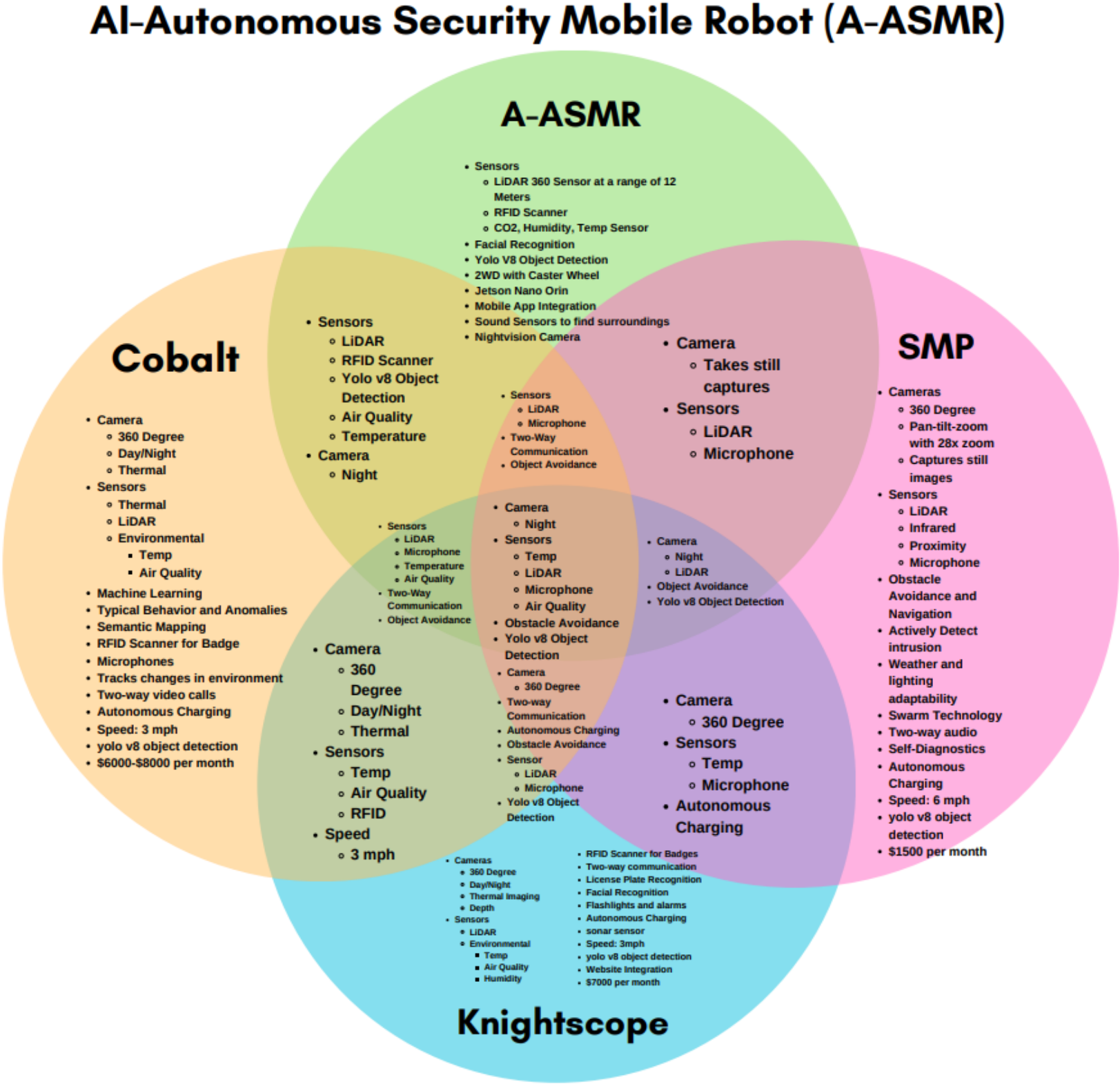

Figure 4: Security Robot Comparisons

The above Venn diagram shows a comprehensive comparison between four autonomous security robots: A-ASMR, Cobalt, SMP, and Knightscope. Our A-ASMR model shares several core features with these competitors while offering unique capabilities of its own.

Key shared features across all platforms include:

- LIDAR sensors for navigation and obstacle detection
- Camera systems with night vision capabilities
- Two-way communication
- Environmental sensing (temperature, air quality)
- Autonomous charging capabilities
- YOLO v8 object detection

Our A-ASMR distinguishes itself with:

- LIDAR 360 Sensor with a specific 12-meter range
- Integration with a mobile app
- Sound sensors for surrounding awareness
- Jetson Nano Orin processing unit
- A cost-effective 2WD with caster wheel design

In comparison:

- Cobalt offers semantic mapping and charges $6000-8000 per month
- SMP features pan-tilt-zoom cameras with 28x zoom and costs $1500 per month
- Knightscope includes license plate recognition and costs $7000 per month

Each platform has its specialties. Cobalt emphasizes machine learning and behavioral analysis, SMP focuses on advanced camera capabilities and swarm technology, and Knightscope includes features like license plate recognition and website integration.

2. Problem Statement

Current autonomous security robots present key limitations in terms of affordability, adaptability, and system integration. High subscription fees, such as those charged by Cobalt and Knightscope, create barriers to adoption, particularly for small businesses and facilities with limited budgets. Many existing platforms are optimized for specific environments and often lack streamlined interfaces or flexible authentication methods, reducing usability for security personnel and facility managers. These challenges restrict the broader adoption of autonomous security systems in environments that require reliable and continuous indoor monitoring.

This project addresses these limitations by developing an indoor mobile robotic platform designed for autonomous navigation and AI-powered surveillance. The system integrates facial recognition for intruder authentication, environmental sensing, and a web-based interface that provides live video, system alerts, and user access logs. The software is developed using Python and C++ within the ROS 2 Humble framework, and AWS services are used for cloud connectivity. The design prioritizes modularity, scalability, and maintainability to support long-term use in indoor security applications.

3. Marketing Requirements

To accomplish the different set of tasks we have in mind the components must meet a certain criterion.

The mobile robot should:

1. Have a lightweight frame to compensate for the added mass of electronic components.
2. Use motors with a decent RPM rating for effective movement and travel speed.
3. Include a dependable and easy-to-use RFID scanner for access control.
4. Feature a powerful and fast microprocessor unit (MPU) suitable for AI tasks.
5. Use a microcontroller (MCU) that is cost-effective, has sufficient I/O, and is compatible with peripherals like RFID.
6. Include a microphone with low distortion and good audio capture quality.
7. Support ROS (Robot Operating System) communication via a capable MCU.
8. Use a camera with sufficient resolution for both clear video and still image capture.
9. Include LiDAR for obstacle detection in the robot's path.
10. Provide an intuitive and easy-to-use user interface via a web application.
11. Use ultrasonic sensors capable of reading distances from 1 cm up to 2 meters.
12. Include a 360-degree continuous rotation servo for pan functionality.
13. Be tall enough for the camera to consistently detect human faces.
14. Have a charger capable of charging the battery at a constant rate of 29.4V and 0.9A.
15. Be able to operate for extended durations (e.g., overnight) without manual recharging or intervention.
16. Be easy to assemble and maintain with modular components where possible.

3.1 Engineering Requirements

Marketing Requirements	Engineering Requirements	Justification (or validation rationale)
4,10,15	The system should be 100% autonomous with the option for manual control through user-end application.	Enables overnight operation without human intervention with manual override available for safety Enables overnight operation without human intervention with manual override available for safety
4,9	The system should have LiDAR capable of 360-degree object detection with a scanning range of 12 meters.	Supports SLAM-based navigation and obstacle avoidance in all directions.
4,7,13	The system should have a camera capable of consistently and accurately detecting people within 6 ft and their face within 2 ft.	Ensures accurate person detection and facial recognition for security purposes.
4,5,6	The system should have a microphone capable of detecting sound anomalies such as glass breaking (~125 dB) or a door squeaking (~60 dB).	Allows audio-based alerts and directional sound tracking for situational awareness.
3,5	The system should have an RFID scanner that would allow for an individual to be identified.	Adds a secondary authentication layer for access control.
1,2,5,15	The system should be able to travel at about 10mph while being power efficient to last the entire night.	Provides speed and energy performance suited for both rapid movement and long-term surveillance.
1,9,11	The system should be able to stop and detect when the robot is close to bumping into its surroundings.	Prevents collisions and enables safe navigation, especially in tight environments.
12	The camera servo should allow full 360-degree panning for visual coverage.	Expands visual field without requiring the robot to reposition.
14	The battery charger should deliver a steady 29.4V at 0.9A.	Supports proper lithium battery charging for optimal lifespan and safety.

16	The robot should feature modular hardware for easier assembly, maintenance and upgrades.	Simplifies repairs and enables future component enhancements.

Table 2: Engineering Requirements

Table 2 outlines the engineering requirements for the major tasks we plan for the robot to perform, along with the justification for these metrics.

3.2 Objective Tree Diagram

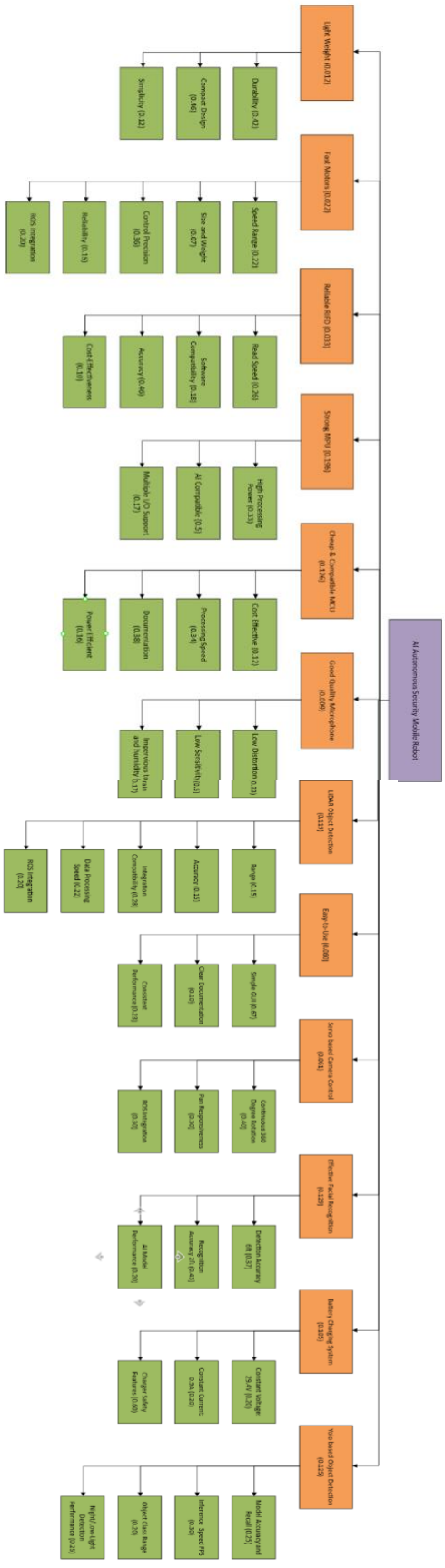

Figure 5: Objective Tree Diagram

Figure 5 presents the various objectives and tasks of the project, outlining the contributions for each objective and their overall weighted importance as determined by the team. The top three categories are: LiDAR Object Detection (11.9%), Strong MPU (19.6%), Cheap/Compatible MCU (12.6%), YOLO based Object Detection (12.5%), and Effective Facial Recognition (12.9%) These were identified as the most essential components and tasks for the project.

3.3 Objective Tree Table

Table 3: Objective Tree Table

	Light weight	Fast Motors	Reliable RFID	Strong MPU	Cheap & Compatible MCU	Good Quality Microphone	LiDAR Object Detection	Easy-to-use Interface	Servo based Camera Control	Effective Facial Recognition	Battery Charging System	Yolo based Object Detection	Weight
Light weight	1	1	1/3	1/5	1/4	1	1/5	1/5	1/6	1/7	1/8	1/8	0.0119029
Fast Motors	1	1	1/3	1/5	1/4	5	1/7	1/5	1/5	1/5	1/7	1/9	0.0220351
Reliable RFID	3	3	1	1/7	1/5	5	1/6	1/6	1/6	1/8	1/8	1/4	0.0334859
Strong MPU	5	5	7	1	7	9	8	6	8	7	8	7	0.1957527
Cheap & Compatible MCU	4	4	5	1/7	1	8	3	7	7	4	3	4	0.125841
Good Quality Microphone	1	1/5	1/5	1/9	1/8	1	1/9	1/4	1/5	1/8	1/7	1/8	0.0090098
LiDAR Object Detection	5	7	6	1/8	1/3	9	1	4	5	3	4	3	0.1191038
Easy-to-use Interface	5	5	6	1/6	1/7	4	1/4	1	2	1/6	1/5	1/6	0.0604646
Servo based Camera Control	6	5	6	1/8	1/7	5	1/5	1/2	1	1/8	1/5	1/8	0.0612803
Effective Facial Recognition	7	5	8	1/7	1/4	8	1/3	6	8	1	5	3	0.1298146
Battery Charging System	8	7	8	1/8	1/3	7	1/4	5	5	1/5	1	1/5	0.1056772
Yolo based Object Detection	8	9	4	1/7	1/4	8	1/3	6	8	1/3	5	1	0.1256319

Table 3 illustrates how the percentages from the tree diagram were calculated. The objective tree table systematically compares each task, enabling proper weighting of options. While each category is independently important, comparing them against one another reveals that some warrant greater weight. This process helped us identify the areas where we wanted to allocate more time and budget.

4. Design Objectives and Realistic Design Constraints

4.1 Design Objectives

Objectives of this project must include:

- Have a light-weight frame to account for the additional weight from electronic components.
- The motors should also have a decent rpm rating for speed.
- The RFID scanner must be dependable and easy to use.
- The MPU must be powerful and fast for AI use.
- The MCU must be cheap, have enough input/outputs, and compatible with RFID.
- The microphone should be of a decent quality and low distortion.
- The Camera should have enough quality to show clear video captures and have the ability to take still captures.
- The LiDAR should be able to detect objects in its way.
- Have an easy-to-use interface on mobile app.

4.2 Realistic Design Constraints

4.2.1 Economic

The final system must not cost more than $1,500 especially since the average cost of a mobile robot online is $2,000.

4.2.2 Environmental

The system be able to traverse different terrains or environments without getting stuck or knocked over.

4.2.3 Sustainability

The system should be able to run a full night without human intervention allowing companies to use the system without changing the battery during a session.

4.2.4 Manufacturability

The system should be easy to utilize and use common proprietary parts that can be purchased for quick and easy maintenance.

4.2.5 Health and Safety

The robot should not pose a threat to all humans that are being detected and only serve as a means of detection to ensure safety for users.

4.2.6 Electronic Component Durability

The components that make up the system should have a lifespan of at least 1 fiscal year allowing for minimal maintenance.

4.2.7 Security

The system should not be vulnerable to attacks to ensure no private information is leaked.

4.3 Design Standards

4.3.1 Battery Standards

The battery should allow the system to stay alive for at least 12 hours before having to charge.

The battery type should be a lithium-ion which would offer high-capacity storage for long operational hours.

4.3.2 Self Routing Standards

The system should be able to create its own pathing through the use of the LiDAR sensor and an algorithm utilizing reinforced learning to determine a point of interest to navigate ensuring constant navigation through unknown areas.

4.3.3 App Standards

The app should be simplistic enough for anyone to understand and allow for integration of sensor data and live camera feed.

The app should be able to establish communication with the mobile robot to ensure sensor data, view, and the controlling of the robot is possible.

The app should ensure that no other person can access another user's video stream or connection with the mobile robot.

The app should be unit test per feature to ensure no bugs or vulnerabilities are coded.

The app should be coded in a way that the code is readable, maintainable and efficient.

4.3.4 Authentication Standards

The RFID scanner should be linked with an authorized user's ID that is coded into a badge.

The facial recognition should be able to identify authorized users and unknown perpetrators, taking a still capture regardless of accessories such as caps and glasses.

5. Project Timeline

5.1 Original Gantt Chart

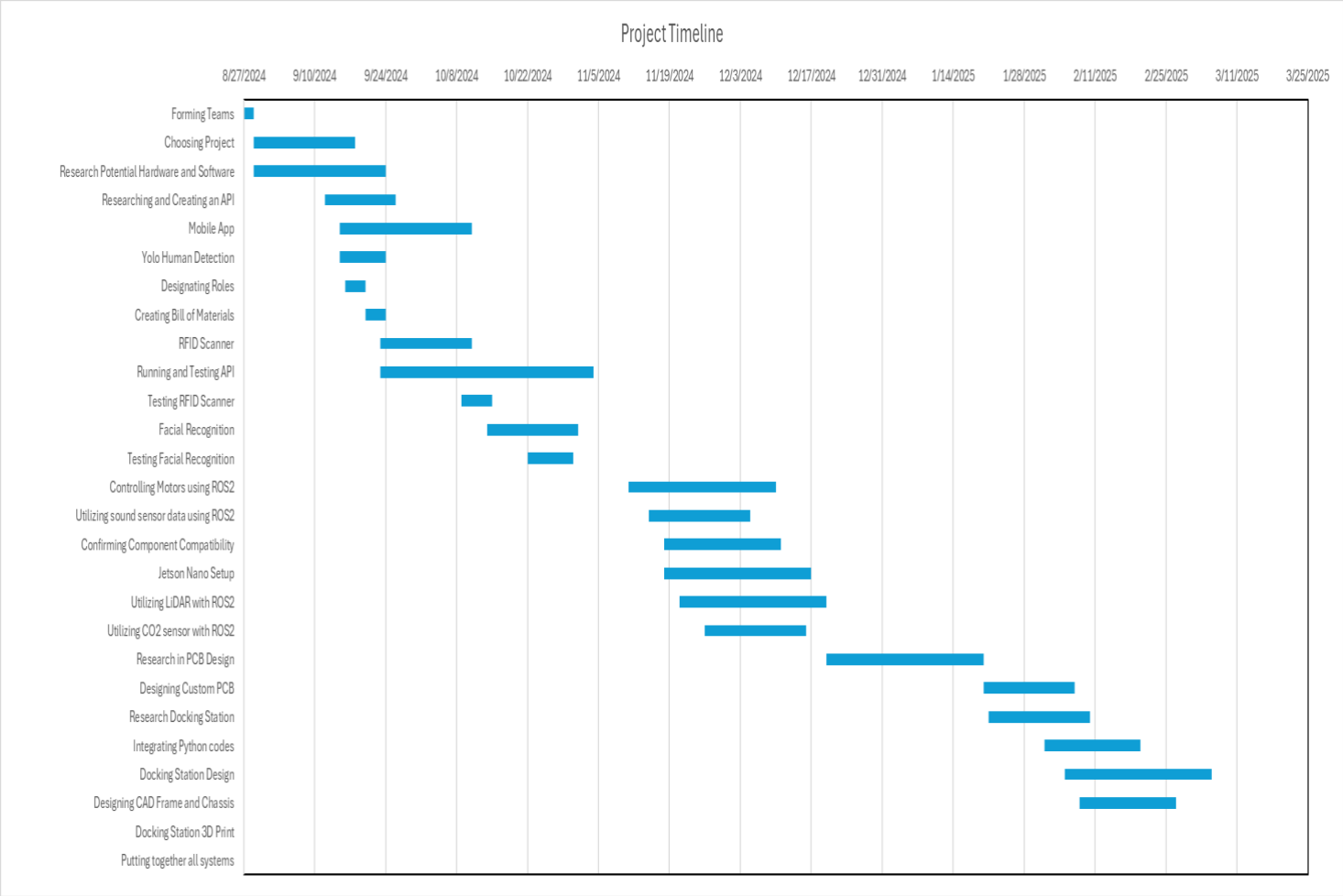

Figure 6: Original Gantt Chart

Figure 6 shows the original Gantt chart created at the beginning of the semester, outlining our initial time estimates for each major task and deliverable. This schedule was based on assumptions regarding task complexity, team availability, and an ideal development timeline. However, as the project progressed, unforeseen challenges such as technical issues, component delays, and iterative design changes required significant adjustments. Revisions to the project timeline were made at the end of the Fall semester and again at the start of the Spring semester. These revisions are discussed in the following sections, along with updated Gantt charts that more accurately reflect the actual development and testing process.

5.2 Revised Fall Gantt Chart

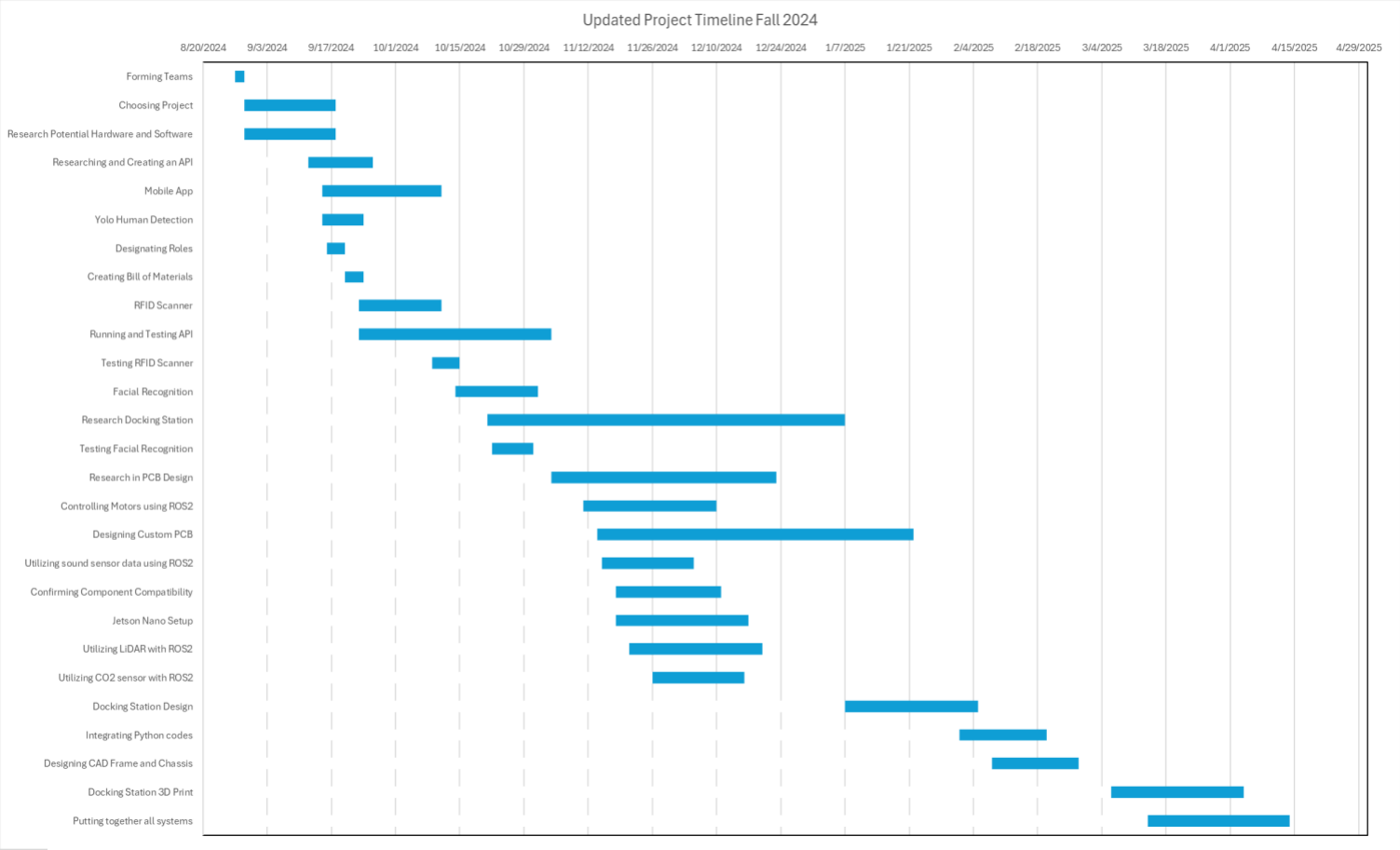

Figure 7: Revised Fall Gantt Chart

Figure 7 shows the updated Gantt chart reflecting adjustments made during the Fall semester. The primary reason for the timeline adjustment was delays in part procurement. All components were ordered simultaneously at the beginning of October following detailed research and planning. However, significant delivery delays for key components such as the motors, LiDAR, and Jetson Orin Nano postponed critical hardware tasks and prevented physical testing of the code. Another major change was the timeline for PCB design. Originally, PCB work was scheduled for the Spring semester, but it was moved up to the Fall semester to shift the focus toward hardware development earlier in the project.

5.3 Revised Spring Gantt Chart

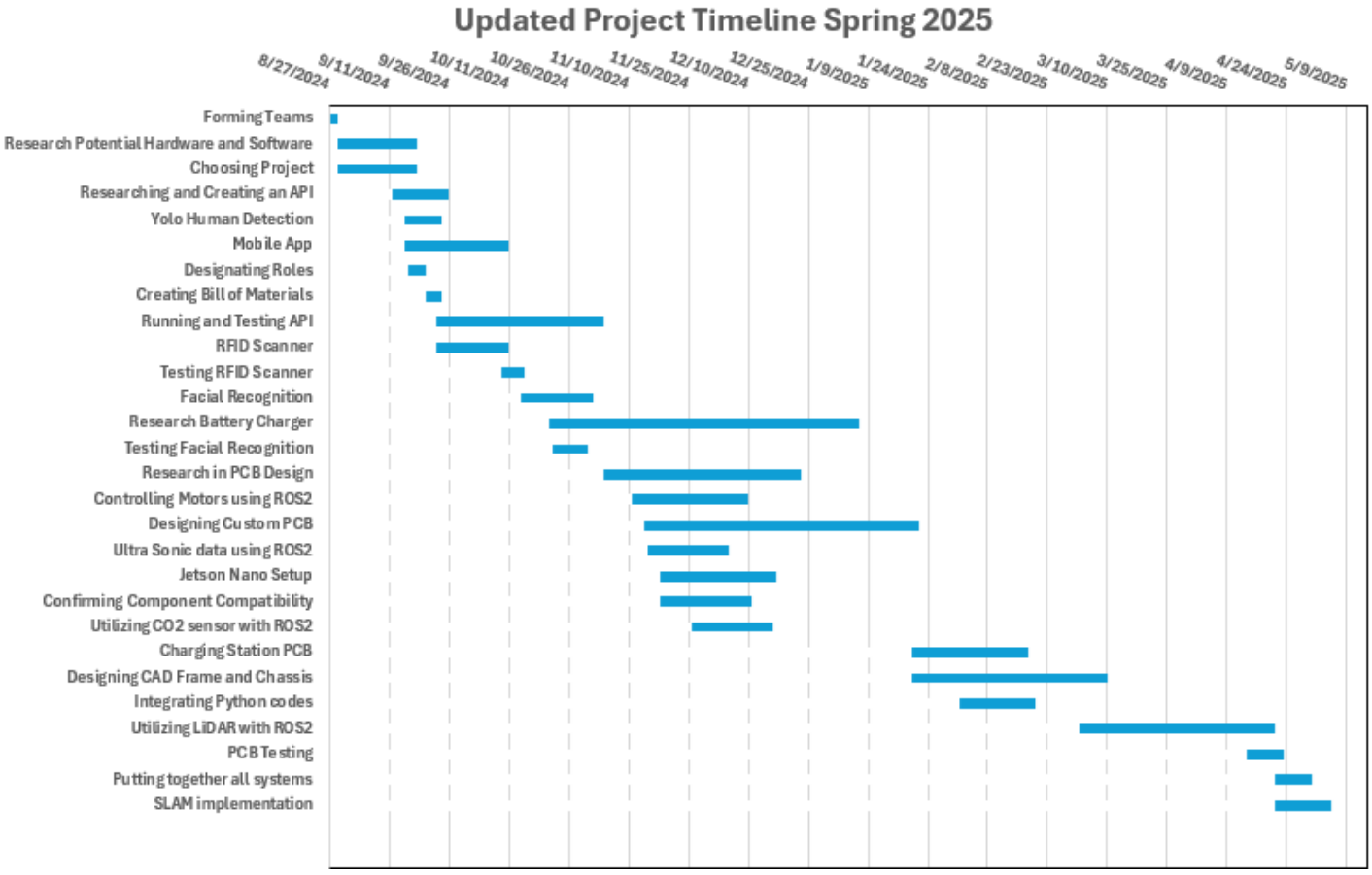

Figure 8: Revised Spring Gantt Chart

Figure 8 shows the updated Gantt chart reflecting adjustments made at the beginning of the Spring semester. At the start of Senior Design II, a meeting was conducted to evaluate the progress made during the previous semester and to refine the project objectives based on realistic development timelines. During this session, the team identified components that would require additional development time, reconsidered tasks that extended beyond the original scope, and discussed new objectives that had emerged during the design process. These discussions led to several adjustments in the project schedule. The most significant change was a shift in focus toward the implementation of the PCB, which required reallocation of time and resources to meet new design and testing demands.

5.4 Task Delegation

Table 4: Task Delegation Table

Tasks	Rolando Garza	Yahir Jasso	Eduardo Salinas
Facial Recognition	☑	☑	
Dashboard			☑
PCB Design	☑	☑	
Motor Control	☑	☑	
UltraSonic Sensors	☑		
Website Implementation			☑
YOLO Human Detection			☑
Fusion 360 Chasis		☑	
ROS2 Communication	☑	☑	☑
ROS2 Simulation	☑	☑	
SLAM Navigation	☑	☑	☑

Table 4 outlines the distribution of tasks for the project. While many responsibilities encompass smaller subcomponents, they have been grouped and summarized under broader categories. These assignments serve as a general framework and do not limit team members from contributing to other areas as needed. For complex or unfamiliar tasks, collaborative efforts were emphasized during both the research and implementation phases. Overall, task assignments were determined primarily based on individual skill sets, with consideration given to each member's interests.

6. Project Design Process

Before beginning development of the main system and its subcomponents, the team created a general schematic to capture the overall architecture of the project. Initial planning included flowcharts to organize system logic and visualize data flow between components. This evolved into a block diagram that provided a high-level overview of the hardware and software

integration. Using this as a foundation, the project was divided into focused subprojects for implementation. The following sections, beginning with Section 6.1, provide a detailed breakdown of each element, including hardware structure, software functionality, navigation systems, and the final PCB design.

6.1 General Schematic of the whole project

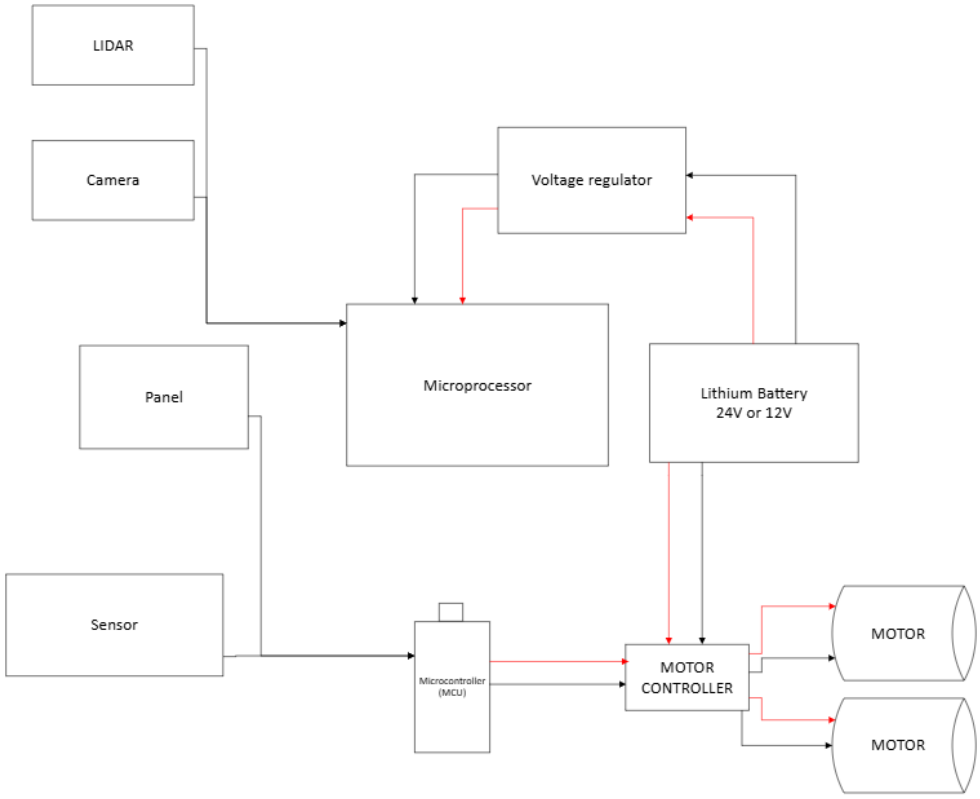

Figure 9: General Project Schematic

The design presents a comprehensive autonomous mobile robot system where each component serves a specific purpose. The LiDAR sensor enables precise distance measurement and mapping capabilities, while the camera provides visual input for navigation and object recognition. The system is powered by a lithium battery (24V or 12V) managed through a voltage regulator, ensuring stable power supply to all components. The microprocessor acts as the brain, processing inputs from sensors and coordinating with a microcontroller (MCU) that

handles motor control. The dual motor setup with a dedicated motor controller suggests a differential drive system, allowing for precise movement and maneuverability. The external battery design indeed enables tether less operation, making the robot fully autonomous and capable of operating in various environments without being restricted to wall power.

6.2 Hardware Block Diagram

Figure 10 presents the updated Hardware Block Diagram for the system. The hardware architecture is divided into two primary control components: the microcontroller unit (MCU), represented by the ESP32, and the microprocessor unit (MPU), represented by the Jetson Orin Nano. A custom-designed PCB forms the foundation of the power system, managing AC input from a transformer, full-wave rectification, and regulated outputs via LM2596T and LM2575T buck converters. This board delivers a 29.4V charging output for the lithium battery, as well as 5V logic-level power, and includes comparator and differential op-amp circuits to ensure safe and stable power regulation.

The ESP32 handles low-level control and interfaces directly with peripherals including ultrasonic sensors, CO_2, temperature, and humidity sensors, as well as the 12V DC motors, motor driver, and a 360-degree servo motor responsible for camera panning. It also manages MicroROS communication, transmitting sensor data and receiving control commands from the Jetson Orin Nano via a USB serial connection.

The Jetson Orin Nano executes high-level processing tasks such as facial recognition, YOLO-based object detection, and environmental mapping. It uses a wide-angle USB-connected external camera for visual processing, and a LiDAR module capable of 360-degree scanning with a 12-meter range for spatial awareness and obstacle detection. Together, the hardware system enables modular, scalable, and autonomous operation of the mobile robot.

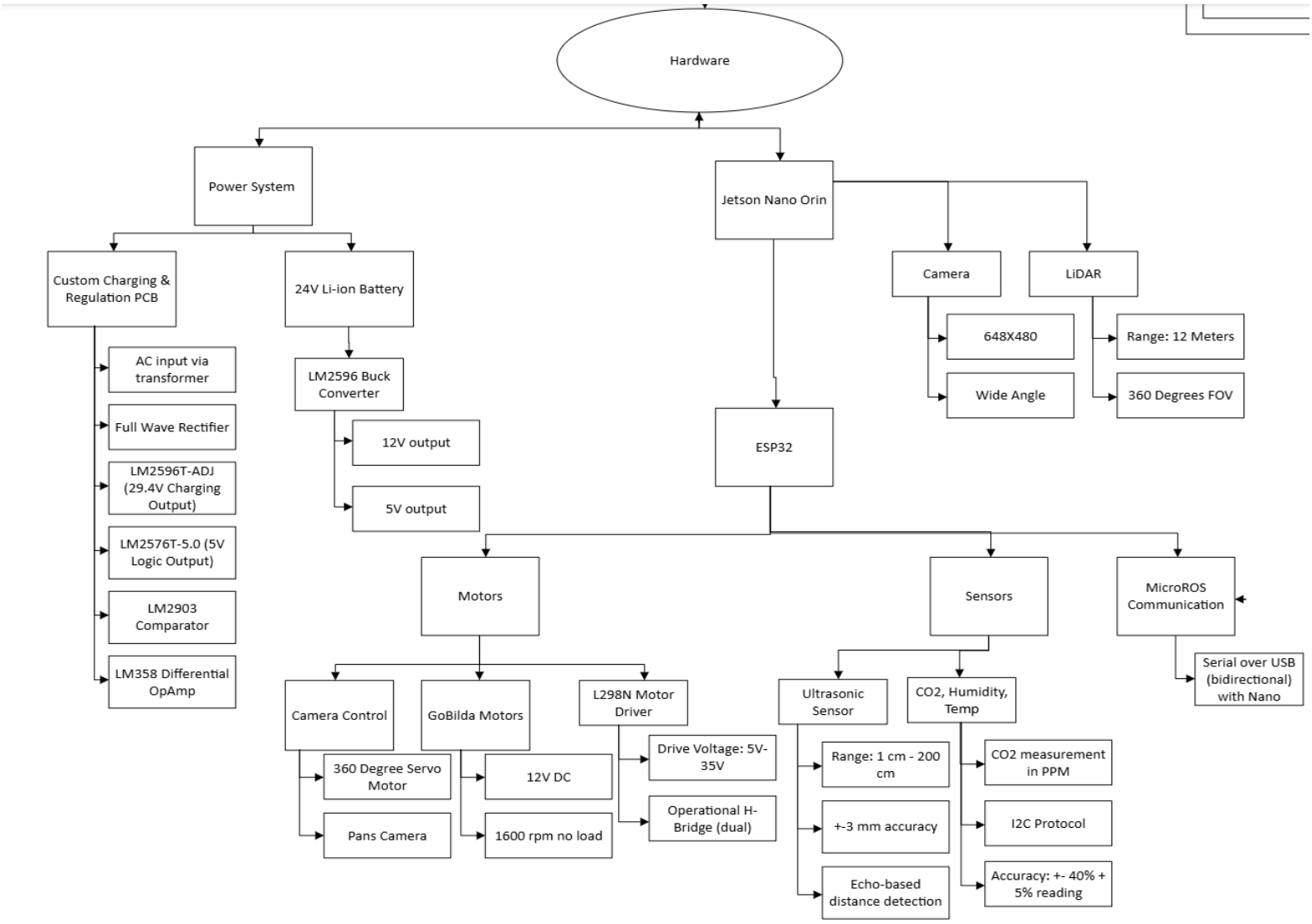

Figure 10: Hardware Block Diagram

6.3 Software Block Diagram

The software is organized into three primary tasks: ROS2, Backend infrastructure, and Web App. For ROS2, the system manages the robot's internal operations, including motor control, sensor data acquisition, and autonomous navigation. Dedicated nodes were implemented for motor control, LiDAR data processing, obstacle detection, and SLAM (Simultaneous Localization and Mapping) to enable autonomous decision making. In addition, MQTT publisher nodes were created to stream real time telemetry data, such as sensor values and robot status, directly to IoT Core. Secure communications with AWS was ensured through the use of device certificates and private keys issued by AWS.

The backend infrastructure includes the usage of AWS services such as Cognito for authentication, S3 for storage, RDS for database management, and IoT Core for real time MQTT communication. FastAPI was used as the main HTTP framework to manage routes and CRUD (Create, Read, Update, Delete) operations. Postman was heavily used during development to test and validate API endpoints, ensuring the data was properly structured and transferred between services.

The web app was built using React and Typescript, with AWS Amplify assisting in authentication flows through Cognito and live streaming handled by AWS Kinesis. The frontend enables real-time visualization and control of the robot by receiving live telemetry and video data. This provided users with a real-time view and control panel accessible from any modern browser.

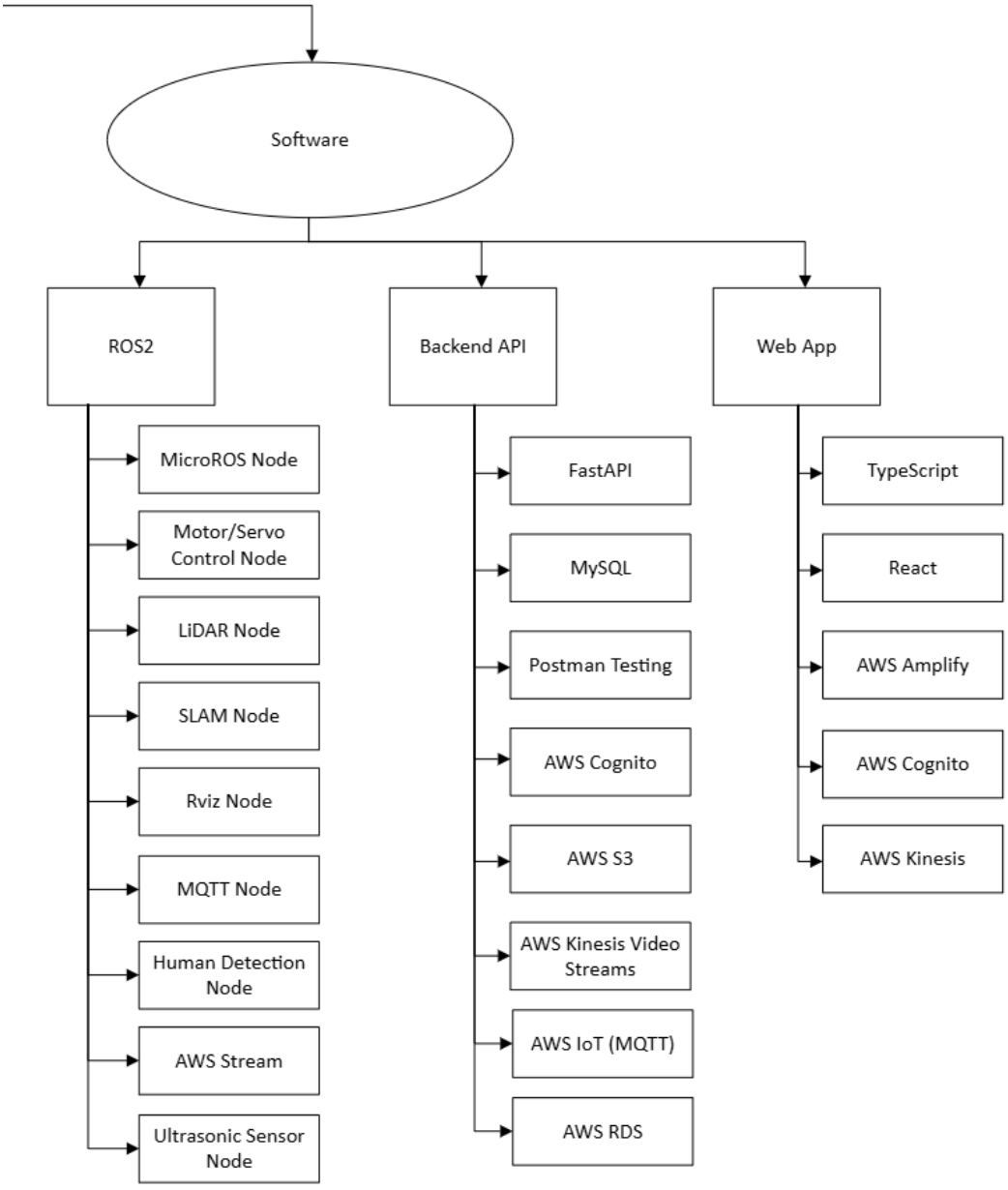

Figure 11: Software Block Diagram

6.3 Facial Recognition

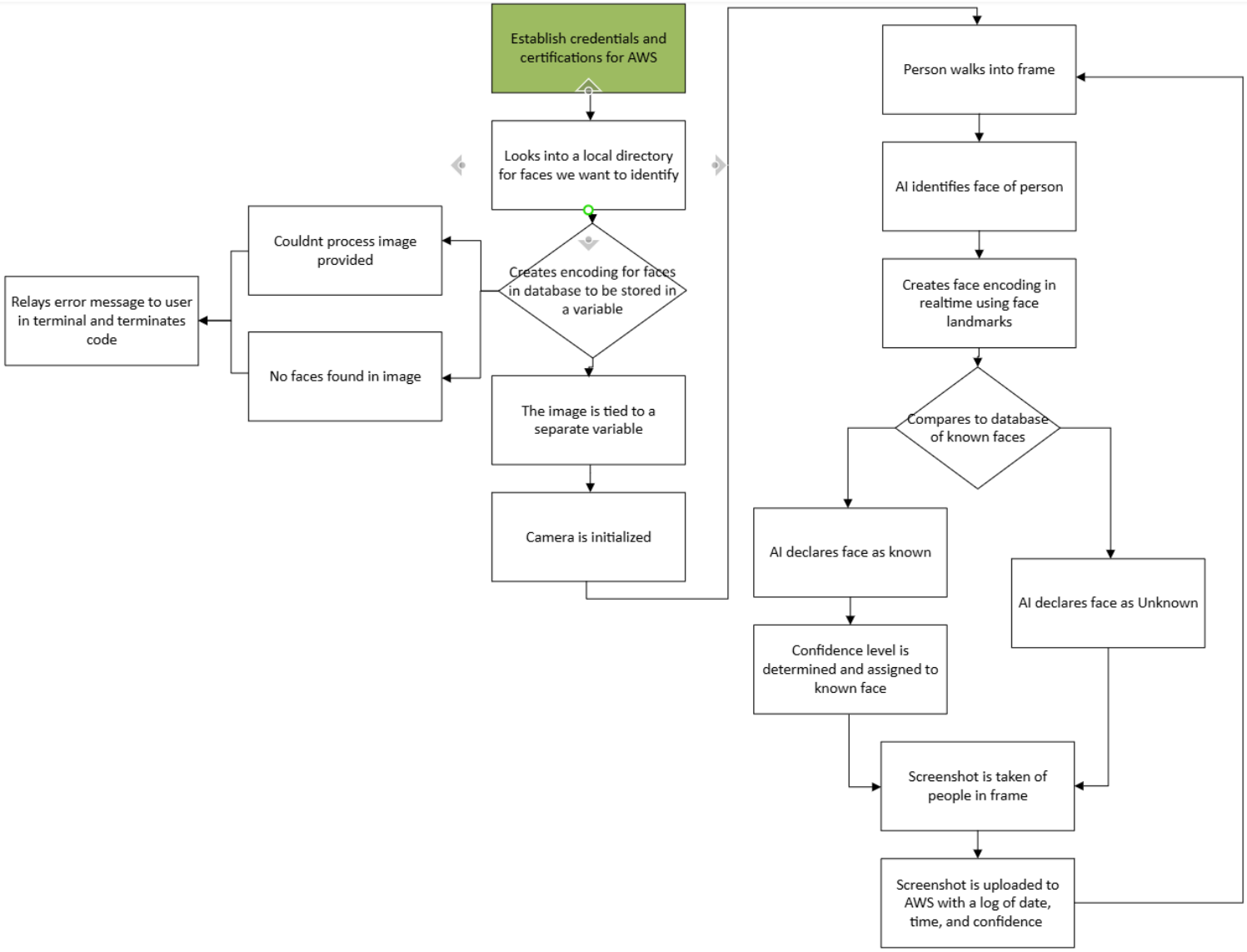

Figure 12: Facial Recognition Flow Chart

Figure 12 illustrates the facial recognition system flow chart, which is divided into two main operational paths: initialization and detection.

The initialization path begins with the system establishing credentials and certifications for AWS access. It then references a local directory containing images of known individuals. For each image, the system generates facial encodings and stores them as variables in memory. If an

image cannot be processed or no faces are detected, the system terminates execution and returns an error message to the terminal. Once all valid images are encoded, the camera is initialized and prepared for real-time detection.

The detection path activates when a person enters the camera frame. The AI system detects the face and generates a real-time encoding using facial landmarks. This encoding is then compared against the stored database of known faces. If a match is found, the system labels the face as "known" and assigns it a confidence level based on the similarity score. If no match is found, the face is labeled as "unknown." In both cases, a screenshot is captured and uploaded to AWS. The uploaded data includes the image along with metadata such as the date, time, and confidence level.

6.4 RFID Scanner

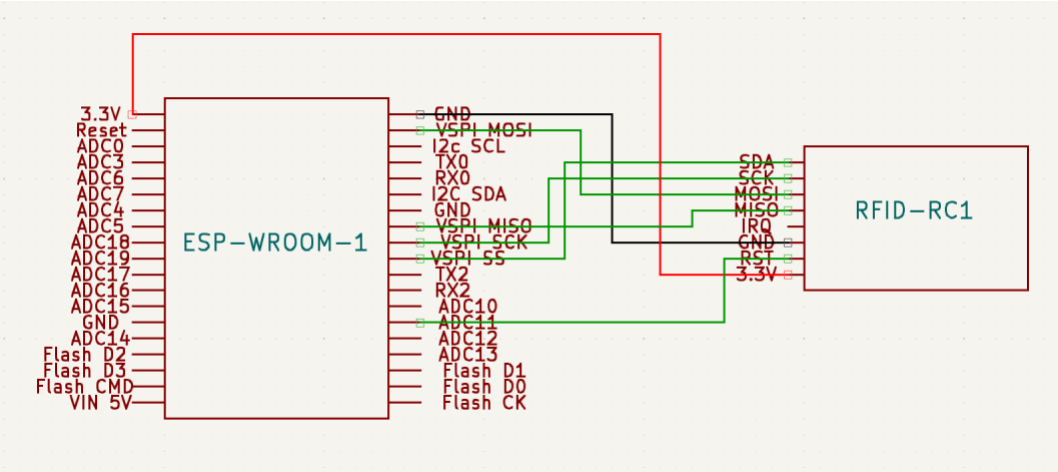

Figure 12: ESP32 and RFID Schematic

Below is the final schematic of the ESP32 connected to the RFID module. The RFID and ESP32 communicate through This project utilizes an ESP32 microcontroller connected to an RFID module to efficiently manage and log user authentication. When powered on, the system establishes a Wi-Fi connection and connects to a Firebase database. If any connection fails, the

system terminates and provides error feedback. Once both connections are successful, the RFID module scans for the presence of a card. When a card is detected, its unique identifier (UID) is

Figure 13: RFID Data in Firebase

read and uploaded to the Firebase database, along with the local time and date. This process creates a reliable log of verified user interactions, which can be utilized for secure access control or attendance tracking. The system ensures real-time data handling and offers a robust solution for RFID-based authentication systems. The figure below shows the upload of the UID in the firebase.

6.5 ROS

ROS 2 (Robot Operating System 2) is the software framework we used to manage the robot's high-level navigation, sensor processing, and control logic. In our implementation, ROS 2 runs on the Jetson Orin Nano and serves as the core of the robot's decision-making system, while micro-ROS extends ROS 2 capabilities to the ESP32 microcontroller for direct hardware interfacing.

Initially, we attempted to transmit sensor data directly over a serial connection to the terminal. However, this approach led to data congestion. The constant stream of sensor readings overwhelmed the communication channel, creating a queue that delayed critical movement commands. To address this, we restructured the system to use micro-ROS, enabling structured,

asynchronous communication between the ESP32 and the Jetson Orin Nano. This change allowed movement commands to be prioritized and ensured that data from the microcontroller could be published efficiently within the ROS 2 ecosystem.

The Jetson Nano handles computationally intensive tasks such as path planning, localization, and sensor fusion. It hosts several ROS 2 nodes that coordinate the robot's behavior. The ESP32, running micro-ROS, subscribes to these commands and publishes selected sensor data, such as ultrasonic distance readings and encoder feedback, over serial using ROS 2 message formats.

This distributed architecture supports responsive motor control, efficient data flow, and modular software design that can be easily expanded to include additional features.

RViz, the 3D visualization tool included with ROS 2, was used to monitor the robot's URDF model and visualize real-time LiDAR data during navigation. This allowed us to confirm that the robot's sensors were functioning correctly and to observe how the robot perceived and responded to its environment as it moved. RViz proved especially helpful during development for debugging and verifying system behavior.

6.5.1 Motor Control Flow Chart

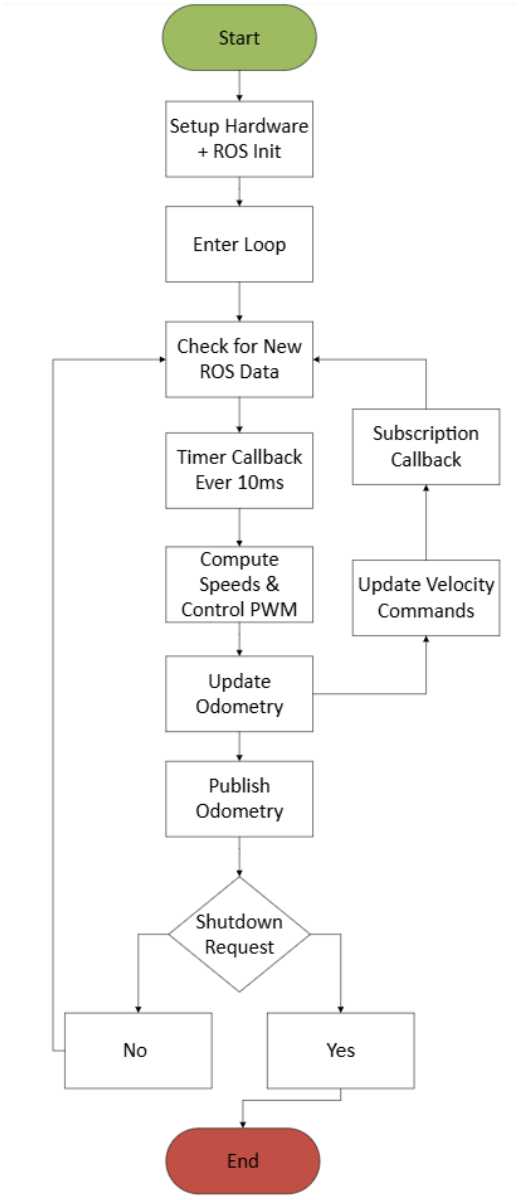

Figure 14: Motor Control Flow Chart

Figure 14 shows the motor control flowchart outlines the logical operation of the robot's drive system. The process begins with hardware setup and ROS initialization, followed by entering the main control loop. Within this loop, the system continuously checks for new ROS data, ensuring updated velocity commands are received through the subscription callback. Every 10

milliseconds, a timer callback triggers computations for wheel speeds and PWM control signals based on the latest velocity inputs. The system then updates the odometry information based on encoder feedback and publishes this odometry data to the ROS network for use by higher-level navigation nodes. To ensure proper flow and avoid uncontrolled infinite looping, the chart incorporates checks for data availability and includes a shutdown condition, allowing the system to exit or reset safely if required. This structured flow ensures the motor controller responds accurately to command inputs while maintaining real-time feedback for autonomous navigation.

6.5.2 Servo Control Flow Chart

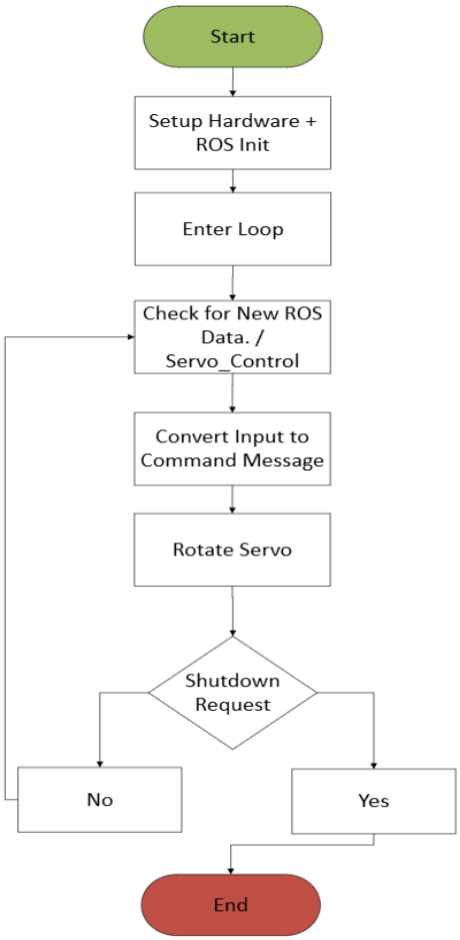

Figure 15: Servo Control Flow Chart

Figure 15 shows the servo control flow chart illustrates the operational logic for managing servo movement using ROS integration. The process begins with hardware setup and ROS initialization, followed by entering the main control loop. Within the loop, the system continuously checks for new ROS data or servo control commands. Incoming data is converted into a command message, which is then used to rotate the servo to the desired position. After executing the command, the system checks for a shutdown request. If a shutdown is requested, the process exits cleanly; otherwise, it loops back to await new commands. This flow ensures responsive servo operation while allowing for safe and controlled system shutdown when necessary.

6.5.3 Ultrasonic Sensor Flow Chart

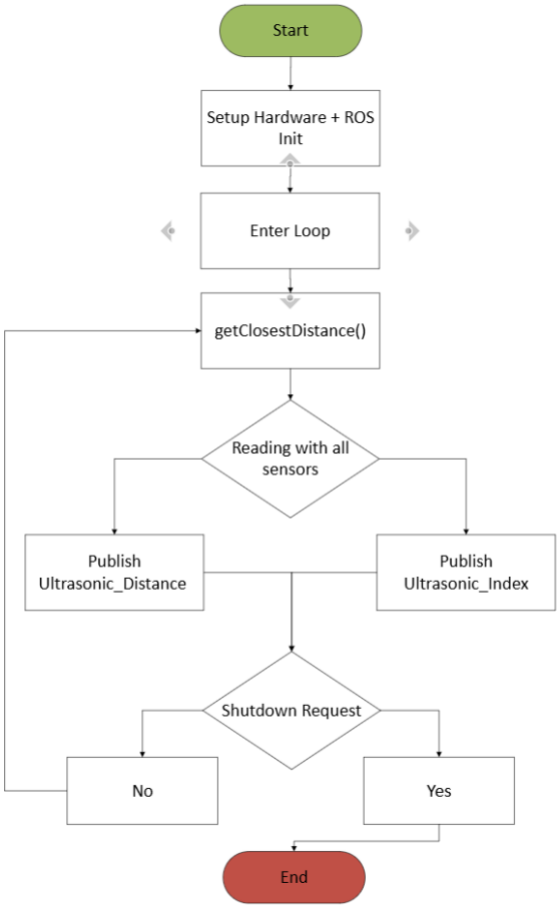

Figure 16: Ultrasonic Sensor Flow Chart

Figure 16 shows the ultrasonic sensor flow chart outlines the process used to obtain and publish distance readings from multiple sensors. The sequence begins with hardware setup and ROS initialization, followed by entering the main control loop. Within the loop, the system continuously calls the getClosestDistance() function, which reads data from all connected ultrasonic sensors. Based on these readings, the system publishes the closest distance to the /Ultrasonic_Distance topic and the corresponding sensor index to the /Ultrasonic_Index topic. After publishing the data, the system checks for a shutdown request. If no shutdown is requested, the process loops back to continue monitoring the environment; if a shutdown is requested, the system exits cleanly. This design ensures continuous and reliable sensor feedback while maintaining the ability to safely terminate operation when needed.

6.6 Web App

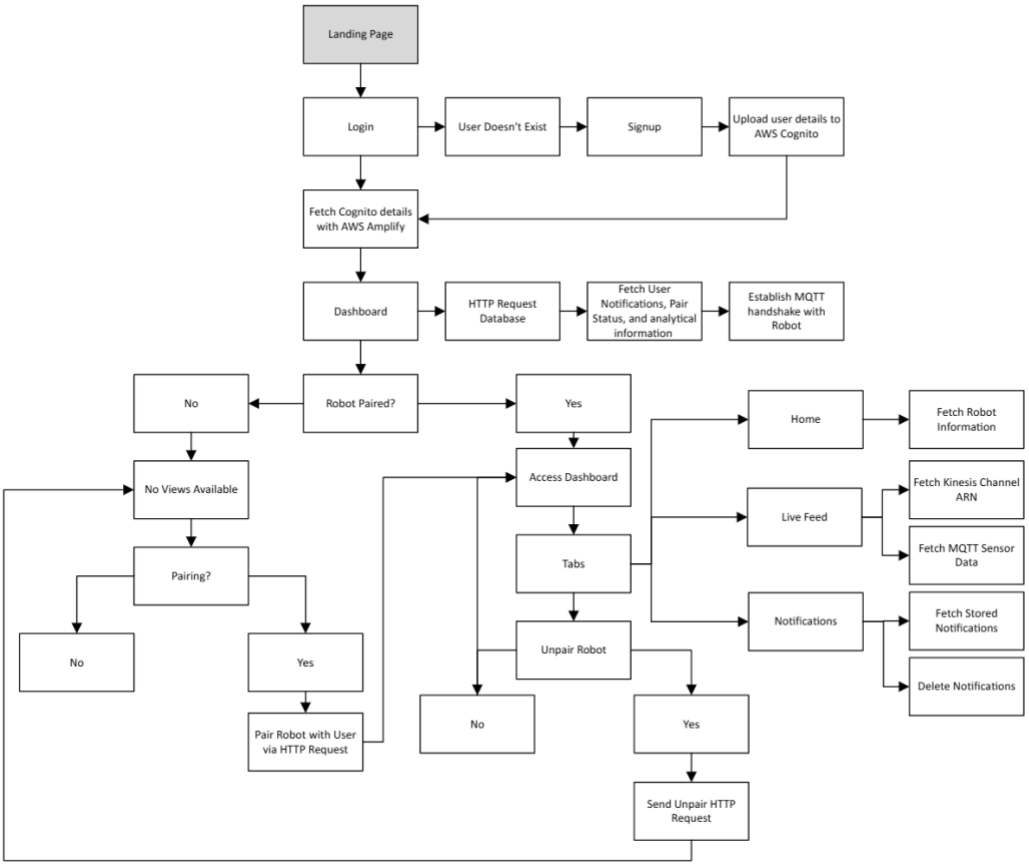

Figure 17:Web App Flow Chart

The flowchart illustrates the full user experience for the Web application, which manages authentication, robot pairing, and real time security monitoring. The system at the landing page, directing users to either login, or if they have no account they can opt for registration, both flows securely fetches/uploads user details to AWS Cognito. Upon successful authentication, the dashboard is accessed, where the initial database queries fetch user notifications, robot pairing status, and establish an MQTT handshake with the robot.

Users are then routed based on their robot pairing status. If the robot is unpaired, all dashboard views are disabled, and the user is prompted to initiate pairing through an HTTP request. Once paired, users gain full access to the dashboard, which is organized into three main tabs: Home, Live Feed, and Notifications. The Home tab displays robot information and basic status, the Live Feed tab streams video from AWS Kinesis, and real time sensor data via MQTT, and the Notifications tab allow users to review and delete stored security alerts. A persistent Unpair Robot button is always available during paired sessions, allowing users to disconnect their robot and return to the restricted view in which robot is unpaired.

The design ensures seamless, secure access to robot functionalities while maintaining backend scalability and per-user resource isolation. Each user is uniquely mapped to a specific robot ID, and dedicated AWS service routes such as pre-signed Kinesis channels, MQTT topics, and S3 storage paths are dynamically generated and securely fetched at runtime. This database driven approach enforces strict ownership, preventing unauthorized access between users, and ensures that notifications, video streams, and robot control data remain logically separated and secure. Overall, the system maintains clear pathways for user actions, real time robot control, and cloud resource management within an optimized web interface.

6.7 Charger PCB Revisions and Development

6.7.1 Charger PCB – Version 1

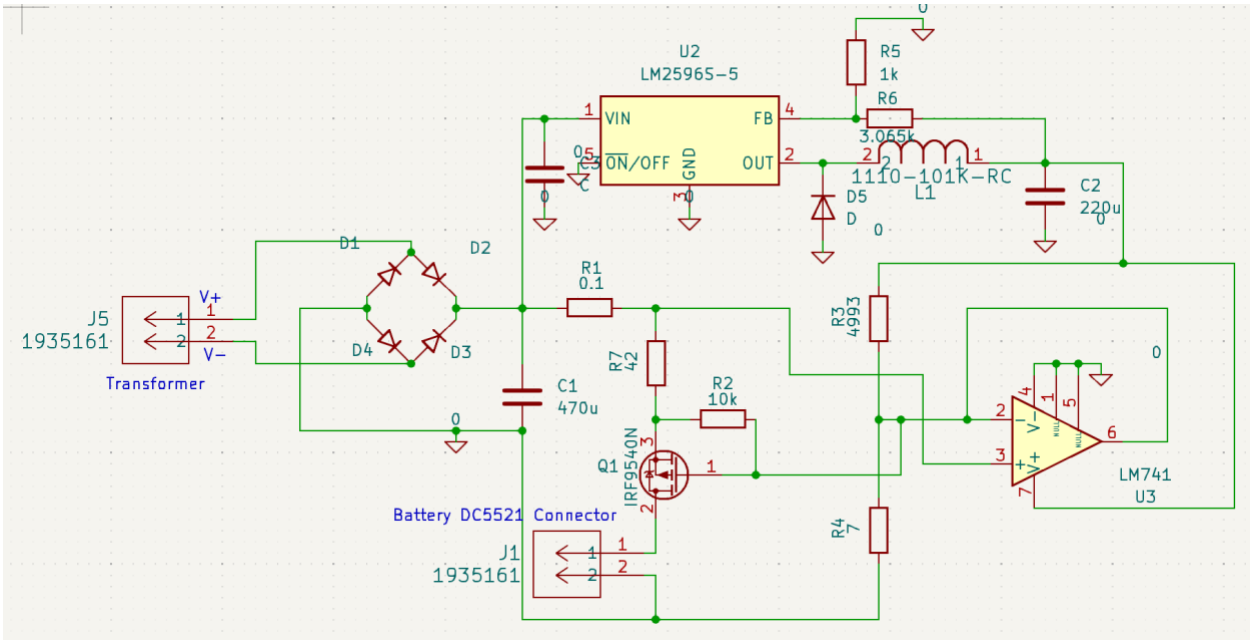

Figure 20: Initial Charger Schematic

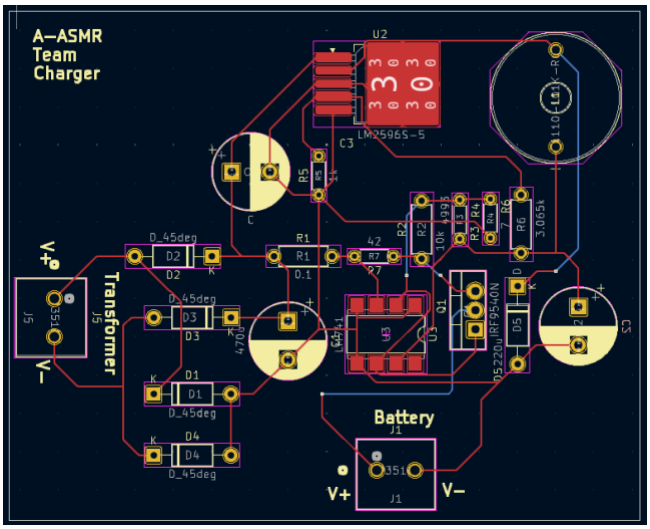

Figure 19: PCB Layout V.1

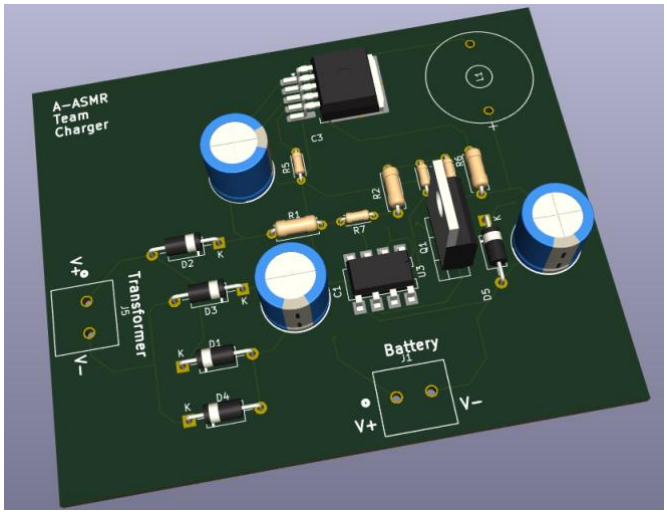

Figure 18: PCB 3D View in KiCad V.1

Figures 18, 19, and 20 show the initial charger design. This version had several mistakes related to wiring and component placement. It used only one buck converter, the LM2596S 5.0,

to output a constant 5 volts. This output was connected to the gate of the PMOS and also used to power the operational amplifier responsible for monitoring and controlling the charging process.

However, this version was scrapped after testing revealed multiple design flaws. One of the main issues was the incorrect positioning of resistors. The 0.1 ohm resistor, intended to act as a current sense resistor, was placed in a location where it could not effectively monitor current, and its resistance value was too low to generate a measurable voltage drop for the op-amp to detect. Additionally, a 42 ohm resistor was included as a load to supposedly limit the current to 0.7 amps, but we later learned that this approach was not valid for current regulation in this context.

Overall, this version served as a basic starting point but required major changes in component placement, circuit feedback, and regulation logic, which were addressed in later iterations.

6.7.2 Charger PCB – Version 2

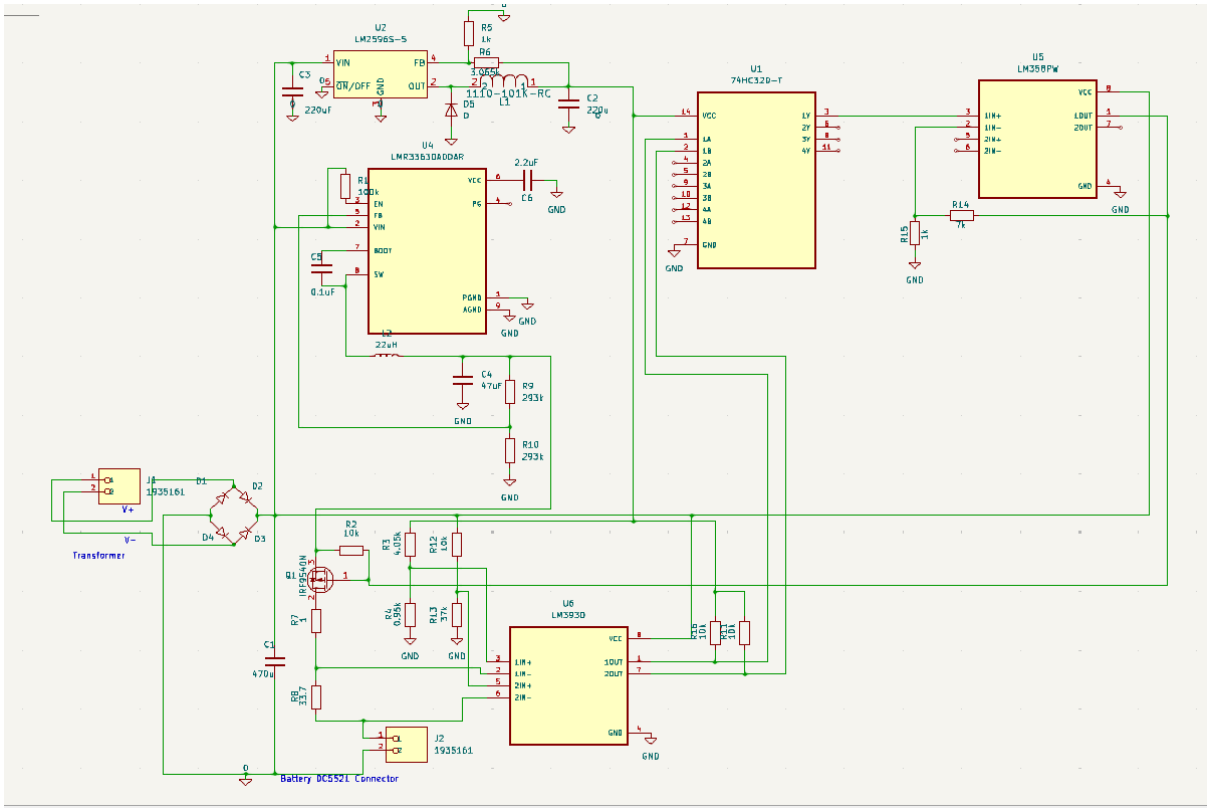

Figure 21: Second Charger Schematic

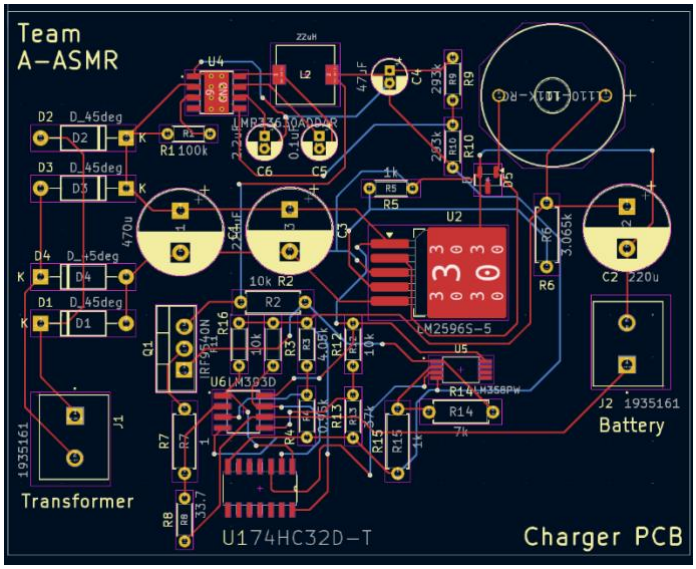

Figure 22: PCB Layout V.2

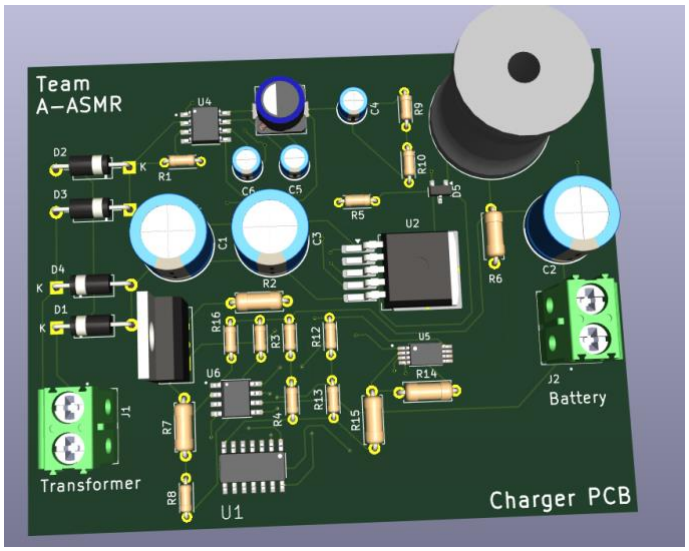

Figure 23: PCB 3D View in KiCad V.2

Figures 21, 22, and 23 show the second iteration of the charger PCB. While this version was a step forward in several areas, it still had some significant issues. One of the main problems was that the resistor load, which we had originally believed would help regulate current, was still incorrectly implemented.

A major improvement in this version was the addition of a second buck converter configured to output 29.4 volts. This output was connected directly to the source pin of the PMOS transistor, which was a correct step toward implementing current control. The current sense resistor was also properly introduced in this version. However, the way we attempted to measure the voltage across it was incorrect. We only captured the voltage after the resistor, rather than measuring the difference across both sides, which made it impossible to accurately determine the voltage drop and therefore the current flowing through it.

Another feature added in this version was the use of comparators to monitor both battery voltage and current. Each comparator was intended to compare its input to a reference voltage. If either the voltage or the current exceeded the set threshold, the corresponding comparator would output a high signal. These outputs were then fed into an OR gate, allowing either condition to trigger a response. The output of the OR gate was then amplified and used to drive the gate of the PMOS high enough to turn it off.

The goal of this logic was to shut off charging when either the current was too high or the battery voltage reached a critical level. However, due to incorrect voltage sensing across the current sense resistor, the system could not function as intended and required further refinement.

6.7.3 Charger PCB – Version 3

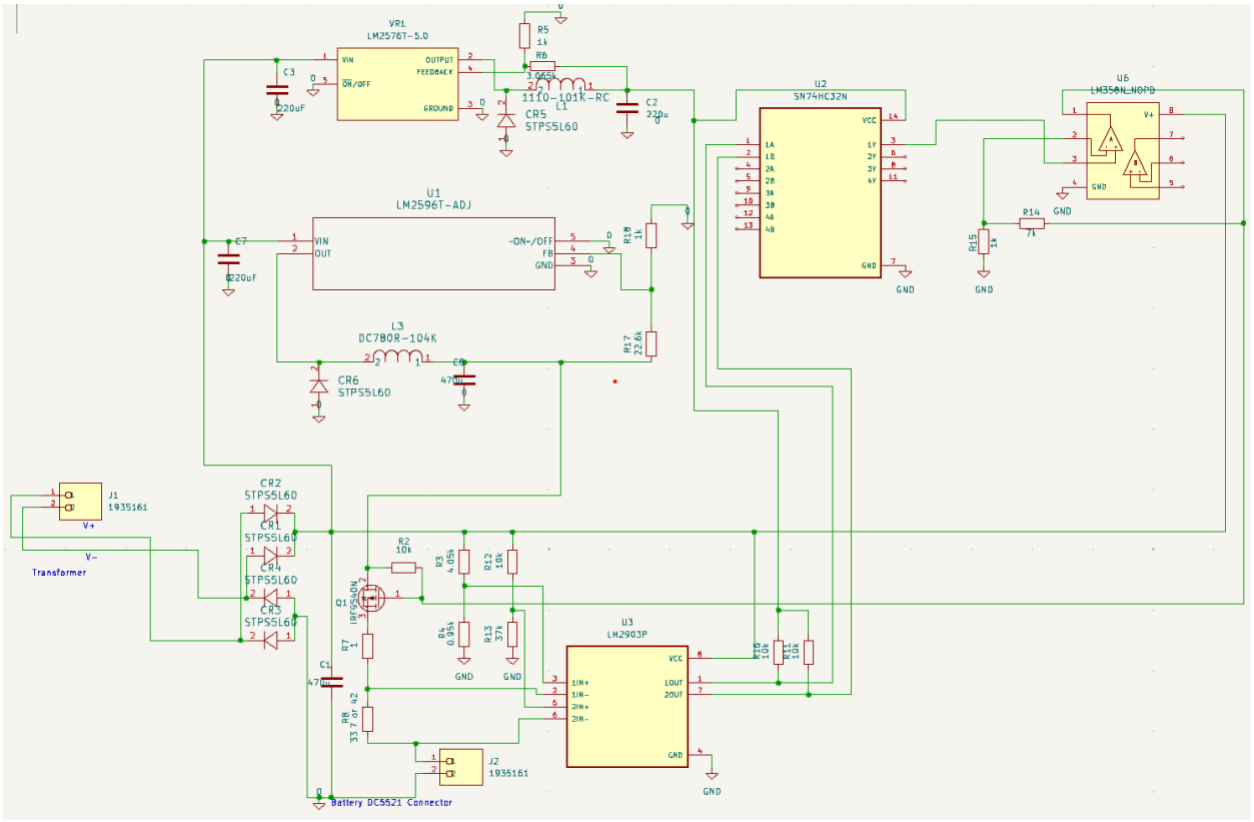

Figure 26: Third Charger Schematic

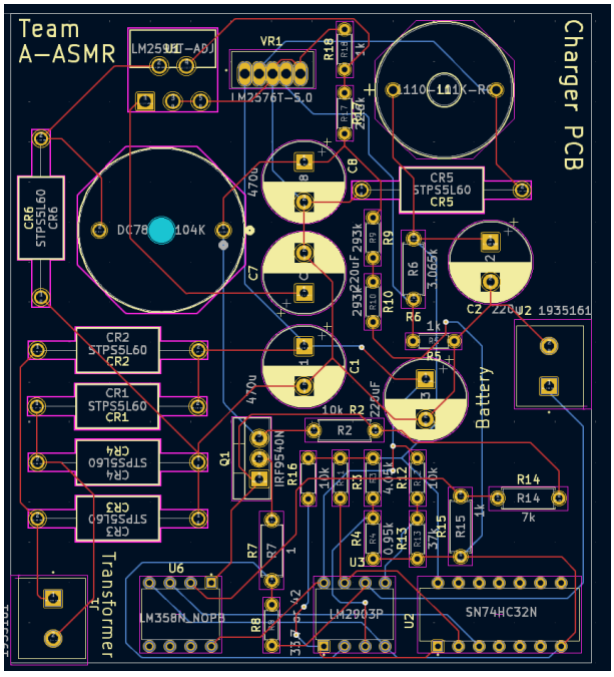

Figure 25: PCB Layou V.3

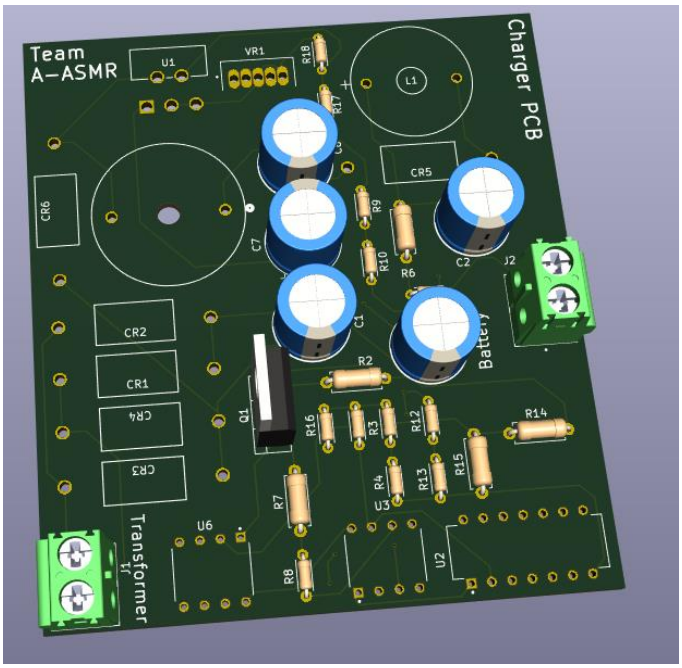

Figure 24: PCB 3D View in KiCad V.3

Figures 24, 25, and 26 show the third version of the charger schematic. The primary difference from the previous iteration was a change in component type. We initially overlooked the fact that many of the components we ordered were surface-mount, which became an issue since we did not have access to a reflow oven or the tools necessary to properly solder surface-mount parts to the PCB. As a result, all surface-mount components were replaced with through-hole equivalents, ensuring the new parts would perform the same function while being easier to assemble and test.

In terms of circuit logic, this version did not introduce any major changes. The overall charging structure and control methodology remained the same as in Version 2. However, this was the version we brought to Dr. Foltz for review and feedback. During that meeting, we discussed the design in detail to confirm whether we were heading in the right direction and properly implementing our control concepts.

The feedback we received during that discussion played a key role in guiding the improvements that led to the final version of the PCB.

7. Final Design and Test Results

Following the development and planning outlined in the Project Design Process, this section presents the finalized features of the robot along with the results of system-level testing. Each subsection highlights a specific component or functionality of the robot, detailing its implementation, performance, and any challenges encountered during integration. Together, these results demonstrate how the design evolved into a working, tested solution.

7.1 Facial Recognition

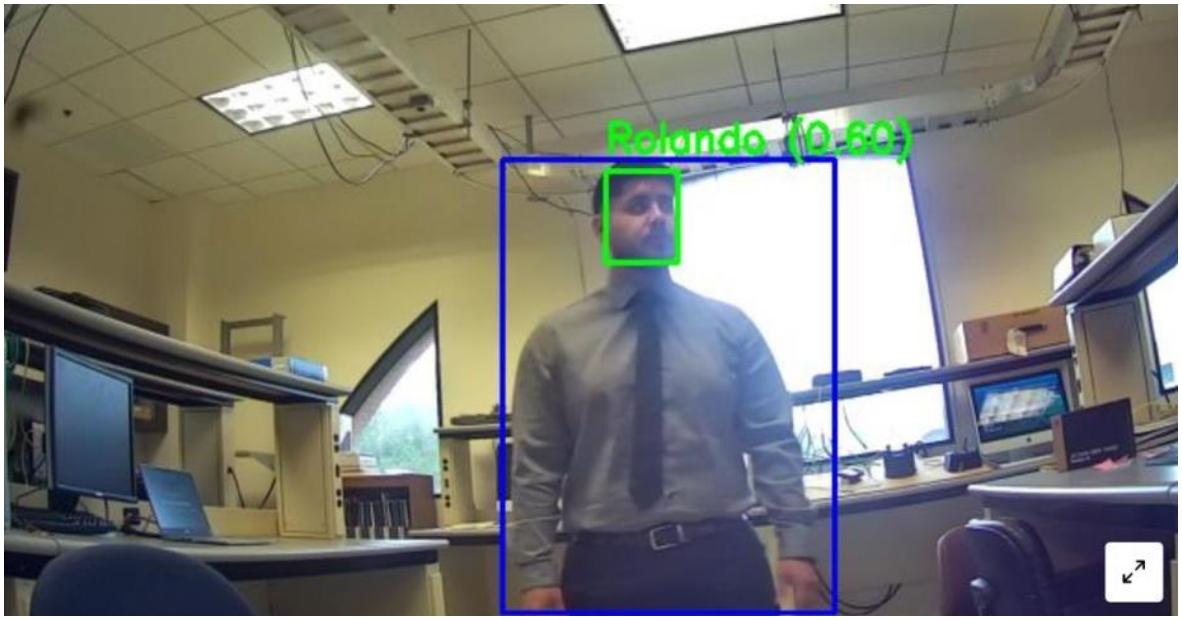

Figure 27: Facial and Human Detection

The facial recognition system uses a real-time pipeline that combines OpenCV's DNN-based face detector with a TensorFlow Lite version of the FaceNet model. The goal of the system is to detect and identify individuals by comparing incoming face embeddings to a library of known identities. The interface of the facial recognition can be seen in Figure 27 above.

7.1.1 Testing and Evaluation

To assess the system's accuracy and robustness, a confusion matrix was created.

Confusion Matrix and Verification Testing:

A verification-based testing approach was used to evaluate the accuracy of the facial recognition system under well-lit conditions. For this test, a known identity was designated as the ground truth. That individual appeared in front of the camera for 30 seconds while performing various head movements such as turning, tilting, and shifting position. This helped assess how well the system could maintain identity recognition despite changes in facial orientation.

Following the ground truth test, several imposters, individuals not included in the known face dataset, were introduced under the same lighting conditions. The system's predictions were recorded and compared to the expected outcomes to generate a confusion matrix. Figure 28 shows the resulting matrix for this test conducted under standard lighting conditions.

These results provided insight into the system's performance in a controlled, well-lit environment, highlighting how facial orientation affects recognition accuracy even when lighting is not a limiting factor.

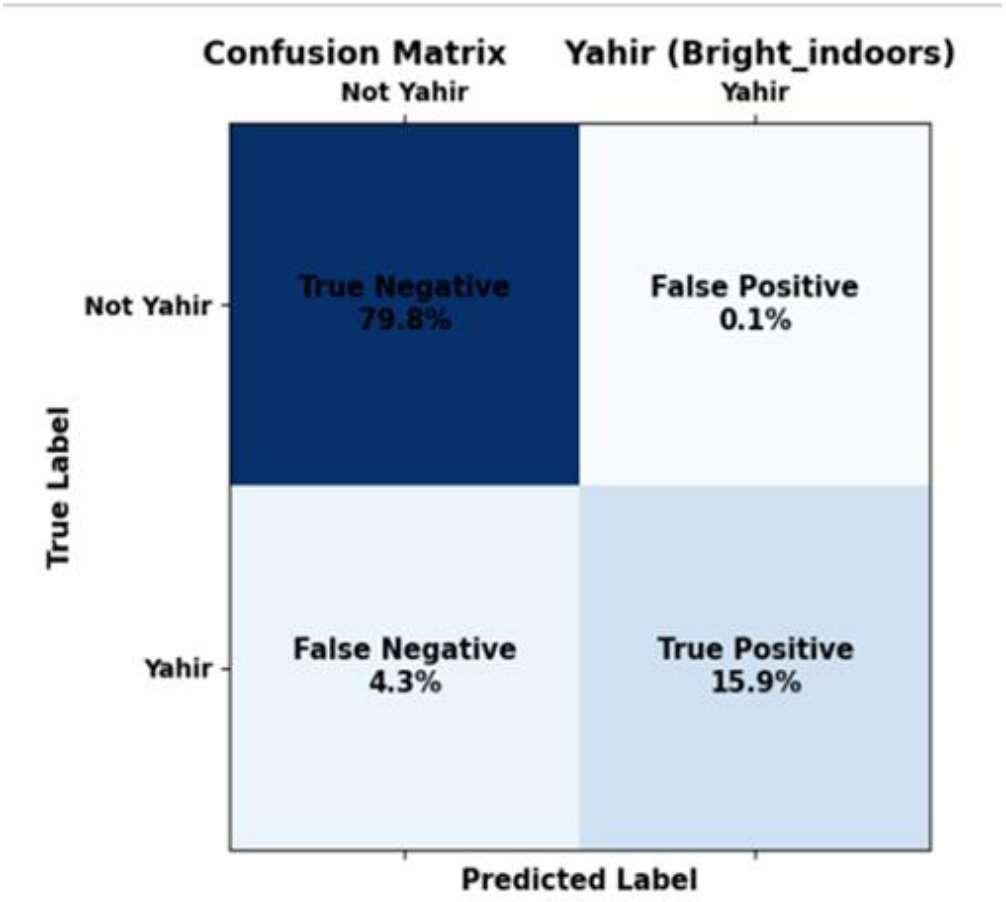

Figure 28: Confusion Matrix Verification

7.1.2 Security Limitation – Photo Spoofing:

It was observed that the system can be tricked using a 2D image displayed on a smartphone screen. This vulnerability is inherent to 2D RGB-based systems and poses a common challenge in lightweight face recognition applications. Addressing this issue would require either additional hardware, such as a depth sensor, or more compute-intensive software-based liveness detection. A potential software-only solution could involve requiring a dynamic facial gesture, such as blinking twice or smiling, before confirming authentication, similar to CAPTCHA-style verification.

7.2 Human Recognition

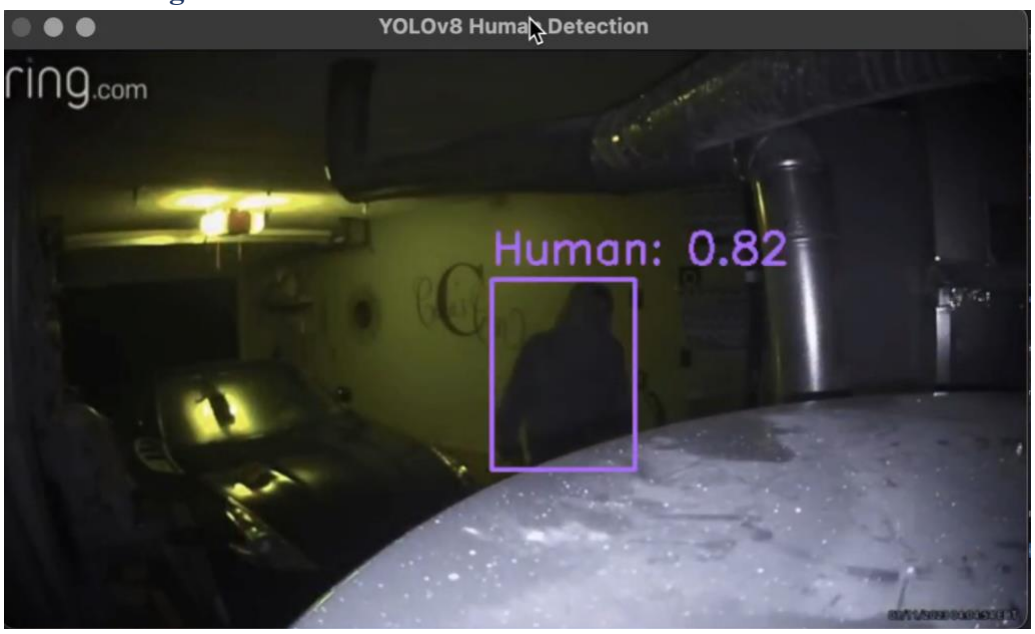

Figure 29: YOLO V8 Example

The human detection system initially leveraged YOLOv8 models on a macOS platform, achieving robust detection performance with local OpenCV integration. However, upon transitioning to the Jetson Nano for onboard processing, initial performance degraded significantly to 3-6 frames per second, primarily due to missing or outdated CUDA and TensorRT drivers. After successfully resolving the hardware acceleration issues by installing the

necessary CUDA stack, the system adopted a new deployment approach focused on optimizing the inference for the embedded environment.

The updated setup involved generating TensorRT optimized YOLOv8 Engines tailored for Jetson Nano, a process that typically requires 10-15 minutes per model in order to run the neural network for object detection. Without these engines a frequent error would arise due to the mismatch float values (float32 vs float16) due to architecture of the Jetson Nano which differs from your typical NVIDIA graphics card. This process ensured that object detection models were correctly quantized and scaled for stable execution.

Following local optimization, the detection system was integrated with a Python based AWS Kinesis Video Stream implementation. This enabled remote video streaming and human detection processing via the cloud, which would work on any network. The system maintains a stable detection-to-upload interval of approximately 1-2 sections, which the language chosen contributes to and the YOLO model version. Despite the additional cloud-based delays, the implementation maintains reliable detection capabilities under varying lighting conditions and enables centralized event logging.

To further improve efficiency, we experimented with replacing YOLOv8 with the lighter YOLOv4-tiny model on the Jetson Nano. This change significantly reduced the computational load and improved inference speed, while still maintaining acceptable detection accuracy for the security use case. Although YOLOv4-tiny has lower precision compared to YOLOv8, the tradeoff between performance and resource savings proved feasible and practical for real-time human detection on embedded hardware.

Overall, the updated architecture enables the robot to perform real time human presence detection while ensuring compatibility with cloud infrastructure, achieving a balance between embedded inference performance and remote accessibility for security monitoring.

7.3 User Application

To complement the autonomous functionality of the security robot, we developed an application designed to provide users with full control and awareness of their assigned robot. The app acts as the primary interface between the user and the system, offering live video streaming, real-time alerts, and remote interaction features. The goal of the application is to ensure that users can seamlessly monitor their environment, respond to detections, and manage robot behavior, all from an interface in a browser.

Initial design efforts were focused on developing a native iOS application using Swift. This early prototype, illustrated in Figma mockups, incorporated all the key features which included live videos streaming, sensor dashboards, alert notifications, and robot control interfaces. However, because Swift is exclusive to Apple platforms, this direction posed limitations on device compatibility, excluding Android and desktop users.

As a result, while the Swift based iOS app remains polished and a testable prototype, we shifted toward a web-based solution to maximize accessibility across devices. Nevertheless, the native iOS version provided valuable insights during development and served as a fully functional testing environment for user experience and validation.

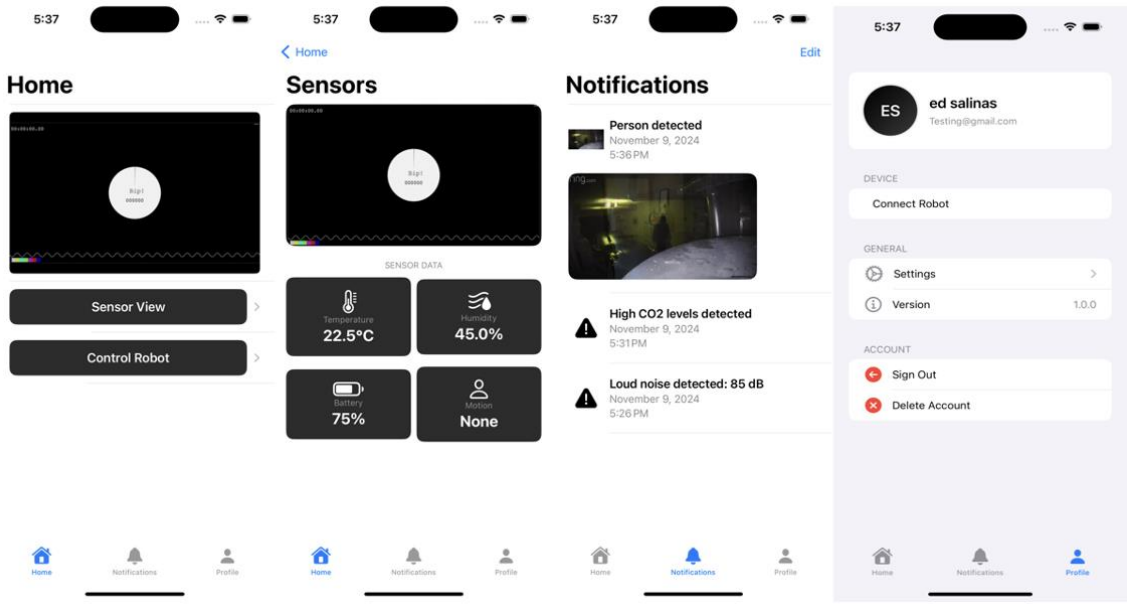

Figure 30: IOS App Interface

Although the iOS application shown in Figure 20, reflects a polished interface, much of its functionality remained in the prototype stage during the initial development. Core features such as the notification system and environmental sensor dashboard were primarily visual mockups and were not connected to live backend services. The only fully functional components at the time were the video stream and robot control, which operated through a local WebSocket connection. This implementation was limited to devices on the same local network, with no cloud infrastructure or remote access integrated.

These working features were successfully demonstrated during our final presentation for Senior Design I. However, the limited accessibility and platform lock-in of the iOS app highlighted the need for a more universal solution. In response, we transitioned to building a web-based application designed for broader compatibility and improved scalability.

The web version of the application was developed using TypeScript, HTML, and CSS and is accessible from any modern browser regardless of the operating system. Our initial goal

with the web application was not only to replicate the functionality of the iOS prototype but also to introduce the project as a product. For this reason, the application includes a polished landing page designed to represent the Robot's capabilities, create a professional first impression, and guide users into the control dashboard. By moving to a web-based stack, we ensure platform independence and easier deployment across devices including Windows, Android, Linux, and macOS.

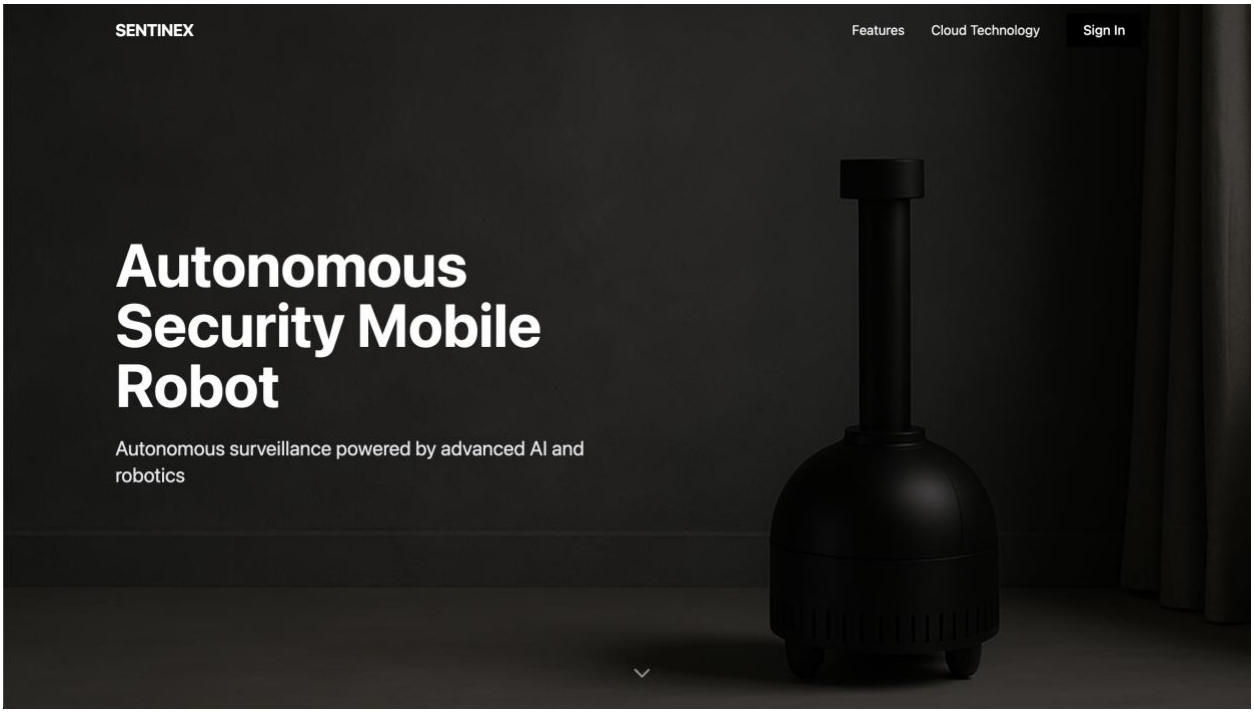

Figure 31: Web Application Landing page

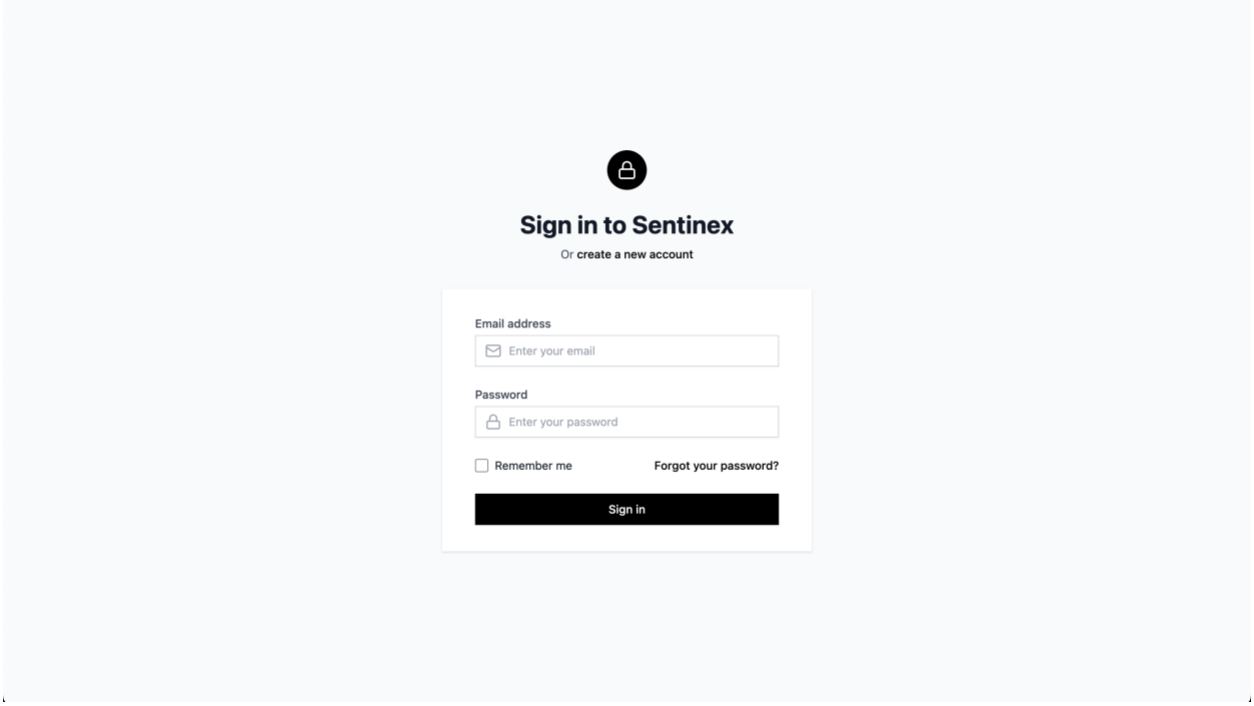

Figure 32: Dashboard login interface

The web app now supports core functionalities including dynamic robot pairing, real-time sensor data retrieval, and notification delivery. The control dashboard integrates live video feeds, environmental metrics, and event alerts into a responsive UI. Additionally, the architecture is designed with cloud-based extensibility in mind, featuring future integration with AWS services for user authentication, media storage, and global device connectivity. These features enable real-time monitoring and control from virtually anywhere, regardless of network or platform.

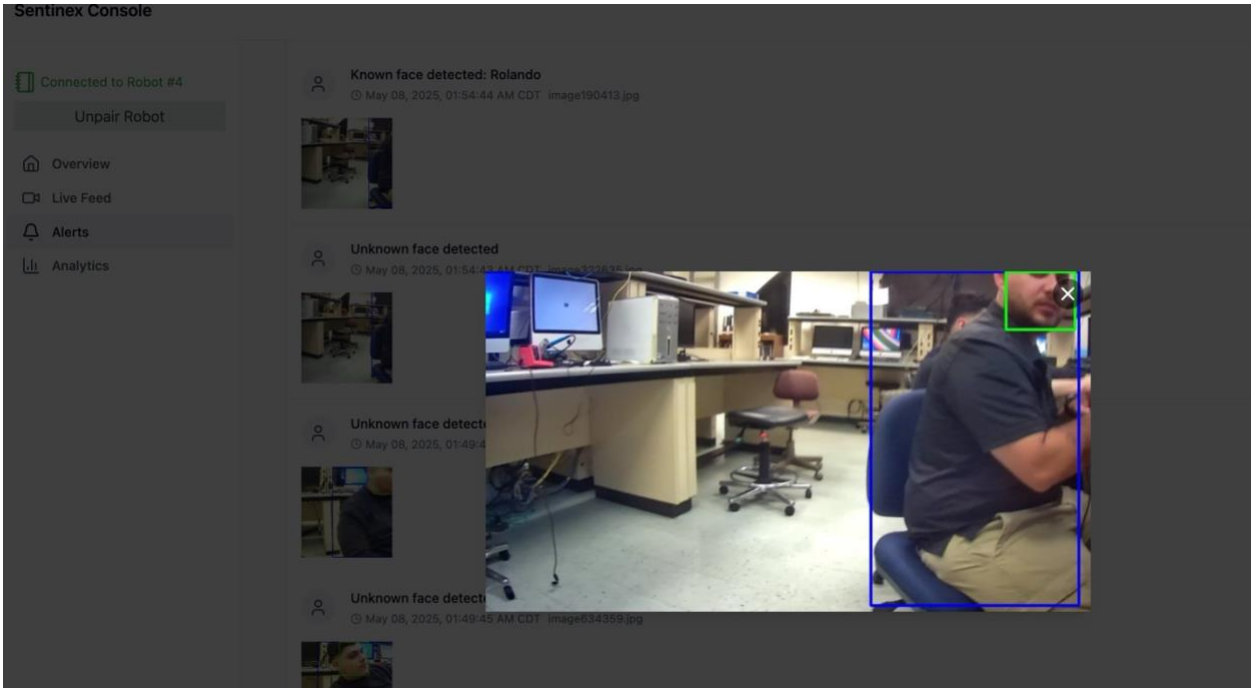

Figure 33: Web dashboard showing notification system

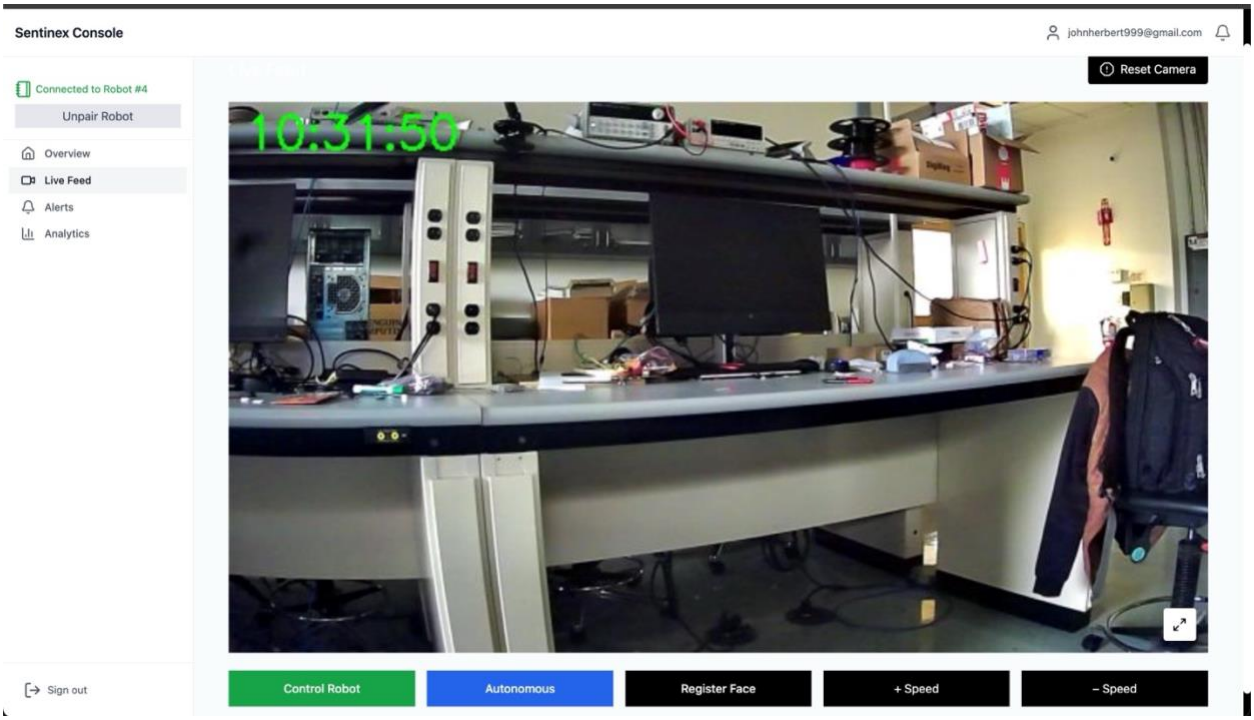

Figure 34: Web dashboard showing video feed

7.4 Cloud-Based Backend Infrastructure

To support the user-facing application and autonomous functionality of the security robot, we developed a cloud-based backend infrastructure designed to meet modern requirements for scalability, reliability, and real-time interaction. This system not only manages robot connectivity and command handling but also ensures secure user authentication, media storage, and sensor data streaming across platforms.

Our infrastructure was built using a combination of Amazons Web Services (AWS) components which are all within the AWS ecosystem. Amazon Cognito handles user registration and authentication through secure token-based access. Authenticated requests are processed via Amazon API Gateway, which routes traffic to multiple AWS Lambda functions responsible for storing media files and retrieving alert metadata. Real-time sensor data and robot events as well

as robot control is transmitted via the MQTT protocol, utilizing AWS IoT Core for publish/subscribe operations between the robot and the cloud.

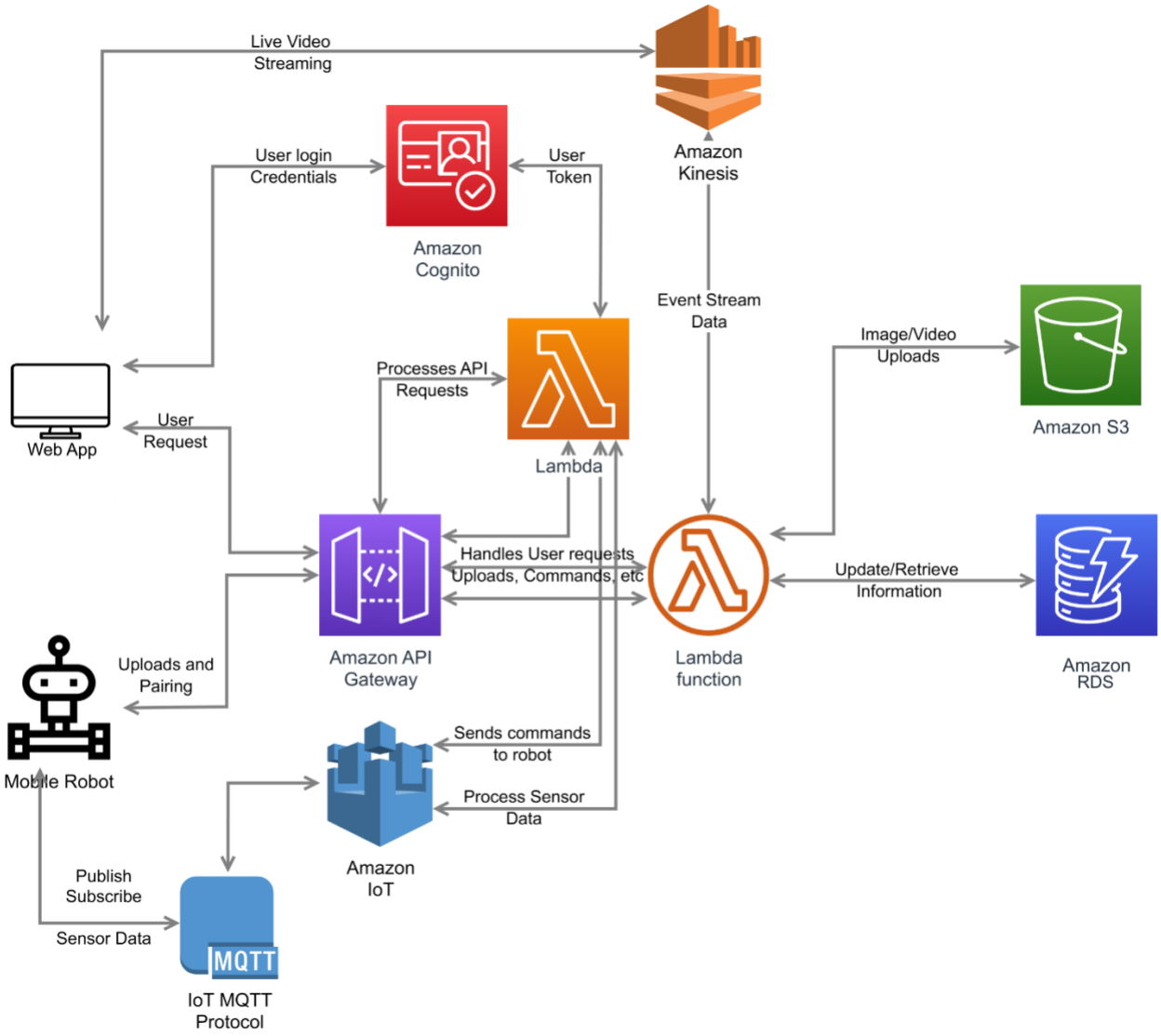

Figure 35: Cloud infrastructure overview with AWS services.

Captured images and video events triggered by the robot are uploaded to Amazon S3 for persistent and scalable storage, while descriptive metadata including timestamps, user association and robot identifiers is stored in a MySQL database hosted in AWS RDS. This separation of actual media and descriptive metadata allows for efficient retrieval and query

optimization. For video streaming, Amazon Kinesis Video Streams is used to deliver live feeds from the robot to the dashboard with minimal latency.

The backend logic is handled by a FastAPI-based service running on an EC2 instance, interfacing with the database, Amazon s3, and Kinesis through secure API routes. Media uploads, robot pairing, alert logging, and notification delivery are all orchestrated via REST endpoints, with authentication managed through AWS Cognito.

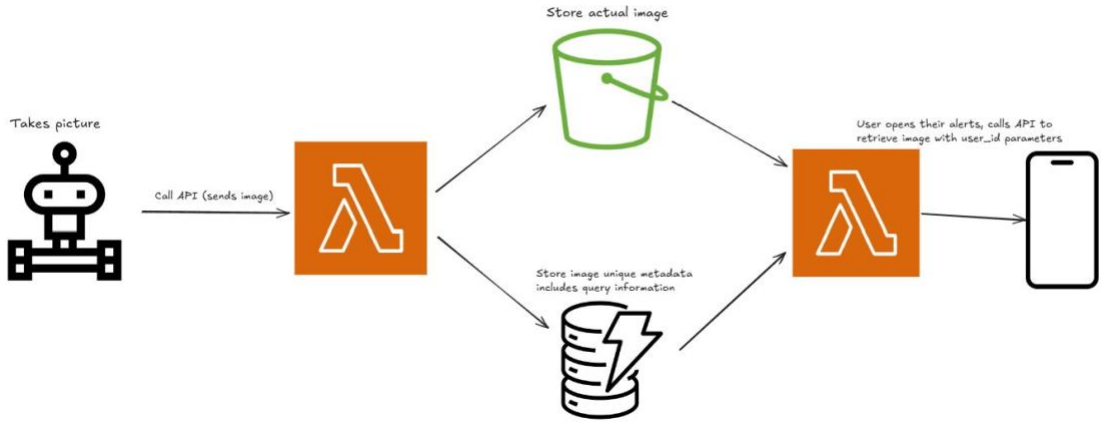

Figure 36: Lambda + S3 + DB flow for storing and retrieving media.

The database schema was built to represent key parts of the system like users, robots, alerts, screenshots, clips, and notifications. Each media file and alert are linked back to its robot and user using foreign keys, which keeps everything connected and traceable. The tables are normalized to avoid duplicate data and keep things organized, and support for JSON fields (MQTT topics) makes it easy to expand the system later if needed.

Table 5: Robot Table

Robots		
Field	**Type**	**Description**
id	INT(PK)	Unique identifier for each robot
uuid	VARCHAR(255)	Globally unique robot identifier
User_id	INT(FK)	References owner user
Channel_arn	VARCHAR(255)	ARN for AWS Kinesis Stream
Pairing_enabled	TINYINT(1)	Boolean flag for pairing mode
Pairing_expires_at	DATETIME	Expiry timestamp for pairing
Mqtt_topics	JSON	JSON object for MQTT paths

Table 6: Users Table

Users		
Field	**Type**	**Description**
id	INT(PK)	Unique identifier for each user
email	VARCHAR(255)	User's email address (unique)
Created_at	TIMESTAMP	Timestamp of user creation
Cognito_sub	VARCHAR(255)	AWS Cognito UUID (unique)

Table 7: Notifications Table

Notifications		
Field	**Type**	**Description**
Notification_id	INT(PK)	Unique identifier for notifications
User_id	INT(FK)	Target user for notification
Robot_id	INT(FK)	Origin robot
S3_filename	VARCHAR(255)	Filename path of media
Media_type	ENUM('image','clip')	Type of media
timestamp	TIMESTAMP	Notification creation time

These core tables serve as the foundation for managing users, devices, and media in the system. Their relationships ensure traceability and efficient data handling across robot events, notifications, and user interactions.

7.5 CAD

For the CAD design of our robot, we utilized Fusion 360 to model the physical structure and layout of the components. One of the key design decisions was to implement a two-wheel drive system combined with a caster wheel for stability and maneuverability.

We started by designing a circular base with a diameter of approximately 15 inches to provide enough space to mount all required components inside the robot. The initial base design is shown in Figure 37. Mounting holes and cutouts were included to accommodate the motors, electronics, and battery.

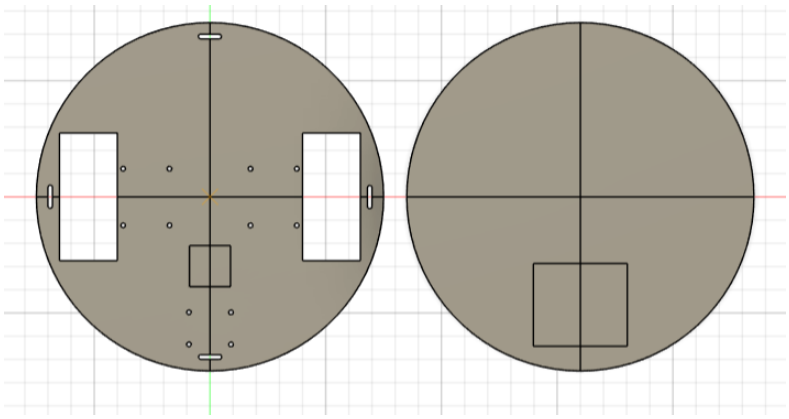

Figure 37: 3D CAD Base Design

The next phase involved designing side walls to enclose the perimeter of the robot, offering protection against external tampering and providing structural rigidity. This wall design is shown in Figure 38.

To increase the height of the robot while maintaining a smooth, aerodynamic shape, we created a dome-like roof with a central hole. This hole allows wiring to pass vertically through the structure. The dome is shown in Figure 39.

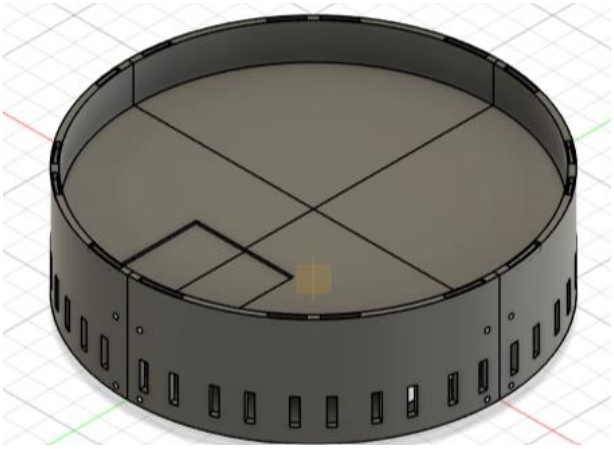

Figure 38: 3D CAD Wall Design

Attached to the dome is a vertical tube assembly with a diameter of approximately 3 inches. This tube provides vertical extension and enough internal space to house a servo motor and a camera mount at the top. The tube design incorporates dovetail joints to allow sections to slide together securely. These joints ensured easy assembly and alignment; once fitted, the sections were permanently bonded using super glue. The tube and dome design can be seen in Figure 40 and Figure 39 respectively.

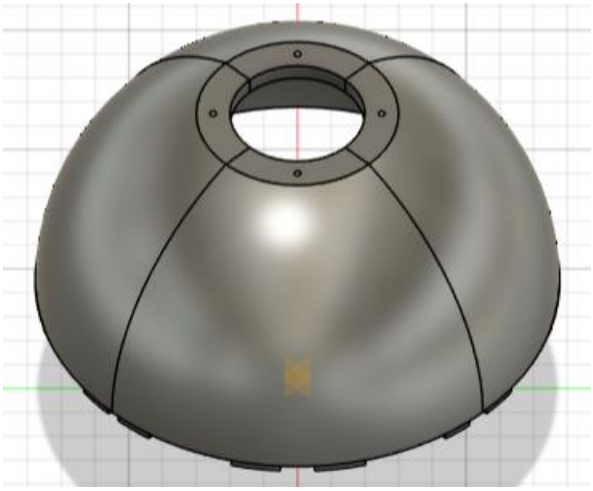

Figure 39: 3D CAD Dome Design

Figure 40: 3D CAD Tube Design

Given the limitations of our available 3D printer, which has a maximum build volume of 10 × 10 × 10 inches, we sliced larger components such as the foundation, dome, and upper tube using Bambu Studio software. We added mechanical connectors during slicing to allow the pieces to be reassembled accurately after printing.

In addition, we designed custom motor mounts to fit our GoBilda motors. These mounts allow the motors to be securely attached to the bottom of the base plate, providing sufficient clamping force to prevent movement during operation. The motor mount designs are shown in Figure 41.

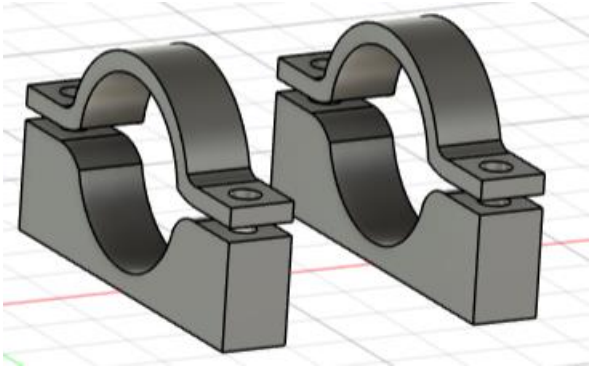

Figure 41: 3D CAD Motor Mount Design

7.6 LiDAR Sensor

The 360-degree LiDAR sensor plays a central role in the robot's autonomous navigation system, providing continuous environmental scanning for obstacle detection and real-time mapping. The sensor integrates with the ROS 2 navigation stack and publishes data visualized in RViz, allowing the robot to plan and adjust its movement based on detected objects.

7.6.1 Initial Mapping and Obstruction Issues

During early testing, it became apparent that the LiDAR's field of view was partially obstructed by the robot's own structure. The main drive wheels and rear caster wheel appeared as fixed obstacles in the RViz-generated map, creating the illusion of walls surrounding the robot. This interference negatively impacted map accuracy and path planning.

Figure 42 below shows a sample map generated before any filtering was applied. The dense circular artifact around the robot indicates false positives caused by reflections off internal components.

Figure 42: Rviz2 LiDAR Simulation

7.6.3 Ultrasonic Sensor Integration

In addition to ground-level sensing, the LiDAR was unable to detect overhanging obstacles such as table edges or elevated ledges due to its fixed mounting height. To address this limitation, four ultrasonic sensors were mounted at different vertical levels along the robot's frame to provide extended height coverage.

These sensors were connected to the ESP32 microcontroller, and data was transmitted to the Jetson Orin Nano using microROS. Rather than continuously streaming raw data from all sensors, the ESP32 locally processed the readings and selected the sensor reporting the shortest distance. Only this value, along with its corresponding sensor identifier, was published to the ROS2 network. This approach minimized unnecessary data traffic while providing critical obstacle detection at the appropriate elevation. In cases where multiple sensors detected nearby objects simultaneously, reporting a single closest reading was sufficient to prevent the robot from proceeding in that direction.

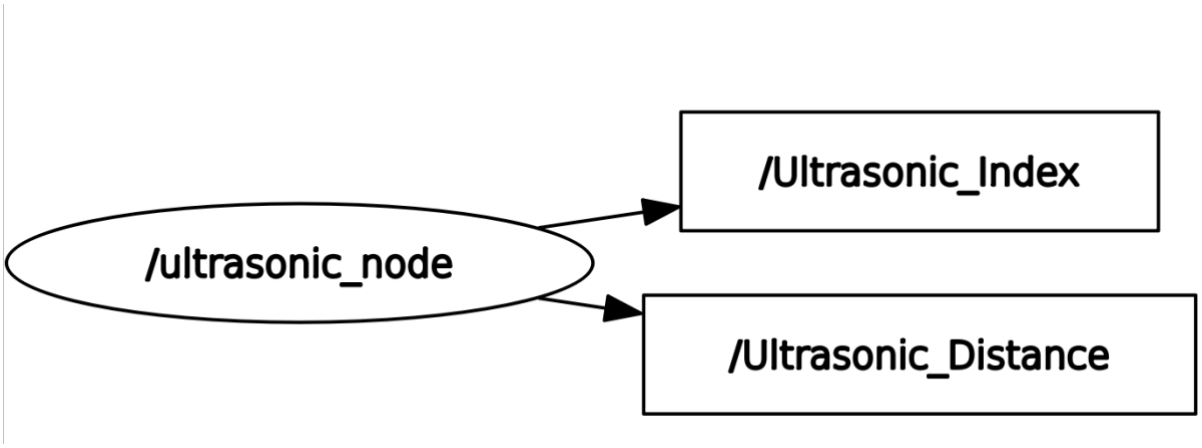

Figure 43: Node and topic structure for ultrasonic sensor data publishing

Figure 43 shows the ROS 2 node graph used during ultrasonic sensor operation. The ESP32 publishes sensor data through microROS, which is then processed by two additional nodes—one that extracts the distance value and another that identifies which sensor detected the obstacle.

Table 8: Checking Accuracy of Ultrasonic Sensors

Test #	Expected Distance (cm)	Sensor 0 Actual (cm)	Sensor 0 Error (cm)	Sensor 1 Actual (cm)	Sensor 1 Error (cm)	Sensor 2 Actual (cm)	Sensor 2 Error (cm)	Sensor 3 Actual (cm)	Sensor 3 Error (cm)
1	20	21.11	1.11	21.52	1.52	20.60	0.60	21.30	1.30
2	30	30.08	0.08	31.53	1.53	30.76	0.76	30.75	0.75
3	40	40.64	0.64	40.66	0.66	40.30	0.30	41.33	1.33
Total Error Sum (cm)	$Sum\ of\ Sensor\ Error$		1.83		3.71		1.66		3.38
Average Error (cm)	$Divide\ sum\ by\ 3$		0.61		1.2367		0.5533		1.1267
Percent Error (%)	$Percent\ Error = \frac{Sum\ of\ Errors/3}{30}\ x\ 100$		2.03%		4.12%		1.84%		3.76%

To validate sensor accuracy, a set of controlled tests was conducted by comparing actual measured distances with known distances. The results are presented in Table X.

These tests demonstrate that the ultrasonic sensors perform reliably within a small margin of error, suitable for collision avoidance in indoor environments. While minor deviations were

observed, the measurements remained consistent enough to support confident navigation, particularly when combined with the LiDAR system for overall spatial awareness.

7.7 Implemented Charger PCB Design

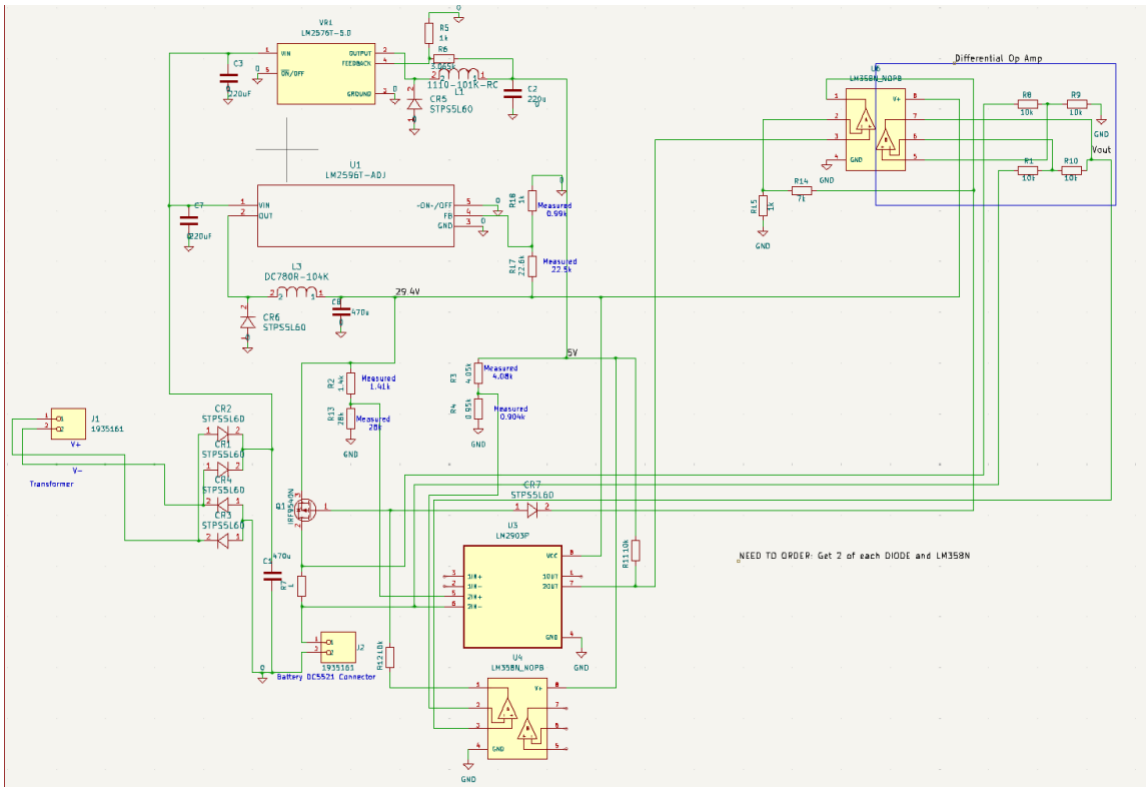

Figure 44: Finalized Charger Schematic

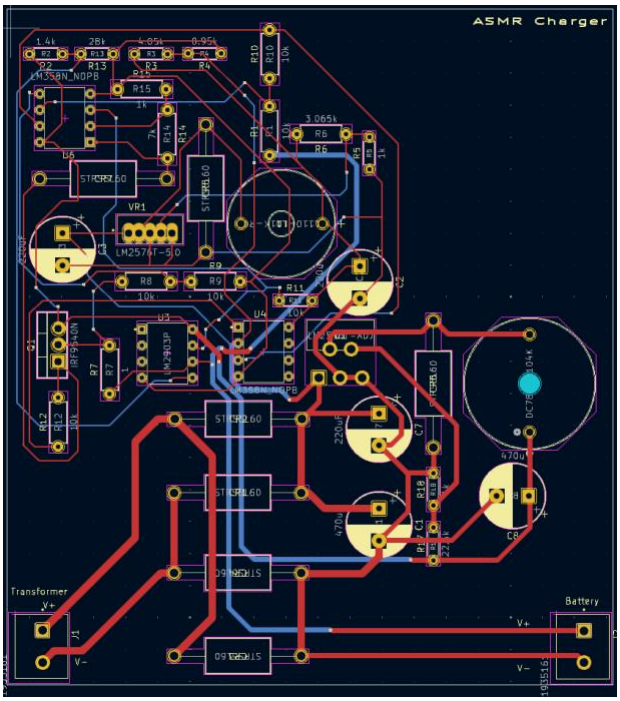

Figure 46: Finalized PCB Layout

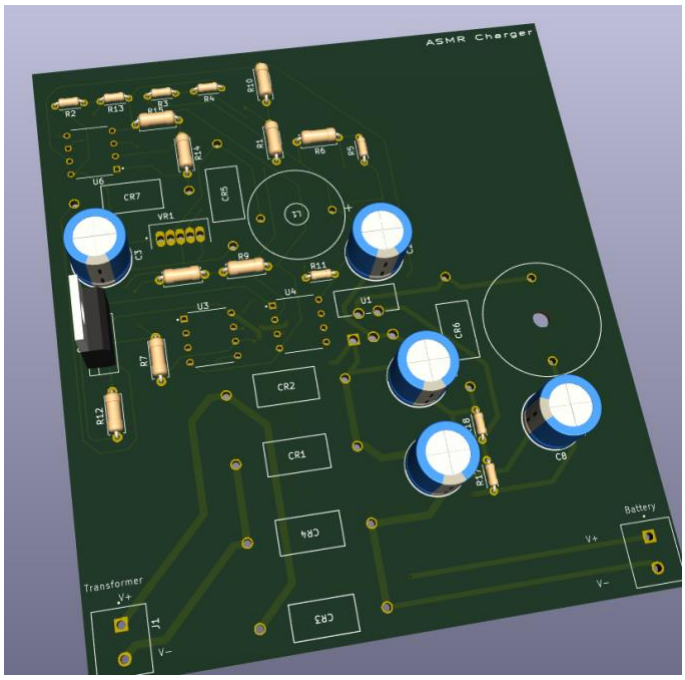

Figure 45: Finalized 3D View

Figures 44, 45 and 46 show the latest schematic for the custom charger PCB. The board is designed to receive input from a transformer with a 5 to 1 ratio, producing an output voltage between 32.5 volts and 33 volts. This AC output is fed into a full-wave rectifier, followed by a 470-microfarad filter capacitor to reduce voltage ripple to approximately 0.5 volts. The filtered DC output is then routed to two separate buck converters: an LM2576T 5.0, which supplies a constant 5 volts for logic-level circuits, and an LM2596T adjustable regulator, configured to output a constant 29.4 volts to charge the battery.

The 29.4 volt charging line passes through a PMOS transistor, which controls whether charging is enabled or disabled. A 1-ohm resistor is placed in series with the source pin of the PMOS and serves as a current sense resistor. The voltage drop across this resistor is fed into a differential operational amplifier configured for a gain of 1, allowing us to monitor the current through the system. This output is then compared against a reference voltage generated by a voltage divider from the 5-volt rail and set to 0.95 volts using another operational amplifier acting as a comparator. The result is used to regulate the gate of the PMOS through a resistor, maintaining a constant charging current of approximately 0.95 amps, which is ideal for the lithium-ion battery.

To prevent overcharging, the system includes a cutoff mechanism. The battery voltage is monitored and compared to a reference value of 28 volts, derived from the 29.4-volt rail. This comparison is handled by an LM2903 comparator. When the battery voltage exceeds the threshold, the comparator outputs a high signal, which is amplified using an LM358 operational amplifier. The amplified signal is sent through a reverse-biased diode directly to the gate of the PMOS. Since the PMOS gate-to-source threshold voltage is between 2 and 4 volts, this high

signal brings the gate voltage close enough to the source (29.4 volts) to turn the PMOS off, stopping the charging process.

This circuit is designed to regulate the charging current while providing a voltage-based cutoff for battery protection. The use of analog components allows reliable real-time feedback without relying on any digital control systems.

7.7.1 Testing Charger PCB

Figure 36 shows the complete PCB when the assembly was finished, we connected both the battery and the transformer to PCB. Initially, the circuit appeared to function correctly for about five seconds—until the transformer suddenly sparked, prompting us to immediately power everything off. Figure 35 highlights the location where the spark occurred.

Afterward, we consulted Dr. Foltz to discuss potential causes of the issue. Following our discussion, we began diagnosing the problem. The first observation was that none of the components on the PCB had warmed up, suggesting that power was not flowing through the board as expected. This points to the possibility of a short circuit somewhere within the system.

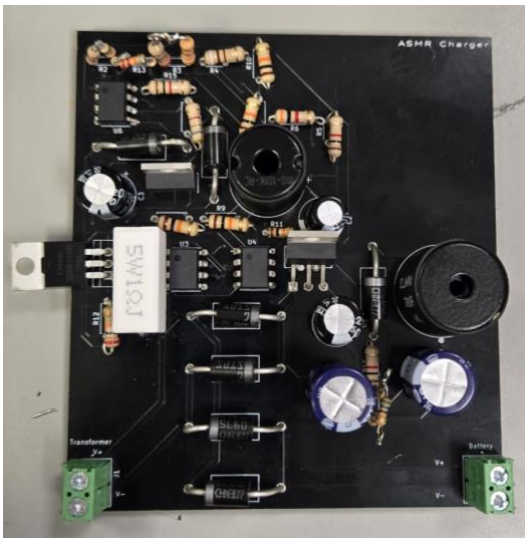

Figure 48: Fully Assembled PCB

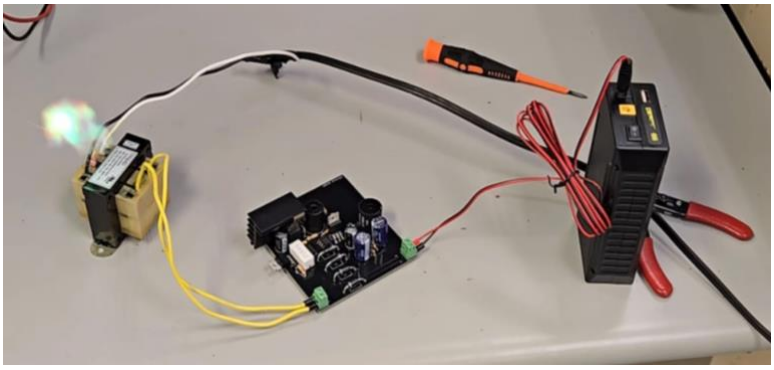

Figure 47: Testing Charging PCB

To diagnose the issue we will be running a few test and documenting the results in the table below.

Table 99: PCB Testing Results

	Expected	Actual/Measured
Continuity Test on Screw Terminals	J1 and J2 are not shorted	Not shorted
Physical Inspection	No burn marks or charred components	Board looks clean and all components are good, and no component got hot during test.
Inspect Full Wave Rectifier and Diodes	All diodes are oriented properly and forward direction shows some voltage.	All diodes are working properly and show a internal resistance of 280 Ohms.
Charging without Transformer	Regulator out 5V and 29.4V, and battery is being charged.	We read 24V then 10V then 1V there must be a short.
Inspecting Inductors Continuity	Multimeters beeps when connecting both pins	Multimeter beeps
Inspecting Phoenix Terminals	Should show some resistance	Not shorted since it reads about 5k Ohms.
Inspecting Continuity on all Resistors (Placing one probe on one leg and the other to the next expected connection)	Multimeter read the resistance value to make sure its soldered onto the border properly.	R12,R3,R10,R1,R14, R11, R8,R9 are not properly soldered which doesn't make proper contact with the board.
Inspecting all ICs (Placing one probe on one leg and the other to the next expected connection)	Multimeter should beep when probing one end of an IC and the other probe to the next expected connection.	Everything is connected properly.
Inspecting all Capacitors	Multimeter should beep when probing one end of a Capacitor and other probe to the next expected connection.	Everything is connected properly.
Testing the Charger with 30V Power Supply	For the Buck1 to output 5V, Buck2 to output 29.4V, and the logic to work properly and allow the battery to charge.	No power seems to go into the board.

After completing the tests shown in the table above, we proceeded to connect a 30V power supply directly to the output of the full-wave rectifier. This was done to isolate the issue and confirm that the transformer was not the root cause, as previous transformer tests had been conducted without a load, which led to overheating.

Upon connecting the 30V power supply, we noticed that no power was being delivered to the board. After further investigation, we realized that connector J1 where the transformer is

typically connected had been left open, resulting in an open circuit. To address this, we connected J1+ to the 30V supply and J1– to the system ground, ensuring a proper ground reference and eliminating the possibility of a floating ground.

We also encountered issues when using the battery. The battery in question is the TalentCell 24V 82.88Wh lithium-ion battery (Model PB240A1), which requires the power switch to be turned ON while charging. If the switch is OFF, the battery behaves as an open circuit, preventing current flow. With the switch ON, the battery provides a shared ground reference for the PCB, and we were able to read voltages throughout the circuit.

However, we suspect that the battery voltage may be feeding back into the circuit, resulting in incorrect voltage readings across the board. It is possible that the internal Battery Management System (BMS) of the TalentCell battery interferes with proper current flow or voltage regulation when connected to the charger circuit. Additionally, operating the transformer and rectifier without an appropriate load may have introduced unstable conditions, potentially affecting subsequent tests even when using the bench power supply. Finally, improper isolation or management between the battery ground and system ground could be contributing to unwanted voltage feedback throughout the circuit.

8. Project Budget Plan

Table 1010: Bill of Materials

Item #	Qty	Part	Description	Price
Fall 2024				
1	1	Jetson Orin Nano Dev Kit	Microprocessor	$500
2	3	ESP- Wroom - 32	Microcontroller	$20
3	10	LM3671	5V Regulator with low Quiescent current	$1.50
4	1	RFID Kit	Mifare RC522 + RFID Cards	$5.69
5	1	SD Card	32GB	$10
6	1	RPLIDAR A1M8	360 Degree, 12 Meter Scanning Radius	$99

7	1	Adafruit SCD-41	CO2, Temp, Hum with built in regular to reduce noise	$50
8	1	Caster Wheel	Backend Wheel	$10
9	1	Lithium Battery	24V, 22300mAh	$54.99
10	2	Motors	36D DC Planetary Gearmotor w/ Encoder, 12V 1600 RPM	$42.24
11	2	Motor Drivers	L298N	$6.36
12	6	Electret microphone amplifier	Module MAX4466	$9.99
Spring 2025				
13	1	TCT40-01E07K	120-24V Transformer	$ 18.10
14	1	839-120-ND	Barrel Jack wire	$ 3.07
15	2	M8343	Fixed Inductor 100uH	$ 18.14
16	2	277-1667-ND Phoenix Terminals	Screw Terminals	$ 1.10
17	2	IRF9540	PMOS	$ 3.94
18	6	Schottky Diode	60V and 5A	$ 2.20
19	1	LM2596T-ADJ	29.4V Regulator	$ 7.16
20	1	LM2576T-5.0	5V Regulator	$4.62
21	1	LM358N	OpAmp	$0.77
22	1	LM2903P	Comparators	$0.27
23	1	SLF7055T-220M1R7-3PF	22UH 2A	$0.98
24	1	74HC32D	IC Gate OR 4CH	$0.55
25	1	Silicon Power SSD	256GB SSD	$19.99
26	2	Wasteland Wheels	144mm Diameter, 52mm Width	$49.98
27	2	PETG Filament	Black Filament	$25.98
28	2	8mm REX mount	Motor Mounts	$15.98
29	2	GoBILDA Motors	8mm REX Shaft, 12V and 1A	$89.98
30	2	Encoder Wires		$7.98
31	5	Ultrasonic Sensors	Sensor to detect Upper Level of Robot	$5.99
32	5	PCB		$42
33	1	Lithium Battery	24V, 22300mAh	$61.99

Purchased -Fall	Purchase – Spring	Purchased – Out of Pocket	Total Price
$309.76	$380.32	$500	$690

Table 10 presents the complete Bill of Materials for the project, listing each component along with its description and associated cost. Although the total cost shown exceeds the university's $800 budget, this is due to the inclusion of the Jetson Orin Nano, which was paid for out of pocket and not included in the reimbursed amount. When excluding the Jetson, the total

cost of all remaining components was $690, with $309.76 spent during the first semester and $380.32 during the second. This breakdown demonstrates that the team remained well within the university's funding limits while successfully acquiring all necessary components, including microcontrollers, sensors, motors, batteries, and printed circuit board materials.

9. Project Summary

The AI Autonomous Security Mobile Robot (A-ASMR) project integrates advanced robotics, cloud-based backend services, and real monitoring into a modular security platform designed for indoor surveillance. Our solution merges hardware and software to create an affordable, scalable, and remotely accessible security system that enhances safety through continuous autonomous operation.

The system features hybrid video streaming capabilities, using AWS Kinesis Video Streams for cloud-based production environment and Flask-SocketIO WebSocket streaming for efficient local testing and development. Real-time video is displayed on a React-based web dashboard, providing users with a live monitoring, camera control, and access to historical security alerts. The dashboard is secured with AWS Cognito authentication to ensure that only authorized users can control and monitor their assigned robots.

Robot telemetry and commands are transmitted using MQTT through AWS IoT Core, enabling reliable and low-latency communication between the robot and the user. A modular FastAPI backend manages robot pairing, user-robot mapping, and cloud storage of alerts, and screenshots, ensuring seamless integration between hardware and software components. The database schema was designed for scalability and traceability, with robust links between users, robots, and media assets.

On the hardware side, the robot features a Jetson Orin Nano for AI tasks such as YOLOv8-based human detection and facial recognition, supported by two ESP32 microcontroller for low-level control. Autonomous navigation would be enabled via a 360 LiDAR sensor, ultrasonic sensor for obstacle avoidance, and a custom motor controller integrated through ROS2 and micro-ROS. A custom designed charger PCB rounds out the hardware, supporting safe and regulated battery charging.

The robot's mechanical design prioritizes durability, stability, and ease of assembly. We chose a four-wheel base to ensure good traction and balance on indoor surfaces, with a simple chassis layout that provides enough space to mount sensor, the camera module, and power components securely. The frame is designed to protect internal wiring and electronics while keeping weight low for efficient motor performance. Mounting points for the camera and ultrasonic sensor were position to maximize field of view and obstacle detection. Overall, the mechanical setup is modular, allowing parts like the camera bracket or sensor mounts to be adjusted or upgraded without requiring a full redesign.

Throughout the project, key lessons were learned about the complexity of integrating real-time systems, cloud infrastructure, and embedded hardware. While some advanced features, such as autonomous navigation, the charger PCB, and the Kinesis stream integration with the human detection, faced technical challenges, the core objectives, live surveillance, secure user access, autonomous navigation, and modularity were successfully implemented. The design lays a strong foundation for the future improvements, including more robust charging solutions, advanced intrusion detection, and enhancements to AI based security features.

10. Conclusion and Recommendations

Although not every feature from the original scope was fully implemented, the team is proud of the progress made and the technical knowledge gained throughout the development process. The robot successfully monitors live streaming, human detection, and control via custom web dashboard, integrating key systems such as MQTT- based telemetry, facial recognition, sound-based security notifications, and anomaly-based alerts.

Several challenges were encountered during development. The custom charger PCB could not be fully integrated due to electrical mismatches between the board design and the battery's internal management system. AWS Kinesis Video Streams, which were initially planned for production-grade streaming, proved unsuitable because of a 3–5 second latency that impacted real-time responsiveness; a WebSocket-based streaming approach was adopted instead for faster local performance. Additionally, the physical design of the robot introduced stability issues due to the wheel placements specifically the center drive wheels and a rear caster wheel, the robot tended to tip forward at higher speeds, occasionally causing the LIDAR sensor to misalign or collide with obstacles.

Another tradeoff was the decision to downgrade from YOLOv8 to YOLOv4-tiny to reduce processing load on the Jetson Nano. While this lightweight model offered faster inference and lower resource consumption, the detection performance accuracy and robustness remained acceptable for the robot's indoor security context. This showed that using a smaller AI model was a feasible compromise when balancing computational limits with real-time detection needs.

For future teams, we recommend allocating more time for hardware validation and prototyping particularly for battery charging circuits before full PCB assembly. We also suggest

exploring lower-latency alternatives or hybrid streaming architectures when real-time responsiveness is critical. From a mechanical perspective, we recommend revisiting the robot's weight distribution and wheelbase design to improve stability. Additionally, integrating the LIDAR sensor internally within a protective enclosure while ensuring clear exposure for environmental scanning would reduce the risk of damage and improve long-term reliability. Despite these challenges, the project provided valuable experience in cloud integration, embedded hardware, real-time systems, and mechanical design, laying a strong foundation for future improvements.

11. Acknowledgement

We would like to express my sincere gratitude to everyone who has supported and contributed to the completion of this Senior Design project.

First, we would like to thank our faculty advisor, Dr. Sanjeev Kumar, for their guidance, expertise, and invaluable support throughout the project. Their mentorship and constructive feedback were crucial in shaping the direction of the design and ensuring that we met the necessary objectives.

We would also like to acknowledge and thank our project team members, Yahir Jasso, Rolando Garza, Eduardo Salinas for their collaboration, dedication, and hard work. The teamwork and cooperation demonstrated throughout this project were key to its success.

Finally, we would like to extend my appreciation to thank the University of Texas Rio Grande Valley for providing the resources, tools, and environment that enabled us to carry out this project successfully.

12. Reference

[1] B. Security, "Handling the Night Shift Challenge: Preventing a Security Guard Sleeping on Duty," [Online]. Available: https://www.bossecurity.com/2024/04/16/handling-the-night-shift-challenge-preventing-a-security-guard-sleeping-on-duty/ [Accessed: Oct. 16, 2024.]

[2] Cobalt Robotics, "Security Robots," [Online]. Available: https://www.cobaltai.com/security-robots/. [Accessed: Nov. 25, 2024].

[3] Knightscope, "K5 Autonomous Security Robot," [Online]. Available: https://cdn.prod.website-files.com/6261e4407c2b850439c5d724/673647e0a1a4f6218dc59632_K5V5.pdf. [Accessed: Nov. 25, 2024].

[4] SMP Robotics, "Thermal Security Robot," [Online]. Available: https://smprobotics.com/security_robot/thermal-security-robot/. [Accessed: Nov. 25, 2024].

[5] Service Robotics, "Argus Brochure," [Online]. Available: https://www.servicerobotics.ai/team1st/docs/argus-brochure.pdf. [Accessed: Nov. 25, 2024].

[6] S. Mischianti, "ESP32 DevKitC V4 High Resolution Pinout and Specs," [Online]. Available: https://mischianti.org/esp32-devkitc-v4-high-resolution-pinout-and-specs/#Datasheet. [Accessed: Nov. 25, 2024].

[7] 5203 Series Yellow Jacket Planetary Gear Motor (3.7:1 Ratio, 1620 RPM, 3.3–5V Encoder), GoBilda. [Online]. Available: https://www.gobilda.com/5203-series-yellow-jacket-planetary-gear-motor-3-7-1-ratio-1620-rpm-3-3-5v-encoder/. [Accessed: Mar. 12, 2025].

[8] Adafruit, "Adafruit SCD-40 and SCD-41," [Online]. Available: https://cdn-learn.adafruit.com/downloads/pdf/adafruit-scd-40-and-scd-41.pdf. [Accessed: Nov. 25, 2024].

[9] Ultrasonic Sensor Module (HC-SR04 Compatible) for Distance Measuring, Amazon. [Online]. Available: https://www.amazon.com/dp/B0CBMTR31F?ref=ppx_pop_mob_ap_share. [Accessed: Mar. 13, 2025].

[10] Readytosky Digital 30KG 360° High Torque Metal Gear Servo Motor with 25T Servo Horn for RC Car, Robot Arm, Helicopter, Amazon. [Online]. Available: https://www.amazon.com/dp/B07Z3VGZNP?ref=ppx_pop_mob_ap_share. [Accessed: Mar Mar. 13, 2025].

[11] Slamtec RPLIDAR A1M8 360° Laser Scanner Kit for Robot Navigation and Obstacle Avoidance, Amazon. [Online]. Available: https://www.amazon.com/Slamtec-RPLIDAR-Scanning-Avoidance-Navigation/dp/B07TJW5SXF/. [Accessed: Nov. 5, 2024].

13. Appendix

13.1 ESP-Wroom-32 Dev Kit Pinout

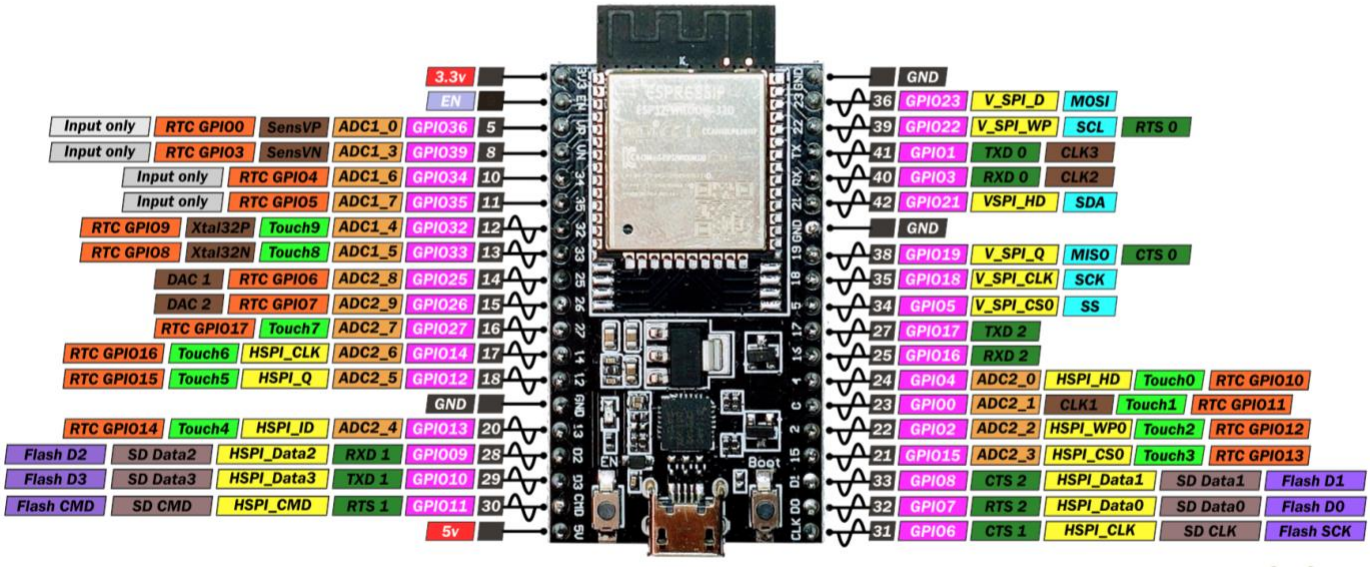

Figure 49: ESP-Wroom-32 Dev Kit Pinout [6]

13.2 12V DC, 1620 RPM GoBilda Motor with Encoder Pinout

Figure 50: 12V DC, 1620 RPM GoBilda Motor with Encoder Pinout [7]

13.3 SCD40 Temperature, Humidity, and CO2 Sensor

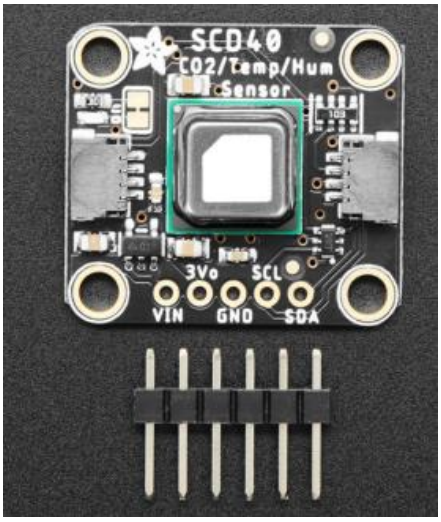

Figure 51: SCD40 Temperature, Humidity, and CO2 Sensor Pinout [8]

13.4 Motor Control Code Snippet

```
// PID controller function
  float pid(float setpoint, float feedback) {
    CurrentTimeforError = millis();
    float delta = ((float)CurrentTimeforError - PreviousTimeForError) / 1000.0;
    error = setpoint - feedback;
    eintegral += error * delta;
    // Anti-windup: Clamp the integrator term
    const float max_integral = 100.0;
    if (eintegral > max_integral)
      eintegral = max_integral;
    else if (eintegral < -max_integral)
      eintegral = -max_integral;
    ederivative = (error - previousError) / delta;
    float control_signal = (kp * error) + (ki * eintegral) + (kd * ederivative);
    previousError = error;
    PreviousTimeForError = CurrentTimeforError;
    return control_signal;
  }
```

```
// Timer callback that controls the motors and computes odometry
void MotorControll_callback(rcl_timer_t* timer, int64_t last_call_time) {
  geometry_msgs__msg__Twist current_cmd = msg;
  float linearVelocity = current_cmd.linear.x;
  float angularVelocity = current_cmd.angular.z;

  // Use half the wheel track for the differential drive calculations
  float halfBase = wheels_y_distance_ / 2.0;
  float vL = (linearVelocity - (angularVelocity * halfBase)) * 20;
```

```
  float vR = (linearVelocity + (angularVelocity * halfBase)) * 20;

  // Reset PID integrators if a significant change is detected
  static float prev_vL = 0.0, prev_vR = 0.0;
  const float threshold_change = 0.5;
  if (fabs(vL - prev_vL) > threshold_change) {
    leftWheel.eintegral = 0.0;
    leftWheel.previousError = 0.0;
  }
  if (fabs(vR - prev_vR) > threshold_change) {
    rightWheel.eintegral = 0.0;
    rightWheel.previousError = 0.0;
  }
  prev_vL = vL;
  prev_vR = vR;

  // Get current wheel RPMs
  float currentRpmL = leftWheel.getRpm();
  float currentRpmR = rightWheel.getRpm();

  // Compute PID control signals
  float actuating_signal_LW = leftWheel.pid(vL, currentRpmL);
  float actuating_signal_RW = rightWheel.pid(vR, currentRpmR);

  // Constrain control signals to safe limits
  actuating_signal_LW = constrain(actuating_signal_LW, -100, 100);
  actuating_signal_RW = constrain(actuating_signal_RW, -100, 100);

  // If command is zero, stop the motors
  if (vL == 0 && vR == 0) {
    leftWheel.stop(pwmChannelL);
    rightWheel.stop(pwmChannelR);
    actuating_signal_LW = 0;
    actuating_signal_RW = 0;
  } else {
    rightWheel.moveBase(actuating_signal_RW, threshold, pwmChannelR);
    leftWheel.moveBase(actuating_signal_LW, threshold, pwmChannelL);
  }

  // Timeout safety: if no new command within 500ms, stop the motors
  if (millis() - prev_cmd_time > 500) {
    leftWheel.stop(pwmChannelL);
    rightWheel.stop(pwmChannelR);
    msg.linear.x = 0.0;
    msg.angular.z = 0.0;
    return;
  }
```

```
  // Drive the motor based on the control signal.
  // This function now limits the PWM output based on maxSpeedPercent.
  void moveBase(float ActuatingSignal, int threshold, int pwmChannel) {
    if (ActuatingSignal > 0) {
      digitalWrite(Forward, HIGH);
      digitalWrite(Backward, LOW);
```

```
    } else {
      digitalWrite(Forward, LOW);
      digitalWrite(Backward, HIGH);
    }
    int pwm = threshold + (int)fabs(ActuatingSignal);
    int max_pwm = (255 * maxSpeedPercent) / 100;  // Calculate max allowed PWM
    if (pwm > max_pwm)
      pwm = max_pwm;
    ledcWrite(pwmChannel, pwm);
  }
```

13.5 Human recognition algorithm

```python
#Open default camera on laptop we can also do a video but we need to specify the video path i try both here
#cap = cv2.VideoCapture(0)
cap = cv2.VideoCapture('/Users/eduardosalinas/Pythonscripts/trimmedVideo.mp4')


#Verify the camera is open if not then just exit the code
if not cap.isOpened():
    print("Cannot open camera")
    exit()

#Starts the video loop where we can for an object we can also do a boolean statement with a variable like isActivate == 1
while True:
    #Reads frame by frame in the camera object
    if not paused:
        ret, frame = cap.read()
    #Verify if its open if not then just exit
    if not ret:
        print("no camera or video found")
        break

    # Run YOLOv8 model on the captured frame
    if frame_count % 3 == 0: #here i utilize frame skip to lessen the resources of the microprocessor
        results = model(frame)

    # Parse results
        for result in results[0].boxes.data.tolist():  # Access the first image in the batch
            x1, y1, x2, y2, score, class_id = result # since result is a tuple we extract x and y but the info we want is score and class_id

            # Check if the detected class is a human (class_id = 0 for 'person' in COCO dataset)
            if int(class_id) == 0:  # COCO dataset label for 'person' is 0
                if score >= 0.80:
                    color = (255,105,180)

                    #implement screenshot here and recording if human with a score of 80 or greater is detected
                    current_time = time.time()
                    if current_time - lastScreenshotTime >= screenshotInterval: #assures a picture is taken every 30 seconds
                        timestamp = datetime.now().strftime('%d_%m_%Y_%H_%M_%S')
                        filename = f'screenshot_{timestamp}.png'
                        cv2.imwrite(filename, frame)
                        lastScreenshotTime = current_time
                        #send post request to the url of our website
                else:
                    color = (255,0,0)
                #use openCV to frame the object
                cv2.rectangle(frame, (int(x1), int(y1)), (int(x2), int(y2)), color, 2)
                cv2.putText(frame, f"Human: {score:.2f}", (int(x1), int(y1) - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, color, 2)

    # Display the frame with detections
    cv2.imshow('YOLOv8 Human Detection', frame)

```

13.6 Ultrasonic Sensor and Code Snippet

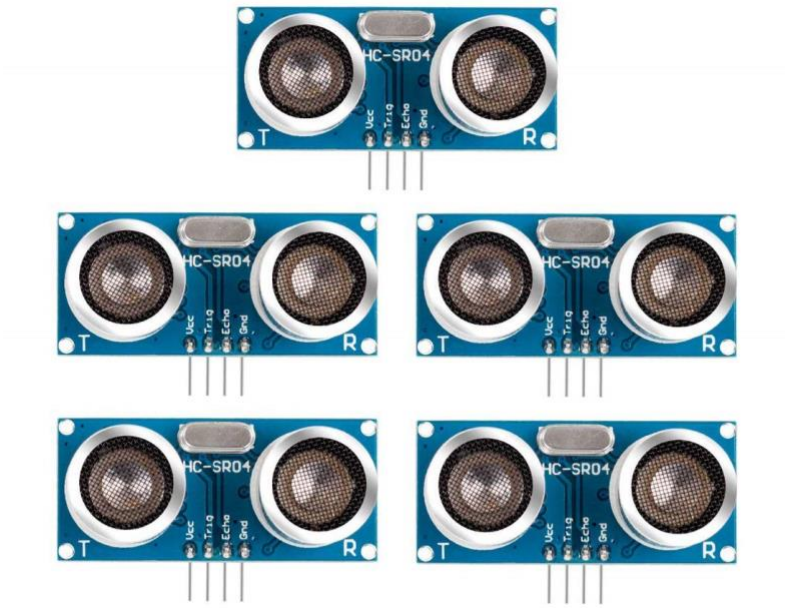

Figure 52: Ultrasonic Sensors [9]

```
  long duration = pulseIn(echoPin, HIGH, 30000);  // 30ms timeout
  if (duration == 0) return 999.0;

  return duration * SOUND_SPEED / 2.0;
}

float getClosestDistance(int &closestSensorIndex) {
  float minDistance = 999.0;
  closestSensorIndex = -1;

  for (int i = 0; i < NUM_SENSORS; i++) {
    float dist = readDistanceCM(trigPins[i], echoPins[i]);
    if (dist < minDistance) {
      minDistance = dist;
      closestSensorIndex = i;
    }
  }
```

13.7 Readytosky Digital 30KG Servo 360 Degree Continuous Motor and Code Snippet

Figure 53: Readytosky Digital 30KG Servo Motor [10]

```
if (turnValue == 1) {
    Serial.println("Nudging LEFT (clockwise)");
    // Publish the Debug Message
    debug_msg.data.data = (char*)"Nudging LEFT (clockwise)";
    debug_msg.data.size = strlen(debug_msg.data.data);
    debug_msg.data.capacity = debug_msg.data.size + 1;
    rcl_publish(&debug_publisher, &debug_msg, NULL);
    // Send a short pulse for clockwise rotation
    myservo.writeMicroseconds(SLOW_CW_US);
    delay(pulseDuration);
    // Then stop
    myservo.writeMicroseconds(STOP_US);
    turnValue = 0; // Reset after actuation
  }
  else if (turnValue == -1) {
    Serial.println("Nudging RIGHT (counterclockwise)");
    // Publish the Debug Message
    debug_msg.data.data = (char*)"Nudging RIGHT (counterclockwise)";
    debug_msg.data.size = strlen(debug_msg.data.data);
    debug_msg.data.capacity = debug_msg.data.size + 1;
    rcl_publish(&debug_publisher, &debug_msg, NULL);

    // Send a short pulse for counterclockwise rotation
    myservo.writeMicroseconds(SLOW_CCW_US);
    delay(pulseDuration);
    // Then stop
    myservo.writeMicroseconds(STOP_US);
    turnValue = 0; // Reset after actuation
  }
```

13.8 A1 RpLiDAR A1M8 2D 360 Degree Sensor

Figure 54: RpLiDAR Sensor [11]