Robotic Observer of Abandoned Materials

R.O.A.M.

Creators:

Daniel Claassen CpE
Nathan Little CpE
Claire Persinger EE
Ethan Robotham CpE

Group 14

Semesters:

Summer 2025 - Fall 2025

Reviewer Committee:

Dr. Xin Xin

ECE Assistant Professor

Dr. Justin Phelps
ECE Assistant Professor

Suboh Suboh
Associate Lecturer

Mentor:

Dr. Chung Yong Chan

Table of Contents

Table Index

Figure Index

Chapter 1 Executive Summary

R.O.A.M. is an autonomous driving, sidewalk-traversable robot designed to identify and report items that we sort as salvageable or trash in residential areas. R.O.A.M. stands for Robotic Observer of Abandoned Materials, and it was created by an interdisciplinary group of individuals to satisfy our senior design project. In this project we will combine software, hardware, embedded systems, and more to create this sustainable and interactive device. In many urban and suburban areas, there is an abundance of overconsumption and limited recycling infrastructures. This results in an over abundance of trash and reusable goods all going to the same place, the streets. A report by EnvironmentAmerica.org states that statistically 12% of the trash on the planet is produced by the United States of America, they also state that 19.5% of the waste produced are durable goods that could be repurposed. We want to address the prevalence in residential neighborhoods of a bulk trash day contributing to some salvageable waste being discarded instead of repurposed or donated. The solution aims to allow people to more effectively find the salvageable bulk trash they would like before it's discarded, and keep streets clean. In tandem with the item-scan routine, the robot continuously quantifies litter density along its path to power targeted community clean-up efforts. We want to be able to extend the life cycle of these items and reduce contribution to landfills. Unlike existing community cleanups, which require organizing groups of people, R.O.A.M. patrols sidewalks autonomously. We are combining real-time detection, classification, and reporting to a user-friendly platform to achieve our goals. To do this we have our autonomous driving robot traverse through a neighborhood searching and scanning these objects. Through the use of our website, we can store and post the locations of these items, and we hope to connect people to trash turned treasure. People in the area will be able to access the website, which will house an interactive online map. On the website, it will display the approximate coordinates of the object it finds, and its general classification. The hope is to have community participation to declutter, recycle, and assist the area they live in. This system is made to serve a multitude of different people: up cyclers, people who want clean streets, and even local waste management authorities seeking to reduce landfill volume. To do this we will combine autonomous navigation, object classification, cellular-based website connectivity, and GPS tracking to tackle these objectives. These will be made possible through several subsystems. We have LiDAR based navigation, machine learning for object detection, and cellular communication for the GPS and real-time data transmission. With LiDAR autonomous driving, we will have to keep the robot driving on the boundary of the neighborhood sidewalk. Additionally, we will have a contingency plan for if our robot gets stuck with an object in its path that it cannot go around. It will signal our website, display the issue on our LCD screen, and release a buzzing signal. We have also anticipated other challenges such as objects in its path or human intervention, because of this we introduced additional contingency plans. These include if our robot tips over or stops driving for any reason, it should also ping the website with a distress signal and display a message on the LCD screen. As for object identification, we will be running a machine learning model on our edge device to classify our objects as trash or treasure. This will be important for our website integration, we will be posting this information with their GPS coordinates to a place where users can access. In addition to all of these technical

challenges, we must create this robot to be safe for public spaces, meaning it will have to operate without disturbing the community. This will require our group to collaborate and find ways to separate the workload to complete all the tasks we have. We were able to overlap many of the objectives we had between people to allow for cross-checking and different perspectives. This paper will continue to outline and delve deeper into the technical objectives of this project, but it will also touch on the methodology and other supporting elements to develop this project. These will include our standards, constraints, and other administrative topics. Through the course of this paper, we hope to give the reader an all encompassing understanding of our project and the scope we hope to accomplish.

Chapter 2 Project Description

2.1 Background and Motivation

R.O.A.M. (Robot Observer of Abandoned Materials) is an autonomous robot that detects and reports discarded and salvageable items near residential sidewalks. The identified items are then uploaded to a website for people in the area to view. Objects are categorized into the two groups using a trained object classification model. The salvageable items, known as treasures, include items such as couches, chairs, or bookshelves. The classification and reporting aims to keep salvageable items out of landfills and to connect them with people who may find good use for them.

The successful operation of this system is dependent on several key functions and components: interfacing a camera to identify objects, LiDAR to direct the autonomous driving, connection to the internet, and an intractable website. The trained object classification model breaks down the images into the different types of objects, trash and treasure. Both classifications are reported to the website with the object classification type and approximate location of the object. The approximate location is sent as a coordinate using a sim card module. We will have to select an object identification model and use OpenCV to implement it. A considerable portion of this project will be training the model with images of trash and treasure within view of sidewalks.

R.O.A.M. is a small, four-wheeled robot meant to traverse neighborhood sidewalks. This restricts the size R.O.A.M. could be, limiting it to about half the width of a sidewalk. R.O.A.M. would house cameras on the top of the chassis so that it can take pictures or videos of the objects it finds on the neighborhood sidewalks with minimal obstruction.

We came up with this idea after seeing how often people leave salvageable or functional items on the side of the road. R.O.A.M. provides a second life to discarded objects before they reach landfills, reducing the garbage on the streets. R.O.A.M. was created to help connect individuals with functional items

2.2 Existing Products and Related Work

There are numerous projects and devices existing on the market that resemble subsystems that R.O.A.M. has, the goal of this section is to discuss similar products and how their systems can be of use to us. We also wanted to discuss possibly the different route R.O.A.M. has taken to these robots that already exist.

2.2.1 Clearbot

Pranjal Malewar, covering the story behind Clearbot, writes about a solar-powered, marine-trash-collecting robot used to identify and collect trash from waterways. This was developed by Open Ocean Engineering along with other organizations to create solutions for the pollution. It uses AI vision to detect floating debris while navigating an aquatic environment. It also relies on GPS guidance to avoid obstacles and maintain its path. Not only can it collect trash, but it has the ability to map underwater terrain, depth, and shape.

Clearbot uses AI technology, surveys and computer models to track an estimate of how much trash enters the ocean and finds it to collect it. The models can also integrate pollution trends over the last few years to find certain zones with more trash. This can also help researchers to collect data on the types of trash in our waterways currently. In the collection process, this bot has the ability to identify and log the types of trash it collects. This process is done through a two-camera system which also uses the AI machine learning system. One of the cameras is to survey the water for trash as well as hazards. The second camera is used for the identification of the rubbish it finds. Once the trash is collected, there is an image and GPS location sent to a data collection site. The obvious objective of this project, to clean up the ocean, is not the only solution it provides. This robot is important to collect data about the types of trash that ends up in the ocean, this data can help researchers and people in the environmental protection sectors. This type of robot is a version of our project made for water. Both of these devices use object detection, but they have slightly different objectives to achieve. While we are hoping to positively impact the environment similar to Clearbot, we also are hoping to connect people with reusable materials. One topic between these projects that overlaps is the use of GPS. We are employing a similar tactic and can possibly research technology references from this project to help us in our own project.

2.2.2 GULP, Garbage Utility Locator & Picker

Another garbage identification project found while researching is GULP. GULP is a solar-powered smart garbage that separates trash to increase the functionality of the waste management process. It was developed to address issues within waste management systems to allow further automated sorting to reduce manual labor. It uses machine learning image processing to separate the waste and even suggest a recommended compressor to create more room for that type of garbage. This item also has the ability to monitor the garbage bins using a SMS notification system to prevent overflow. The message system is triggered when the ultrasonic sensors detect the bin has reached a certain threshold and can allow proactive fixes. This whole system was created using a Raspberry Pi and multiple ultrasonic sensors to aid in the separation of garbage as well. Being able to identify trash types, separate trash, suggest compactors, and detect overflow are relatively intense requirements of a system. This requires reliable classification and accuracy to create a solid database for this, which could have been difficult. In this project, there was an interesting way that they handled their workloads with the image processing. We also see that they used ultrasonic sensors to separate the trash types. The processes they used in this project is something we could consider to look further into as we build our own. For example, their use of ultrasonic sensors to detect fullness of their bins is very efficient. If we looked into implementing something like that it could be for detecting objects in front of our robot. Overall, this project gives very good solutions to technical problems that projects like ours face.

2.2.3 Starship Technologies

There is another robot that is roaming the sidewalks of cities and neighborhoods, it is just completing a task very relatively different from ours. Starship is a small, self-driving robot that is used to deliver packages to people. It travels on the sidewalks to reach its

destination and navigate a multitude of different areas. This robot uses lidar, ultrasonic sensors, and cameras to navigate in the world. The technology of this robot is similar to R.O.A.M. in the way that they both will rely on LiDAR cameras to stay on the sidewalks. The movement of the two robots will need to be able to either avoid obstructions in their paths or just not run into them, which the cameras and sensors will help with. Additionally, both of these robots will use cellular networks to track certain things, the starship seems to track the location of where it is traveling to. Whereas R.O.A.M. is tracking the coordinates of the objects that it finds along the way. The concept between the two is similar and when thinking about what technology we needed to use to achieve this project we found similar technologies to starship.

2.2.3 Takeaways

While R.O.A.M. shares ideas with the two projects discussed above, It differs in how each part is applied. Clearbot focuses on identifying trash in the ocean and autonomous navigation, while G.U.L.P. focuses on identifying and sorting trash. Using these projects as references, R.O.A.M. is going to extend these ideas to neighborhoods by detecting trash and valuables while driving autonomously on the sidewalk. By referencing projects that work, R.O.A.M. offers a solution to waste management in neighborhoods.

2.3 Goals and Objectives

The overall goal of our project is to be able to deliver the information we have in a safe and affordable way. By combining object detection, autonomous driving, wireless transmission and more, we aim to create a device that can increase accessibility to second-hand items. Along with improving access to second-hand goods, this system can reduce waste and promote community. This design is a practical and reliable device that can be implemented in communities without extensive maintenance. Our goal is to allow all kinds of people regardless of background to easily use the website to participate in this system and benefit from it.

2.3.1 Our Goals

A. Basic Goals

a. Identify salvageable and discarded items near the sidewalk
b. Autonomous sidewalk following.
c. Upload object's classification and location to a server from a distant location.

Identifying reclaimable items is at the center of R.O.A.M. using computer vision R.O.A.M. can classify objects on the side of the sidewalk. This gives the ability to relay that information to the website. Uploading the information is a vital next step to display it to any potential users on the website. The location must be attached once sent to the website, so it is retrievable. Being able to autonomously follow a sidewalk gives R.O.A.M. functionality and purpose. Combining with the other basic goals, this gives R.O.A.M. a full range of features that allows R.O.A.M. to roam the sidewalks and gather information without human control or intervention.

B. Advanced Goals

a. Use a Map API to visualize data collected on our website.
b. Request assistance through human intervention

Once data has been collected, it will be displayed on a website for readability. Using a Map API to put icons on the map at the coordinates provided. This gives information an intractable element to the website, allowing the users to not just read coordinates of the objects but see where those coordinates place on a map. When R.O.A.M. becomes blocked by an object in the sidewalk, it will call out for help, communicating that its path is blocked so it can be assisted.

C. Stretch Goals

a. Further classification to identify treasure type.
b. Integrate a search via location or treasure type on the website.
c. Send photo of identified artifact to website with additional information
d. Solar battery charging system.
e. Battery life indication

To broaden R.O.A.M.'s abilities, we have added goals to a stretch section in an attempt to expand on the non-critical functions of our robot. One way is by improving how it classifies the items it detects. Rather than simply categorizing items into trash and treasure, the system will support more detailed classification of items. These items will be displayed on the website with additional abilities to filter or search. To extend its use time, we plan to incorporate solar capabilities. This would allow more time between charges, as we can supplement it with solar energy. Additionally, we plan to have a battery life indicator to the LCD screen to improve functionality and monitor the robot's status in real-time.

2.3.2 Objectives

A. Identify salvageable and discarded items on sidewalks in neighborhoods.
    a. Camera Communicating with Edge device.
    b. Trained object identification.
B. Autonomous sidewalk following.
    a. LiDAR Sensor that communicates with Edge Device.
    b. Algorithm to make a 2D point cloud and interpret data, identifying Sidewalk.
    c. Further data interpretation to adjust motors.
C. Upload an object's classification and location to a server from a distant location.

a. Server to host a website.

b. GPS Locator working on R.O.A.M.

c. Ability to communicate information from R.O.A.M. to server.

The objectives of R.O.A.M. are derived from our project goals. Each goal requires several different components to achieve. The basic goals outline the core functionality of R.O.A.M. which include achievable items that we believe can be completed within our timeframe. The first objective is to identify items and classify them as either salvageable or discarded, to do this we will need to employ a camera that can communicate with our edge computing devices. The robot will use trained machine learning for object identification. Additionally, we want our robot to function as an autonomous entity which will have a few objectives to meet. Autonomous functionality will entail having LiDAR sensors that will be able to communicate with our edge device. These will be responsible for sending information to our ESP32 to then use further interpretation to adjust our motors. Another goal is to upload information about detected objects and their coordinates to a website. This would require GPS tracking and communication protocol between the edge and the website. We also need to host a website to provide that information.

A. Use a Map API to visualize data collected on our website.
   a. Upload GPS coordinates with trash or treasure markers.

   b. Employ map API.

B. Asks for help when the path is obstructed.
   a. Use a buzzer to audibly signal an object in its path.
   b. Use an LCD screen to visually display this information.
   c. Website notification to alert the robot is stuck.

The advanced goals of R.O.A.M. build on the basic goals to enhance the core functionality, improve ease of use, and provide insurance for edge cases. These objectives are more difficult but remain achievable. We aim to display the date collected on a map. To execute that idea, we would need the GPS coordinates of those items, as well as a map API. Additionally, since our robot is autonomous, we need to be able to handle a case where our robot's path gets blocked. To do this, we plan on adding a buzzer to alert people nearby, as well as an LCD screen with signals for assistance on it. Finally, we will send a signal to our website that the robot needs assistance.

A. Further classification to identify treasure type.
   a. Another object classification model running on the base server.

   b. Ability to send treasure images to the database for analysis.

B. Integrate a search via location or treasure type on the website.

   a. Categorized Database.

b. Further classification of treasure

c. GPS coordinate translation and organizer.

C. Solar battery charging system.

a. Charge Controller.

b. Access to battery.

D. Battery life indication

a. Voltage divider to “read” battery.

b. Sending it to the LCD screen.

The stretch goals of R.O.A.M. add features that build off existing concepts within R.O.A.M. but aren't required for core functionality. Instead, they improve R.O.A.M. by working alongside existing components. To enhance its classification capabilities, one of our objectives involves running a more advanced model on the server to allow hyper specific identification. This would also include sending photos that the robot takes of the objects to the website. Another improvement includes integrating searching on our map, enabled through additional object categorization and better GPS data structuring. Solar battery charging was another topic we discussed, allowing the autonomous robot to spend more time on the road without intervention. This would require a solar charge controller and access to that battery to charge it. Finally, we want to add a battery life indicator on the LCD screen, to provide a real-time update for users that interact with our robot. For this we would need to use a voltage divider and connect it to our ESP32, which with some code can allow us to read the voltage of our battery.

2.4 Functionality

2.4.1 Autonomous Driving

R.O.A.M. will use LiDAR and an analyzing algorithm like RANSAC. LiDAR’s main function is to provide the information that makes it possible to identify the sidewalk, allowing for R.O.A.M. to stay on the sidewalk. The LiDAR will interact with our edge device that will run the algorithm and interpret the current position with the sidewalk to provide directional correction to the ESP32. The ESP32 will control the information sent to the motors.

2.4.2 Object Detection & Classification

A large goal of this robot is to use an object classification machine learning algorithm implemented on an edge device fitted with a camera to identify and classify objects on the side of the sidewalk as trash or treasure. This feature drives our main goal of allowing users to reclaim discarded objects that would be useful, such as a used chair for a college student. The information gathered would be accessible on our website for people to interact with. This will reduce the amount of unnecessary waste on the side of the road.

R.O.A.M. will also track the trash it sees, so areas with high volumes of trash are noted and cleaned up.

2.4.3 Interactive Website

The website will organize and provide information for the public to view and use to their discretion. The website will be sent information by R.O.A.M. depending on what is seen by the robot. If treasure is seen like couches, chairs, or otherwise usable furniture, a package from R.O.A.M. will be sent to the website containing the treasure classification, an image, and GPS location of the treasure. This could show up on an interactive map or in a list format. Information provided about the trash in the area will be displayed differently from treasure, since the exact location of a piece of trash is not required, R.O.A.M. will send the amount of trash it's seen every 5 minutes with a GPS location and then reset the trash seen variable. This will aggregate the amount of trash seen by R.O.A.M. between two GPS locations.

2.5 Key Specifications

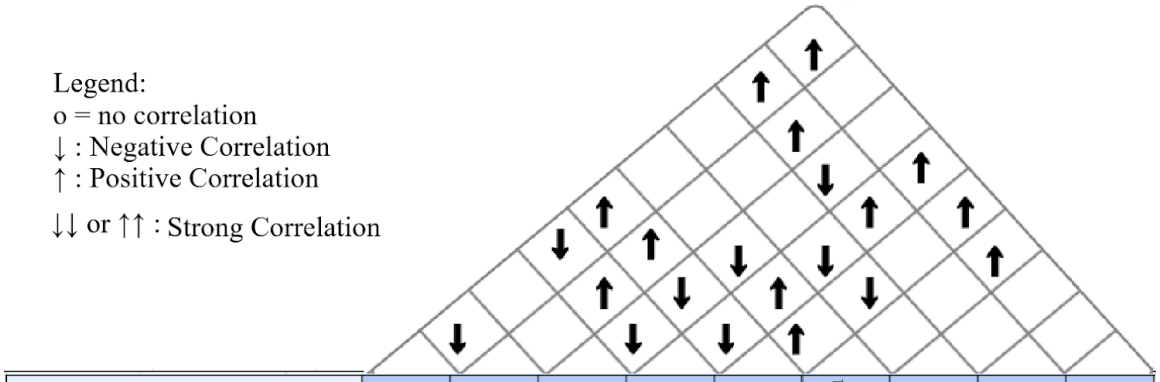
Table 2.1 — Key Specification

Parameters	**Specifications**
Speed (legal requirement)	< 10 mph
Width (wheel to wheel)	< 46 cm
Traversable Obstacle Height	< 35 mm
Website Data Upload Time	< 10 seconds
Object Detection Response Time	< 2 seconds
Treasure Identification Accuracy	> 70%
Battery Life	> 15 minutes
GPS accuracy	Within 20 m
Distance traveled without going out of bounds	> 100 meters
Object Detection Range (R)	0.5m < R < 3m
Power consumption	< 55 W
Cost	< $1500

2.6 House of Quality

Our marketing requirements in Figure 1.1 are aimed at a safe and easy-to-use autonomous robot. With this, a lot of consideration will be placed upon the reliability of this functioning bot. This is not the only driving force behind our engineering development as GPS accuracy and operating time will be important deliverables on our reliability front, while remaining under $1500 and with a width to remain within 46 cm.

Table 2.2 — House of Quality



Engineering Requirements			Battery Capacity	Prototype Width	Object Identification Accuracy	Cost	Battery Life	Obstacle Detection Response Time	GPS Accuracy	Object Detection Range (R)	In-Bounds Distance
Marketing Requirements			+	-	+	-	+	-	+	+	+
	Portability	+	↓↓	↑↑	o	o	↓	o	o	o	o
	Durability	+	↑↑	↓	o	↑↑	↓	o	o	o	o
	Reliability	+	o	o	↑	↓	↓	↓	↑	↑	↑
	Affordability	+	↑↑	↓↓	↑	↓↓	↑↑	↓	↑↑	↑↑	↑↑
	Usability	+	↑	↓	↑	↓	↑	o	↑↑	↑↑	↑↑
	Safety	+	o	↓	o	o	o	↓↓	o	o	o
Targets for Engineering Requirements			> 25 Ah	< 46 cm	> 70%	< $1500	> 15 min	< 2 s	< 12 m	0.5 m < R < 3 m	> 100 m

2.7 Hardware Description

Figure 1.2 describes the basic system of our project's Hardware. Beginning with the power supply section, there is a battery that will be the main contributor to the rest of our hardware. That will supply the power through our regulators which will be a part of our PCB board, this is the connection point to the rest of the system. Going through the regulators allows us to protect the rest of our equipment in case of an unwanted supply of

amps. The three paths from the regulators are the ESP32, the edge device, and the motor controllers. From there we have a multitude of different functions that will happen. With the regulators supplying power to the ESP32, it then can send and receive data. One of these data paths is to the edge device, that edge device is also sending data to the ESP32, this data would include images and data about the object identification for the website. On top of that, the ESP32 will also be receiving information about how to adjust the motors based off of the LIDAR sensors. LIDAR, in conjunction with the edge device, will determine the movement of the robot based on the environment, and send that information to the ESP32 to then control the motors. We additionally aim for the ESP32 to communicate with the edge device as with a communication protocol, they will need to acknowledge each other. The last thing interacting with the ESP32 is the GPS. The GPS is going to be tracking the coordinates of our objects, which we will then post to the server. Since the ESP32 communicates with the server, we will send the GPS information to the ESP32 as a middle connection. From the esp32 and sim card module, we will be communicating information wirelessly via cellular data to reach the main server hosting the website. The server is our website, which will store the coordinates and classifications that it receives. This sensor is to allow our robot to autonomously drive.

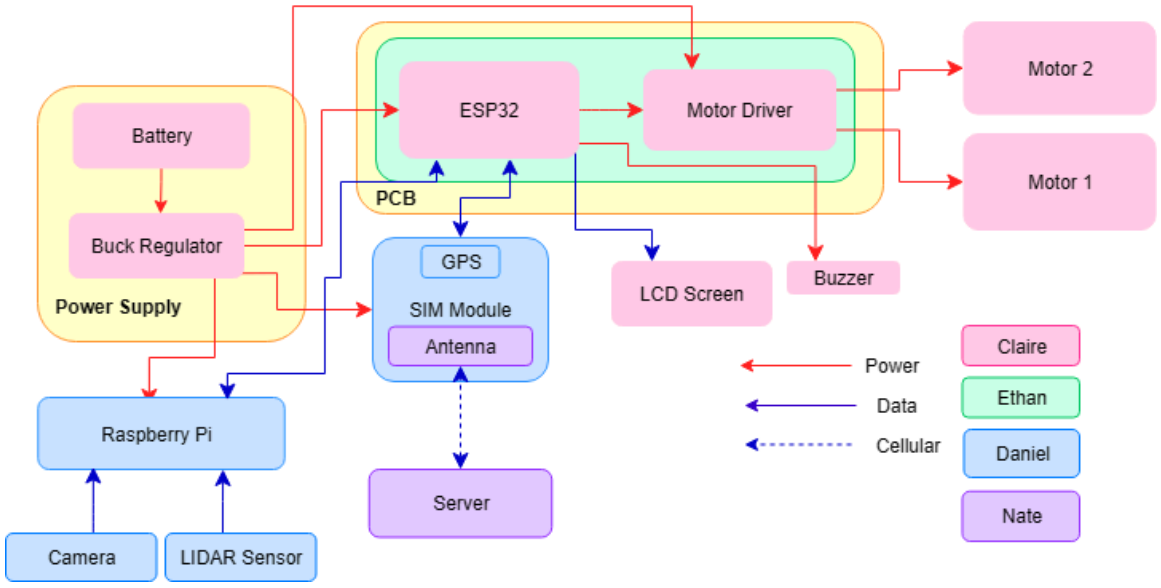


Figure 2.1 — Hardware Block Diagram

2.8 Software Description

Figure 1.3 is the outline of how the software of R.O.A.M. behaves overall and how the subsystems interact with each other. The software serves three major purposes: autonomous driving, object classification, and data transmission. The autonomous driving works by using LiDAR to scan the local environment in front of R.O.A.M. and create a 3-D point cloud of the surroundings. This point cloud will be used to identify the sidewalk and thus provide directional adjustments to the motors. R.O.A.M. will only have two motors, directional adjustments will be provided in the form of the voltage supplied to the left and right motor, allowing for turning as well as smaller path

corrections. Data transmission will only happen in two scenarios, either a treasure is found or the trash amount needs to be updated. If treasure is found, then a package is sent to the ESP32. The ESP32 will take the package and attach a GPS location to it which will be sent to the website making the information and location viewable. If trash is found an internal counter will increment by one, this will be set on a timer that would periodically update the website with how much trash was found.

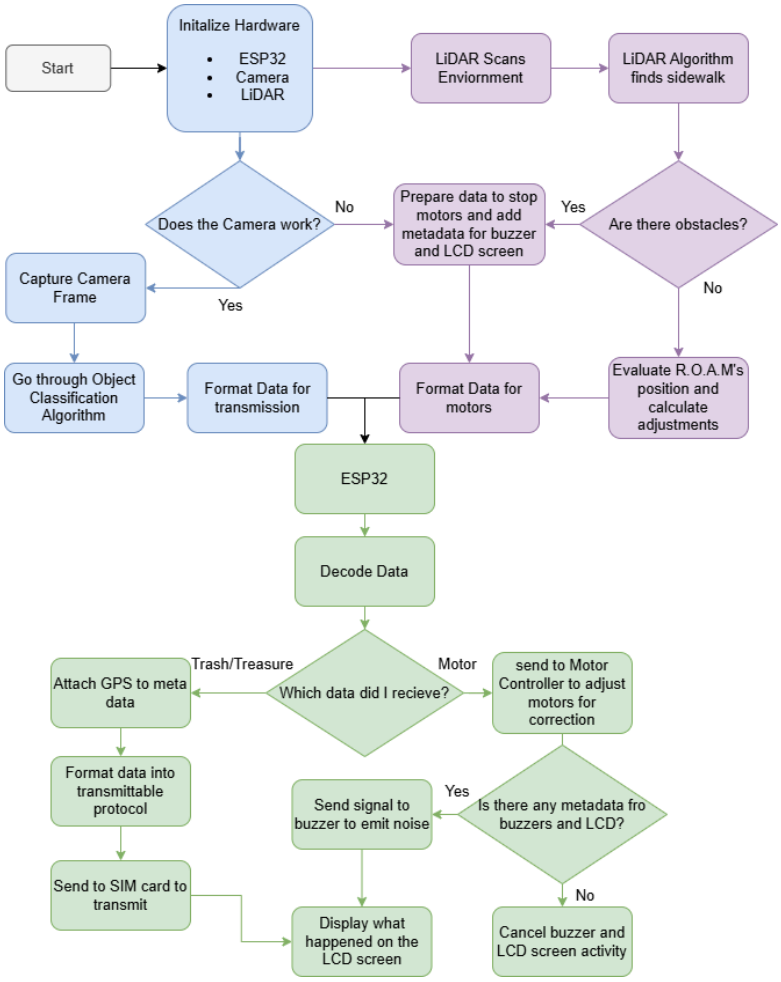


Figure 2.2 — High-level Software Flowchart

2.9 Prototype Illustration

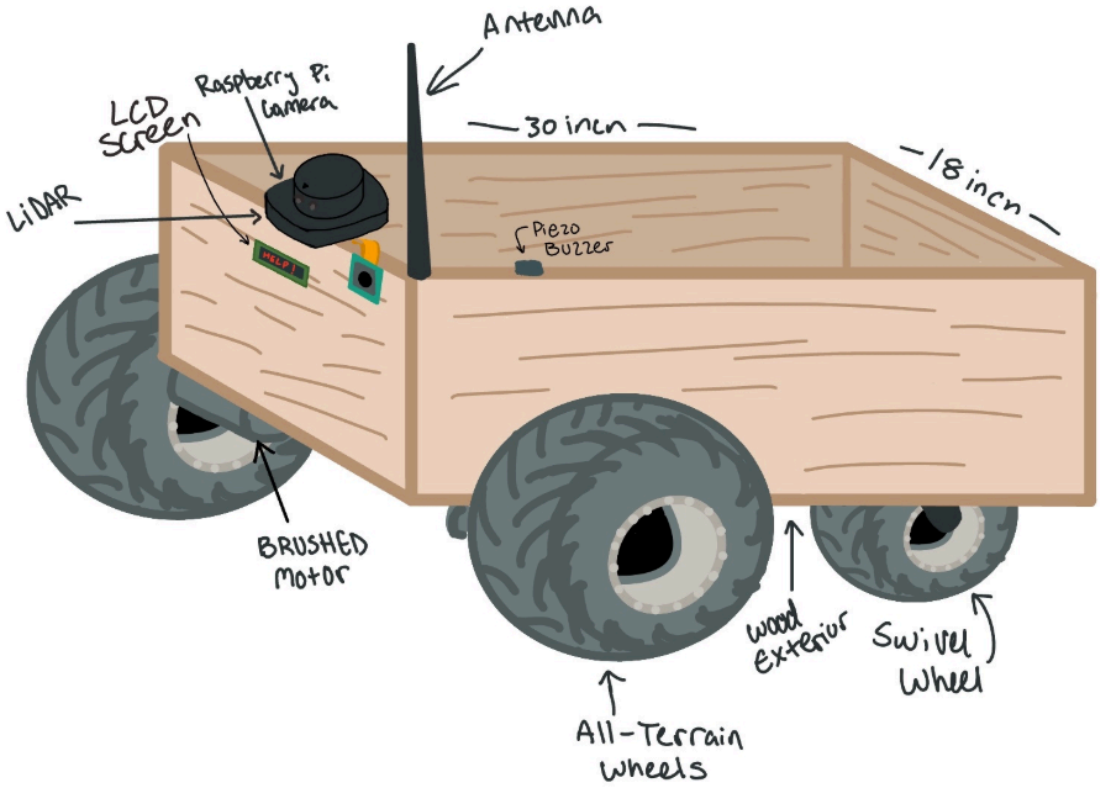


Figure 2.3 — Prototype Illustration

This image above is our proposed idea of what our product will look like. A few things to define are the larger wheels to compensate for cracks in the road, the LIDAR sensor, and the camera. The camera will be used for object detection, and the LIDAR sensor will be used for autonomous driving. These will both be attached to our edge device to assist with the image processing before sending the information to the ESP32 for our website. This product has a few things that are contingent upon more research. For example, what specific types of tires are used, the material for the body frame, and even the exact measurements of the body. This image is an approximate description that gives our audience a general idea of what we are creating, along with approximate expectations to consider.

Chapter 3 Research and Part Selection

In this section, we will spend time learning about the different systems and needs that R.O.A.M. has. To figure out the best way to configure this project, we need to look through multiple different versions of the options we have. This will include the best way to have our robot move, what language we should develop our embedded system code in, how we want to communicate with our website, and more. These preliminary steps to our project are essential for making sure we have the best configuration possible to produce the best version of this device. This section will also further allow a more careful selection of such topics. The following paragraphs will go into detail about what technologies are needed and what would be the best way to go about them.

3.1 Mobility Hardware

3.1.1 Drive System

In this section, we would like to review how we will add mobility to our design. This robot needs to be able to traverse through a neighborhood and stay within the boundary of a sidewalk. This is important to deliberate on as without a functional driving car we would not be able to have autonomous driving. Without autonomous driving, we would not be able to collect the object identification information. Typically, with urban sidewalks there are slight turns, or even corners that are in need of traversing. Additionally, sidewalks also are not perfect, the robot will have to be able to drive over uneven cracks and other debris that may be on the sidewalk. Taking all of that into consideration, there are a few obvious ways we thought we could overcome those issues. Those being a tank track drive system, car-like steering system, and a dual-motor front-wheel system with rear swivel wheels. These three were the top ideas for mobility with R.O.A.M. which all provide pros and cons.

3.1.1.1 Tracked Drive System

The first system to discuss is the tracked drive system. This is similar to a tank track where you have two continuous rubber tracks for wheels which are propelled by two motors. There are a lot of pros on this, as it is the best option for balance and debris. Since there are two continuous tracks on either side of the robot, it would elevate traction, balance, and debris. This is an option if we were to take those into higher consideration, as the more surface area of the tires would allow for more speed and an easy time traversing over uneven sidewalks or debris on the sidewalk. There are a few more negatives to this type of mobility that we did identify. This type of design would be more complex for calculations. The track system would require more precision to avoid derailing of the rubber tracks and avoid slippage. To add to that, the tracks would have to be enclosed in some way as the debris on sidewalks are not easy to plan for as there could be numerous obstacles. Debris such as twigs could become stuck in this system if it was not properly enclosed, this means you could have operational issues. This robot is supposed to be mostly self-sufficient, so if we have to worry about constant maintenance to pull debris from these tracks, then it would not be self-sufficient enough. With these tracks, while it will have good friction with the sidewalk, this also raises the issue that the

motors will require higher torque. There would also be the issue that turning may be less precise. There will have to be a lot of precise movement that is required to stay on the sidewalks and turn, so precise turning is important. We would need to use skid-steering where one side will increase speed and the other will decrease speed. The rotation will cause a lot of friction and dragging with the tracks, it will not be as efficient. It will also cause a lot of unnecessary and uneven wear and tear. The equipment going into this type of system is also heavier and can cause issues when it comes to energy consumption. With the work it would have to put in to turn and move, it would not be efficient for the energy consumption of this system.

3.1.1.2 Ackerman Steering

The next type of system we considered using was Ackerman steering. This is a type of steering which is designed like a car. The rear wheels are fixed, and the front wheels are free to rotate left and right to allow for steering. The car will follow the arc of that turn to allow the vehicle to move smoothly. This will also allow for a few different radii of turning since you can adjust the angle of the wheel. This is important since we are looking to be able to stay within the sidewalks of neighborhoods. We do not want our robot to veer off the path, and this means having easily controllable and adjustable steering. This steering type means that turning at higher speeds or for more tight corners is more achievable. This also means that any turns that the robot encounters that are not a directly smooth path are still possible for the robot to take. The style here would be very visually pleasing to viewers and possibly the smoothest out of all the options. The biggest drawback to this is the intense complexity of this system. This type of driving has a few significant drawbacks in attempting to incorporate it into our project. The complexity in designing and coding a system like this is relatively complex. This would not only require communication between the ESP23 to the motor controllers, but we now add another layer to have communication for servo steering mechanisms. This portion of the system is the biggest drawback as the complexity takes a step up from our other options since the mechanisms are more advanced than we need. The need to turn on a dime and do so quickly is not a necessity for this project. Most sidewalks we have encountered and are planning for thus far have softer turns. Even the sidewalks that have 90 degree turns are wide enough for us to complete that with a less complicated system. So while this system is technically achievable, it is more advanced and unnecessary for the job that we have at hand. If we wanted to have a more realistic looking robot, something that resembled a car, it could be more practical, but we have other priorities to satisfy.

3.1.1.3 Swivel Wheel System

The next type of system that will be reviewed is less typical, but it involves two wheels in the front and one swivel wheel in the back. We are looking for a low-cost, efficient, and easy way to control this robot. With this system, the front wheels will each have their own motor that will communicate with our motor drivers. These motor drivers will get information on where to go based off of our LiDAR sensor. The swivel wheel in the back will act to support the rear of the robot, as well as to provide more ease in movement. If we had placed traditional wheels in the back, you would be able to see our robot have a more difficult time turning. With the swivel wheel, as we allow one of the front motors to slow and the other speed up, we expect the back to turn with the rest of the robot. The

set-up for this system would be significantly less prone to getting debris on the streets within the tires. This system also may have a better turning radius, as the front of the body will be able to turn without worrying about the back tires or elongated tracks that would have to shift with it. This type of driving would only require two motors, which would remove some of the strain on the load for the power consumption. The logic for this design is also relatively simple, having just two motors to work with you can easily create logic to adjust and control it. The logic going into this would be less intense, and the mechanical movements we would have to put together would also be less complex. With the correct wheels, we will see it traverse over uneven sidewalks and debris. This concept was our favorite of the options as it was the most practical. Not only did it seem like just the right amount of complexity, but it also showed a certain level of efficiency. This system would be best suited for the sidewalks we aim to drive on, this system also would be easy to debug and adjust as needed. Overall, this is the system we chose which we believe to be the most efficient, reliable, and realistic to implement.

3.1.1.4 Conclusion

The evaluation of these three mobility systems allowed us to review the technical specs and different demands each option presents. While the tracked system does offer stability and possible advantages in traversing over objects, the complicated system may be prone to things getting stuck in the tracks. The Ackerman steering offers a visually and functionally appealing option, but the overall complexity of this design introduces unnecessary complications to overcome. We chose the swivel wheel design as it allows for a certain level of complexity while being practical and reliable. The front motors allow a good balance and strength for the robot, while the back swivel wheel allows for good rotation. This design aligns with this project's efficiency and cost goals with a minimal amount of maintenance.

Table 3.1 — Mobility

System Type	Pros	Cons
Track System	● More Friction ● Traversable	● More maintenance ● Less accurate turns ● More wear and tear
Ackerman System	● Visually pleasing ● Faster turns	● Too complex ● Too much power ● Difficult to maintain
Swivel System	● Ease of control ● Easy to program ● Easy to design ● Efficient for power ● Low maintenance	● Slower turns ● Susceptible to debris

3.1.2 Wheels Technology Comparison

Wheels for R.O.A.M. will dictate how well it can traverse over debris and cracks in the sidewalk. While some of our power will come from the motors to be able to actually get past obstacles in the road, our wheels have to be able to withstand those objects. If we chose to use wheels too small they risk getting stuck in larger cracks, if we choose too big, it would be harder to turn in traverse. We found a range between a radius of 2 inches to 4 inches which we believe can do the job to get R.O.A.M. to where it needs to be. The wheels in the back also are the type that will be up for debate, the back wheels will be a similar design except they will be mounted with a swivel wheel structure so that it can be guided by the motors in the front. There were three types of wheel we considered, omni wheels, pneumatic wheels, and all terrain wheels.

3.1.2.1 Dagu 120 × 52 mm Solid PU Wheel

The Dagu 120 × 52 mm wheel features aggressive lug patterns molded into a solid polyurethane body. Its straight 8 mm steel-bushed bore accommodates a 6 mm adapter sleeve, eliminating custom hub fabrication. Tested to carry 15 kg per wheel, it withstands impacts from gravel, glass, and concrete irregularities without rolling flats or punctures. At 220 g mass, it introduces higher rotational inertia, which may marginally slow acceleration, but its deep tread maximizes grip and stability during obstacle traversal. Priced at $15.99, it represents a cost-effective solution with minimal ancillary hardware.

3.1.2.2 VEXpro 127 × 38 mm Extruded PU Wheel

The VEXpro extruded wheel measures 127 mm in diameter and 38 mm across its uniform tread surface. Its 3/8 in (≈ 9.5 mm) hex bore interfaces to standard hex shafts; an off-the-shelf hex-to-6 mm adapter or a simple custom hub bridges to R.O.A.M.'s preferred axle. With a payload rating of 10 kg and a weight of 200 g, this wheel excels in low-inertia rolling while maintaining sufficient adhesion on loose or uneven surfaces. The narrower tread and smooth profile yield reduced lateral stability compared to deeper-lugged designs, but the lighter wheel contributes to lower energy consumption during sustained motion. Unit cost is $12.95.

3.1.2.3 Actobotics 102 × 38 mm PU Wheel

Measuring 102 mm in diameter and 38 mm wide, the Actobotics polyurethane wheel mounts via an 8 mm bore fitted with a set-screw-style aluminum hub. Rated to 12 kg per wheel and weighing 180 g, it balances rolling resistance and traction through a moderate lug depth. Its smaller diameter limits clearance for deep cracks, yet reduces overall wheelbase height and inertia, improving turn responsiveness. The set-screw hub simplifies attachment but may require periodic retightening under heavy vibration. The cost per wheel is $13.49.

3.1.2.4 Conclusion

All candidates meet the basic criteria of diameter, load capacity, axle compatibility, and budget. The Dagu 120 × 52 mm wheel offers the most aggressive traction, highest safety margin under load, and the simplest integration path via its straight bore. Although heavier, its deep lugs ensure reliable passage over debris and cracks without supplemental maintenance. Accordingly, the Dagu wheel is selected for both front drive and rear swivel positions; the rear modules will use identical wheels paired with standard 8 mm caster brackets to maintain consistent handling and torque requirements.

Table 3.2 — Wheel Part Selection

Wheel	Pros	Cons
Dagu	• 15 kg rating • Minimal maintenance	• Limited shock absorption • Heavier
VEXpro	• High efficiency • Low-cost	• Stability issue • Low interoperability
Actobotics	• Easy Connectivity • Lightest	• Small tolerance • Mounting security issue

3.1.3 Motors Technology

When considering what type of motor to get there are a lot of topics to put into consideration, these being the weight of the rest of the body, the type of battery used, how fast we want it to move, and more. To figure out what motors would work best, thorough research went into three types of motors and how they could be incorporated into the design we had. We wanted to make sure we knew how they would affect the system and what they would need to be included. The types of motors we reviewed were stepper motors, brushless DC motors, and brushed DC motors.

3.1.3.1 Stepper

The stepper motor is an electric motor that consists of a stator and a rotor, similarly to most motors. This motor is unique since the rotor and stator interact through timed pulses to get this motion. There are teeth on the stator where coils are energized, creating a

magnetic field. This field is supplied with different phasing, which pulls the rotor in certain directions to follow that magnetic field polarity. The rotor will typically have magnets to align with the stator's field. This causes the rotation of the rotor to be as much or as little as you design it to be based on the energizing you send. The stepper rotates in discrete steps, which allows for precise control. This control does not require a feedback system to give that precise control. You can control these movements and sequences through precise steps with the rotor, meaning you can control the movement. This motor can provide a low speed torque for the system and can hold its position when stopped. The logic needed for this is not incredibly complex. A microcontroller or driver will generate the pulse that is needed to move the motor reliably. The high torque properties do create a limit on the top speed unfortunately. With higher speeds, the torque will decrease significantly and can become too strong when the acceleration is not managed. For this project torque is more necessary and important than speed is, but the option to be able to have a powerful motor in the area of speed is an option we would like to have. There is also a property in the step motor which can cause issues without it having a feedback loop, this is it getting overloaded or jammed. The controller in this situation has no way to detect missed steps, to do that we would have to extend the features that provide feedback and sensors. This can be an issue of functionality where we could have our product stop functioning which would be an overall problem. To mitigate these issues, people often add encoders or monitors to detect the stalls. This is not something we had planned to incorporate as it adds more unnecessary complexity to this motor system meaning we have moved away from the step motors in the end.

3.1.3.2 Brushless DC Motor

As for the brushless motor, this type of motor is kind of the inverse of the brushed motor, it still has a stator with coil and a rotor with magnets, but it does not involve any brushed commutator. This type of motor relies on traditional brushed motors with electronic controls to create the rotation that it uses. On the stator, there are coils wrapped around each tooth which is responsible for the electromagnetics. This is fixed and does not move, the rotor on the other hand does move and relies on having permanent magnets to create that connection. As the coils are electrified there are sensors within the motor to detect the position and switch the current through the stator. This is how we get the movement of the rotor. This is a bit different from the brushed motors because the current switches through the stator in sequences creating this rotating field. If there are no sensors then the back EMF typically is what assists the position of that rotation. This type of motor is more reliable than the brushless for a few reasons. It is more efficient, because there are no brush to commutator interaction, there will be less friction, and less energy wasted in heat. The electronic communication also allows minimized waste because it can have precise current timing. The brushless motor can also deliver higher torque than its competitor, this is because of the magnets chosen, they are more powerful. The ability for electronic control also means that the current can be instant and precise, so rapid acceleration is available. These types of motors also last longer than their competitors since they again do not have that brush commutator, there is no wear and tear over time with these things moving against each other over time. This also has a decreased amount of heat due to friction which means there is overall less stress put on the system. With the amount of work it does, the brushless motor also has ways of staying cool to increase the

lower stress from less heat. The excess heat that can come with these motors may present increased issues. There also can be issues with the speed being out of control at high values. More than this, there may be issues with noise from the motor because the electromagnetic field can cause extra noise. For many of these reasons the brushless motor, while presenting as a good idea, can present issues of overheating while also being a significantly more complex system.

3.1.3.3 Brushed DC Motor

A Brushed DC motor is a kind of motor that uses brushes and a commutator to transfer current in the motor. The components will work together to ensure the rotating remains continuous and reliably. This transfer is what causes the motor to spin. The DC motor uses coils to create a magnetic field and rotate the shaft, this is a part of the rotor. The rotor is a part of the motor that rotates and attempts to minimize the vibration during the movement. On the stator, there are magnets positioned on the inner stator. These unmoving magnets work with the magnetic field to create the turning shaft. As the current flows, the coil rotates, until there is no longer the contact which stops the current flow to the coil. Then, as the momentum causes it to continue to rotate, this will cause the brushes to be in contact again. The switching of the current produces the perpetual rotation of the brushed motor. Because of their simplistic design, there are a lot of positives with choosing this design. These involve a simple way to control them. They do not need a drive circuit, there is also a direct voltage control with speed and reversing is easy to control. This ease of use makes them good for applications where it is not needed to be forever and the long-term is not a priority. They are also very available with a high torque and high speed abilities. Many models are designed to bring consistent voltage with the load. This could be beneficial for our project as we would not need an intense amount of complexity or equipment to function these motors. This type of motor does face some electrical noise from the commutator, which can cause some of that inefficiencies. As well as it is seen that over time this type of motor will wear down. This issue is minor in comparison to the simplicity and cost that we will achieve with this system. With the low complexity of this motor in integration along with its availability, this motor seems like a promising option.

3.1.3.4 Conclusion

In determining the most suitable motor type, careful assessment of the precision, efficiency, and integration complexity were reviewed. The stepper motor offered precision movements but introduced more monitoring equipment which added unnecessary complexity. The brushless motor while bringing complexity and long term reliability, also brings complications and more parts for integration than we want. The brushed motor while presenting challenges for long term wear and tear, overall provide ease of use and reliability that we can use to complete this project. Balancing many of those preferences landed us with brushed motors as they allow for high torque, ease of use, and efficiency.

Table 3.3 – Motor Comparison

Motor Type	Pro	Con
Stepper Motor	• Precise position control • Simple control logic	• Low speed torque • High power consumption • Vibration • Can get overloaded
Brushed Motor	• Simple to control (PWM) • Cheap • Available • High torque available • Less integration concerns	• Maintenance required • Electrical Noise • Limited high speeds
Brushless Motor	• Higher efficiency • Long lifespan • High speed capability • Heat control	• Expensive • Complex

3.1.4 Motors Parts

3.1.4.1 36mm 12V 170 RPM Planetary Gearmotor

The 36mm 12V 170 RPM planetary gearmotor with encoder is a strong option for a mobile system like R.O.A.M. One of the biggest advantages is that it provides a good balance between speed and torque which makes it ideal for driving on outdoor surfaces like sidewalks or pavement. The planetary gearbox makes it more durable under load compared to spur gear designs and helps prevent gear stripping or mechanical failure during regular operation. Having an encoder already built in allows for position or speed feedback which can be used later for closed-loop control or odometry. The shaft is a standard D-shape which makes it easier to mount clamping hubs and attach wheels. Since the motor is compact it doesn't take up much space on the frame and still delivers enough power to move a mid-weight robot like R.O.A.M. across varied terrain.

There are a few drawbacks that should be considered before choosing this motor. It’s not a branded part which means the quality can vary between suppliers and there may be limited datasheets or support available. Even though it has an encoder the resolution is usually low which can make it harder to get accurate feedback at low speeds. The output shaft might require an adapter or custom hub depending on what wheels are used which adds some complexity to the mounting process. Because it’s a generic part there may also be inconsistencies in long-term durability or encoder reliability. Overall it’s a strong candidate for R.O.A.M. but care needs to be taken during sourcing and integration.

3.1.4.2 25D 12V 117 RPM High-Torque Spur Gearmotor

The 25D 12V 117 RPM high-torque gearmotor is a simple and affordable option that can work well in early versions of R.O.A.M. It provides decent torque through a compact spur gear system and has enough power to drive lightweight or medium-weight robots across flat or slightly uneven surfaces. Because it runs at 12V it fits well with the existing power system already planned for R.O.A.M. The low speed makes it easier to control movement without needing additional gearing and it's small enough to mount in tight spaces. It's also inexpensive which makes it a good choice for testing or early development where keeping costs low is important. For basic movement or proof-of-concept driving this motor can get the job done without much setup.

There are tradeoffs that come with using this motor. It doesn't come with an encoder which means there's no feedback for tracking speed or position which limits how well it can be controlled in more advanced systems. Since it uses a spur gearbox instead of a planetary one it's not as strong under load and the gears may wear faster if used heavily or on rougher surfaces. The shaft is smaller than some other options which may not match up with available hubs or wheels without adapters. The overall build is more basic and not ideal for long-term or outdoor use in demanding conditions. It can work for early tests or smaller robots but wouldn't be reliable as a long-term solution in the final version of R.O.A.M.

3.1.4.3 goBILDA 5202 Series Yellow Jacket 19.2:1 Gearmotor

The goBILDA 5202 Series Yellow Jacket gearmotor is a high-quality option that brings strong performance and reliability to a system like R.O.A.M. It comes with a built-in quadrature encoder and a well-machined planetary gearbox that handles torque and shock loads better than cheaper motors. The 19.2:1 gear ratio version offers a good speed for driving 6" wheels at around 0.5 to 0.7 meters per second, which is ideal for navigating sidewalks and similar terrain. The motor shaft uses an 8mm REX profile which matches goBILDA hubs and wheels without needing custom adapters. Since it's a branded part, there's detailed documentation, CAD files, and strong community support which makes it easier to design around and troubleshoot during integration. It's built to last and can handle long runtime and demanding conditions, which makes it a strong candidate for the final version of R.O.A.M.

The main downside of the goBILDA Yellow Jacket motor is the cost. It's more expensive than generic motors and adds up quickly if R.O.A.M. uses four-wheel drive. The 8mm REX shaft is nonstandard outside of the goBILDA ecosystem, so if you're using third-party parts or custom wheels it may require extra components or modifications. While it's compact for the performance it offers, it's still larger and heavier than micro motors, so it needs more space and stronger mounting. This motor is probably overkill for early testing.

3.1.4.4 Conclusion

All three motor options offer different strengths depending on the stage and needs of R.O.A.M. The 25D gearmotor is the most affordable and compact, making it useful for early development or lightweight test platforms, but it lacks feedback and long-term durability. The 36mm planetary gearmotor strikes a balance between performance and cost, offering solid torque, compact size, and an encoder for basic odometry or speed control. It's a strong choice for most builds that need reliable mobility without a high price tag. The goBILDA Yellow Jacket motor stands out in terms of quality, precision, and documentation, making it the most robust and feature-complete option. While it's more expensive and slightly bulkier, it is the best choice for a finalized version of R.O.A.M. that needs accurate control, smooth movement, and long-term durability on real terrain.

Table 3.4 — Motors

Motor	Pros	Cons
25D 12V 117 RPM Spur Gearmotor	• Compact and lightweight • Inexpensive • Easy to mount in small frames	• No encoder for feedback • Spur gears wear faster under load • Lower long-term durability • Less torque than planetary types
36mm 12V 170 RPM Planetary Gearmotor w/ Encoder	• Good balance of torque and speed • Planetary gearbox • Built-in encoder	• Generic • Encoder resolution is low • Shaft may require adapter for some hubs • Limited official documentation
goBILDA 5202 Yellow Jacket 19.2:1	• High-quality, branded motor • Built-in encoder • Strong planetary gearbox • Excellent documentation and support	• More expensive • 8mm REX shaft may not be compatible with non-goBILDA • Slightly larger and heavier • Possibly overbuilt for early prototypes

3.1.5 Motor Driver

R.O.A.M.'s drivetrain relies on two 12 V brushed DC motors that each draw up to 0.6 A peak. To command both speed and direction from 3.3 V logic lines on the ESP32-S3, we evaluated prebuilt dual-channel driver modules alongside a custom discrete MOSFET design. Our goal was to find a solution that could drive two motors without additional

level shifting, minimize conduction losses, and integrate cleanly into a single PCB footprint.

3.1.5.1 Part Comparison

To drive two 12 V, 600 mA peak brushed motors straight from our ESP32-S3's 3.3 V GPIO pins, we knew we needed an H-bridge that wouldn't force us into awkward level-shifting or waste half the motor voltage as heat. So, we kicked off by exploring the easy route—ready-made driver modules—and then looked under the hood at discrete MOSFET designs to see if we could squeeze out even better performance.

Our first stop was the classic L298N board. It's nearly impossible to find a hobby shop that doesn't carry one, and it'll happily channel up to 2 A per motor. But every time you switch directions, you bleed almost 2 V across its internal transistors—like pushing against a hill—and it only speaks 5 V logic. Next up was TI's DRV8833: compact, protected, and comfortable at 3.3 V on its inputs, yet it caps out at 10.8 V motor supply. On paper it should work, but in practice that leaves no breathing room on our 12 V rail. The TB6612FNG breakout quickly became the front-runner in the plug-and-play category: it takes up to 13.5 V, reads 3.3 V signals natively, and drops a mere few hundred millivolts when conducting. Its only caveat is a 1.2 A continuous rating, which sits close to our worst-case draw—so we'll need a solid thermal plan if we push it hard for extended periods.

While modules are great for rapid prototyping, we also toyed with all-MOSFET half-bridges for peak efficiency. IRF3205-based boards are beasts—they'll handle tens of amps at 36 V—but their gates need around 10 V to switch fully on, which our ESP32-S3 simply can't supply. That drove us toward a bootstrapped gate driver like the IR2104, paired with MOSFETs engineered to turn on solidly at 3.3 V gate drive. This combo lets us switch the full 12 V rail with almost zero loss—just a few millivolts across each FET—while still talking directly to our micro's logic pins. It does mean designing our own PCB footprint, placing bootstrap caps and gate resistors close to the chip, and carving out copper for heat dissipation. But the reward is rock-solid, high-efficiency motor control without compromises.

As another option we looked into the BTS7960 motor driver. This is a dev board that is built with two Bts7960 chips which support the bidirectional movements we needed. The devboard also come with a spot for your microcontroller as well as the voltage regulator. These components would be shifted for us in this project as we wont need to step our voltage down from the 12V as our motors can keep up with that. Additionally, we will just be incorporating this into our ESP32 design instead of using a separate microcontroller. This design will allow for a 6 to 27Vdc, this also supports 43A of peak current. This is important so that this motor controller does not get ruined by any stall current that may come from our motor. This design allows for simple development, integration, and support for our motors.

In the end, you can have a working demo in minutes by snapping in a TB6612FNG module on your breadboard. If you decide to chase every last drop of efficiency and

integrate everything into a sleek custom board, the IR2104 + low-gate-threshold MOSFET approach delivers the best results. Either way, your ESP32-S3 will drive two motors cleanly at 12 V, no level shifters or magic smoke required.

Table 3.5 — Motor Driver

Technology	Supply (V)	Logic In (V)	Continuous Current (A)	Voltage Drop per Leg	Integration Effort
L298N Breakout	5–35	5	2.0	~2.0 V	Plug-and-play, not 3.3 V compatible
DRV8833 Breakout	2.7–10.8	1.8–6	1.5	~0.3 V	Low effort, but supply is marginal
TB6612FNG Breakout	2.5–13.5	1.8–5	1.2	~0.2 V	Ideal for 3.3 V GPIO, thermal limits apply
IRF3205 Dual-MOSFET Board	3–36	Requires ≥10 V	10	~0.05 V	No PCB needed, but not logic-compatible
IR2104 MOSFETs (Custom PCB)	5–20	3.3–20	≥1.5	~0.025 V	Medium effort, full layout control
Bts7960	6-27	3.3-5	2	~.2	Easy layout to incorporate into this design

3.1.6 Chassis

For the body of our robot, we have discussed multiple materials that may be practical for our robots chassis. This is an important decision as the structural soundness will come from this body. It needs to protect the internal components, and provide a sound way to travel with these devices. There are three materials we found that were possible in our use: plastic, aluminum, and wood. All of these items would offer a way to transport the object identification hardware. The only restriction we have is our dimensions. Since we

wanted to make this robot roughly half the size of a standard sidewalk, this would be approximately 18 inches by 30 inches. The depth is not something we are restricting ourselves on as long as it is greater than 3 to 4 inches.

3.1.6.1 Plastic

Plastic was the first material we discussed. There were a few significant reasons why this material came up in our list of options because it is very versatile and cheap. We were thinking we could find a tub or a bucket that fit within the range of value we had preferred. This would run us under $15 at target or walmart. This type of material is lightweight and easy to drill and mount. We would be able to replace it easily if we damaged it, and there would be a lot of options for us to choose from. More than this, we would not have to spend many resources on actually designing and assembling a chassis as it would come preassembled. As this was the first option we discussed we also discussed the negatives that would come with these cheaper options. One of the negatives we knew we would experience is the fact that it would not be very strong. We hope to not encounter a situation where we run into objects or something bumps into our project, but this type of material is susceptible to breaking in a situation like that. Because it is plastic, fixing it in a way that looked aesthetically pleasing and proper would not be very likely. We also know that our devices have the chance of heating up or at least getting warm, being that it is also summer in Florida the heat around us as we store it for the next few months could cause quite a lot of warping on the chassis. This is not something we want especially because we will be mounting items to the body which could be affected by any warping. Overall, this option is not practical for the functionality we need it to provide.

3.1.6.2 Aluminum

Aluminum is the next material we had considered when designing this project, this material is strong and would not warp like the previous material. It offers a very supportive structure where we would be able to stably mount items onto the fixture without them falling off or getting damaged. This material would be purchased from an online vendor prebuilt. It would also mostly likely withstand getting hit or bumped without major damage. But the practical application about how we would assemble and mount these parts is where some of the concern lies. Drilling into aluminum is a lot more difficult than drilling into the other materials listed. Purchasing it online also means that if anything happened to it, if we damaged it in some way, it would be more difficult and expensive to order a replacement. The cost is also a concern as this material with shipping and the item alone would be more expensive than the other options we had. For the size we would need, the cost would not only be more, but it also would be the heaviest out of all of our options. Aluminum is more dense than plastic and even wood, this would put more strain on our motors which we do not want to have. The last concern we had was the conductivity of the item. While we plan to protect our devices and make sure they are secured and separate, we also cannot guarantee everything will work perfectly in the trial stages. We do not want to risk damaging our equipment by having something accidentally short and get damaged due to the aluminum. With that being said, aluminum is not the material we wanted to go with due to its restrictions.

3.1.6.3 Wood

Wood was the last material we thought of to employ in this project. Our project in conception started out as aluminum but as we discussed its feasibility, wood quickly rose to our top material. Wood is often used for projects such as this because it offers a lot of variety and flexibility. It is easily accessible at hardware stores, many places will even size and cut it for you which is a positive. Wood is a relatively sound material, it does not bend and flex as much as plastic, but it also is not impenetrable like aluminum. In Florida we do suffer from high levels of humidity which would need to be monitored. We considered covering it in a waterproof substance to prevent water damage. But it offers a lightweight and slightly more sound option than the others. Additionally, since it is more penetrable, drilling and mounting items would not be as impossible of a task as aluminum is. This type of material is also more customizable than aluminum and plastic. We are able to paint or etch it as we please. Wood proved to be the most versatile option with the most accessibility and cost-effectiveness.

3.1.6.4 Conclusion

Selecting a chassis material for this robot required us to think not only about the looks or physical application but the overall long-term durability and functionality. The plastic offered simplicity and low cost, but the issues we would face with added stress and heat did not seem proper for a long-term solution. Aluminum provided a strong structural, impact resistant material, but the high cost, weight, and conductivity risks made us not choose this option. The wood emerged as our most adaptable option allow sufficient strength with ease in modification, as well as a preferable cost and weight.

Table 3.6 — Chassis Material Comparison

Material	Pros	Cons
Plastic	● Light ● Cheap ● 3D printer compatible	● Sometimes too malleable ● Not structurally sound ● Low heat tolerance
Aluminum	● Structurally sound ● Strong ● Lightweight ● Aesthetically pleasing	● Most tools cannot cut through it ● Expensive
Wood	● Many density options ● Cheap ● Accessible	● Weaker than aluminum ● Warps with moisture ● Can be heavy

	• Easily customizable • Strong	

3.2 Path Obstruction and HMI Hardware

3.2.1 Obstacle Handling Method

There were a few different ways that we wanted to handle obstacles obstructing the path of R.O.A.M. They all came with challenges, but we wanted to find the option that was most achievable with all of the other subsystems we will be creating. One would be for the robot to traverse around the object, the next is for it to stop and return to where it came from, and the third would be to stop and alert the website. Each posed their own challenges and prospects so here will be the discussion of the options and which was chosen in the end.

3.2.1.1 Traverse Around

The first option was for us to traverse around the object in its path. This option was considered heavy as it would be the most practical for street use as this robot will encounter many obstacles as it drives autonomously. This would allow for the bot to drive without human intervention for a longer period and it would increase the coverage that we reach. The cons that come along with this idea are numerous as well, first being the complexity. Already, driving this robot autonomously and having it programmed to stay within the boundary of the sidewalks with the LiDAR sensors is quite a heavy task. To add a system that would have it be able to identify that something in its path was not traversable and then attempting to drive around it, while also staying on the sidewalk, seemed like another task that was too complex. This would require more real-time algorithms and resources. It would also mean that there would have to be significant resting to ensure accurate performance, or another thing that could fail in the presentation. This approach was technically feasible, but this system would warrant more resources and technology that we were not able to provide. This in the end is something we would implement only if we have the resources, which as of now we do not.

3.2.1.2 Return to Base

The following system proved to be more achievable but less impactful. This approach for an obstacle was to stop, turn around, and go back to its source. The source would most likely be its charging station. This would not be complex to implement, it removes the risk of veering off course to attempt to go around an obstacle, and it keeps the robot close to the charging station. On the other hand, this approach has by far the most cons. This approach allows for the least amount of operating time since it would have to backtrack completely. Making it back track completely would also bring up a sleuth of other problems and systems that would need to be created. Additionally, if it ran into another obstacle along the way then we would have to make another condition for this system.

This method did not seem the most efficient or practical for the goal that we were attempting to achieve.

3.2.1.3 Signal for Help

The final approach we came up with was what we found to be the most efficient and reasonable to complete. This is to have the robot stop upon meeting an obstacle that obstructs its path, and send out a few distress signals. The first signal we want is to send a message to our website, this will allow the operators or any person using the website to see it and assist the robot. The next signal we want is to have the robot emit a noise to get people's attention around it for possible human intervention. Lastly, we want to display a signal on the LCD screen. These last two are more a humorous way to allow the people around the robot to signal that it needs assistance. This option, while it does not have the least amount of human interaction or the most amount of coverage on the sidewalks, it is within the realistic bounds of what we expect to have time and resources for. This also add a bit of personality by attempting to interact with the people around it. The concept was something we adopted from a previously researched and talked about technology, Starship. In the Starship robot, if it gets tampered with or stuck, it will release a siren noise to warn people around it. This system we believe is the median of time roaming, human interaction, and complexity, which is why we chose to incorporate it.

3.2.1.4 Technology Selection

In the exploration of how to combat obstacles obstructing the object, there were a few technically feasible options which did not align with our resources or goals. While traversing around an object would be the most preferable with the limited human interaction, it is just not realistic with this timeline. This does present another level of its autonomous behavior and does give R.O.A.M. more time on the sidewalks, but the complexity is pretty unachievable at this moment. Returning to the base also offers a level of nonhuman interaction, but the backtracking is not an efficient solution. We chose the option of letting R.O.A.M. signal for help as it does not make the robot retrace its steps, and its complexity is absolutely achievable.

Table 3.7 — Obstacle Obstruction Solution

	Pro	Con
Traverse Around	● No human intervention ● More street time ● Solves issue	● Complex ● More resources required
Return to Base	● No human intervention	● Could get lost

	• Solves issue	• Complex • Unnecessary backtracking
Signal for Help	• Can continue on street • Solves issue • No unnecessary movement • Achievable (not complex)	• Human intervention

3.2.2 Noise Emission

The noise emission for this project is essential to communicate with individuals who interact with this robot in the wild. This is to give warnings or distress signals. The goal is to couple a noise being released with the LCD screen distress message to possibly convey that help is needed. There are three different devices to review that would satisfy that requirement: an audio player, a magnetic buzzer, a piezo buzzer.

3.2.2.1 Audio Player PCM5102

An audio player refers to a small module that stores and plays back audio files through a speaker. The goal of this would be that if the robot got stuck, it would release an alarm or a message asking for help. This would be the most clear way to ask for assistance, as there could be no ambiguity with the signal. We could have a message that clearly states instructions to help the robot. The files would be very customizable and fun to interact with. There also may be more volume potential with the robot, this system could be louder and get more attention than a buzzer could. The drawbacks that come with this are more substantial than our other options, unfortunately. While the audio player would be the most interesting to incorporate and interact with, realistically it adds more complexity to this system than we have the resources for. This system would require its own SD car, amplifier, and speaker. This would involve us needing to have digital to analog audio conversions which is designed to convert our audio files to something that can be heard around us. This devices sues 2V RMS to meet these audio output requirements. The additional equipment would not only be more complex to incorporate, but also adding more cost to the budget that we do not have room for. This device would also pull more energy to play the audio files, which could be an issue with the battery. With all of those issues there is also seen to be delays with audio files which we do not want. The total drawbacks in this solution has turned the distress sound solution away from audio files.

3.2.2.2 Magnetic Buzzer WST-1206UX

The next system is a magnetic buzzer, which is an electromechanical device that creates a buzzing noise when power is applied to a diaphragm. There is only one fixed tone with this device when supplying DC voltage. This is a very simple system, no extra devices needed, just DC voltage applied. This means that it would be very easy to integrate into

this project, allowing us to have this system and not spend too many resources on it. Being as simple as it is, it means it is also very affordable and reliable. There are a few issues that come with this device that create some issues. The first being the fixed tone that it has. It seems to produce in a small range and does not allow for different frequencies or patterns to customize the noise. It can work for up to 30V and has a frequency of 4kHz. This also has a small volume range and in the environments we want to use it, it may not be suitable. This is why we are moving away from this device, as it does not have enough variability to function how we want it to work.

3.2.2.3 Piezo Buzzer PS1240

The piezo buzzer is the last device that we want to mention. It is a ceramic device that vibrates when a voltage is applied, so it can generate a noise. There is variability in tones and frequencies that it is able to use. It can have different patterns and beeps to function as we want. Using PWM, we can generate different noises to allow for a different frequency. Additionally, we can have different patterns for different alarms. This is rated for 5V with a range of 3 to 7. Due to the simplistic technology, integrating it to the microcontroller will be relatively simple. This is also a low power option since it does not require as much energy as the audio player. It is also very light weight and can achieve higher volumes than the magnetic buzzer while still being very affordable. Because our website will be alerted and there will be a help signal on the LCD screen, we only truly need a simple alert. This buzzer will give us enough variation in the buzzing to allow for a simple alert without the added complexity of the extra equipment the audio player needs. Due to these reasons, we are using the piezo buzzer to create the noise to alert people near our robot.

3.2.2.4 Conclusion

In selecting an audible way to get the attention of people surrounding R.O.A.M. we thought about all the information that will go into the system, as well as how it would be interpreted by those around the robot. With this, the audio player seemed like a solid option with the most variability, but the complexity this system brings is more than is necessary for this project. With the magnetic buzzer, it is low cost and very low complexity. But that low complexity comes with more restrictions on tone and range. The bixxer is also not typically suitable for outdoor use. The Piezo buzzer seemed like the best out of the two options as it provided a balanced solution that had the range and tone while also being realistically achievable for this project.

Table 3.8 — Noise Emission Comparison

	Pro	Con
Audio Player PCM5102	● Versatile options ● Explicit spoken instructions	● Higher energy demand ● More system integration ● Additional hardware needed

Magnetic Buzzer WST-1206UX	• Easy to incorporate • Cost efficient • Simple design	• Limited tone • Limited volume • Not suitable for outdoors
Piezo Buzzer PS1240	• Sufficient sound • Adjustable volume • Cost efficient • Easily incorporated • Suitable for outdoors	• No versatile noise

3.2.3 Display Technology

For R.O.A.M. we want to have a screen on the robot, so there is a clear indication of functionality. This means we want the robot to display if it has found trash, treasure, if it needs help, or if it is in a state with none of those things. To do this, we need to find a screen that is suitable for the conditions that the robot is facing. It will be outside often, it will be a part of an intricate system, and much more. That being said we have three options to choose from, E-Ink, OLED, and LCD, we will discuss these at length below.

3.2.3.1 E-Ink

E-Ink is a display that can form images and text with black and white particles. It reflects ambient light and can make the image look like printed paper. They only absorb energy when the image changes, meaning they are relatively energy efficient. This means it could be beneficial to our project as we will be having to read the screen outside as well as worry about the energy usage. There are significant drawbacks with this product that led us to not use this device. One being the refresh rate is slower than the other devices on the market. This means any emergency or quick messages will take more time to update than we may want. There are also limitations with the colors available. While that is not functionally an issue, we want this robot to have character and be fun so we would like to incorporate colors to the screen.

3.2.3.2 OLED

The next screen we want to review is OLED. OLED screens are a type of display where each pixel is made to emit their own light as current passes through them. These screens are self-illuminating and their black screens are the darkest on the market. This also means that their color is incredibly vibrant, and the screens are incredibly thin. With these positive traits, we would be able to have a very colorful and bright device for our messages. These however come with a few drawbacks as well. There is a risk with OLED screens to have severe burn-in if we hold an image or message on the screen for too long,

which is very likely. These devices also tend to be more expensive since their picture is well-made.

3.2.3.3 LCD

The last option for screens we want to consider is an LCD screen. This is a type of screen that uses backlight to shine through crystals to create an image. These screens are practical for all conditions, sunlight, shade, and environment changes. The refresh on this kind of screen is very fast and can allow the robot to change messages quickly, as well as keep messages on the screen without risking burn-in. These devices are also incredibly affordable and often come in numerous sizes. These devices are also very available which will be helpful in case of any type of damage. With their availability, there is a multitude of resources available to troubleshoot with and learn from as well. For these reasons, we chose to go with the LCD screen.

3.2.3.4 Conclusion

Reviewing the details of each different part, they all offer different properties that could assist or hinder our project. The E-Ink presents disadvantages due to color and refresh rate, with the complexity of it being low. The OLED offers exceptional color quality, but offers concerns with burn in and higher costs. The LCD screen was chosen as the best option as it combines fast refresh rate with moderate quality and cost, also being adaptable in different environments. This is why we in the end chose to work with the LCD screen for our robot.

Table 3.9 — Screens

	Pro	Con
E-Ink	• Efficiency	• Slow refresh • Color limitations • Low brightness
OLED	• Vivid colors • Thin/Lightweight	• Image retention • Costly
LCD	• Fast refresh • Vivid colors • Affordable • Adaptable	• Not as compact as OLED • Not as vivid as OLED

3.2.4 Display Product

3.2.4.1 ILI9341 2.8" or 3.2" TFT LCD with SPI Interface

The ILI9341 TFT LCD is a strong choice for R.O.A.M. and can handle a lot of what the screen needs to do. It has a good size and shows color graphics so you can display things like battery, sensor values, lidar data and more. There's even a version with a touchscreen that lets you make a simple user interface for controls or manual override. The screen is bright and has enough resolution to show useful stuff without taking up too much power or space on the robot. The best part is it works really well with ESP32 and there's a lot of support for it especially with the TFT_eSPI library that has all kinds of functions already built in. Since it uses SPI it doesn't need a bunch of pins either which is nice when the ESP32 is already running other sensors and stuff. It's a reliable option that lets you add real interface elements to the robot.

Even with all the good things the ILI9341 screen has some downsides that might make it tricky to work with sometimes. The touch version can be a bit finicky and needs calibration and some extra wires for the touch controller. It's not super high resolution so if you try to draw too much at once it'll start to lag or flicker and eat up a lot of the ESP32's memory. Also the screen needs a regulator if the logic level isn't 3.3V which can make wiring messier. It's not waterproof either so using it outside means you gotta protect it from the weather. Some versions don't come with good documentation as well. So figuring out which pin is what can be difficult depending on where you bought it from.

3.2.4.2 ST7735 1.8" TFT LCD

The ST7735 1.8 inch TFT LCD is a compact and efficient display choice for use on R.O.A.M. It supports full color output with a 160 by 128 resolution which is enough to show sensor values, basic graphs, or system alerts without taking up much space. Its small size and low power use make it a good fit for embedded systems that need to save energy while still giving visual feedback. Since it uses the SPI interface it only needs a few pins on the ESP32 which is helpful when other hardware is also being used. The display is compatible with popular libraries like TFT_eSPI and Adafruit_ST7735 which makes it easier to program and customize with simple text or graphics.

While the ST7735 display has useful features there are some downsides that make it less ideal in some cases. The small screen can be hard to read from a distance and doesn't leave room for detailed layouts or large buttons. It doesn't have touch support so it can only be used to show information and not receive input from the user. The resolution is also lower than some other LCDs so complex visuals like camera feeds or lidar maps wouldn't look clear. Wiring can be difficult depending on the specific version since pin labels may not be standard or well-documented. Overall it's a lightweight and functional option but more limited in interaction.

3.2.4.3 ST7789 2.0" TFT LCD

The ST7789 2.0 inch TFT LCD is a suitable choice for R.O.A.M. when a compact display with moderate resolution is needed. It offers a 240 by 240 pixel layout, which provides enough clarity for displaying numerical data, sensor graphs, or system labels. The square format allows flexibility in mounting and layout design, which can be useful when space is limited on the chassis. It uses SPI communication, which reduces the number of required connections to the ESP32. Libraries such as TFT_eSPI and Adafruit_ST7789 support the display and make it possible to draw basic UI elements and visual feedback. The screen is small but can still be used for essential readouts in real-time operation.

The ST7789 lacks touch functionality in most versions, so it is limited to output only. The screen size restricts how much information can be shown at once and may require simplified layouts. Some modules omit features like a reset pin or level shifting, which can complicate wiring and require external components to match ESP32 voltage levels. Documentation quality varies across manufacturers, and some displays are not labeled consistently. While it works for compact visual output, it is not ideal for complex interfaces or interactive control.

3.2.4.4 Conclusion

All three displays offer different advantages depending on the needs of R.O.A.M. The ST7735 is the smallest and most lightweight, which makes it useful for minimal feedback. The ST7789 improves on resolution and clarity while keeping a small footprint, though it still lacks interaction features and requires more careful wiring. The ILI9341 employs a larger display area, color graphics, and available touchscreen versions. But the complexity of its design creates issues with integration as the available pins will not be able to compensate for this design. The ST7735 offers the most streamlined system with its available functions to allow us to complete the design needs for this system. This screen also will not be as power hungry as its competitors allowing for an efficient use of power. For a system like R.O.A.M. that benefits from both functionality and efficiency, the ST7789 is the most practical and adaptable choice.

Table 3.10 — Screen Part Selection

	Pros	Cons
ILI9341 (2.8"–3.2")	● Touchscreen support (in some versions) ● Higher resolution (320×240) ● Compatible with ESP32 and TFT_eSPI ● Larger display area for	● Larger physical footprint ● Higher power usage ● Touch requires calibration and extra wiring ● Needs protection from outdoor conditions

	data and control	
ST7735 (1.8")	• Very compact and lightweight • Low power consumption • Simple wiring with SPI • Supported by common graphics libraries	• No touchscreen support • Low resolution (160×128) • Small display limits layout complexity • Poor readability from distance
ST7789 (2.0")	• Higher resolution • Square layout offers flexible design • Good clarity for its size • SPI library support • Efficient design	• No touch functionality • Some modules lack reset or level shifting • Limited space for elements

3.3 Autonomous Movement Hardware

Autonomous sidewalk navigation demands precise environmental perception, and selecting between LiDAR and camera modalities defines the system's capabilities and constraints. R.O.A.M. employs a budget-oriented 2D LiDAR mounted at a 10–15° downward tilt to compensate for its lack of native 3D scanning, improving detection of subtle ground-plane features like cracks and expansion joints. Cameras capture data-rich RGB images for semantic understanding but rely on inference to infer spatial geometry. In R.O.A.M.'s context. where low-latency reaction, reliable outdoors performance, and minimal onboard compute are paramount, each method's trade-offs steer its role in the perception stack.

3.3.1 Perception Modalities

3.3.1.1 Spatial Geometry: Tilted-LiDAR Distance Measurement vs. Camera Depth Inference

By tilting the 2D LiDAR downward 10–15°, R.O.A.M. effectively projects its scan plane onto the sidewalk surface at a shallow angle, increasing point density along the ground ahead. This orientation aids in detecting longitudinal discontinuities such as cracks, joints, and slight height variations by creating voids or clusters in the point cloud that map directly to the physical gap. Unlike a flat-mounted LiDAR, the angled setup enhances spatial resolution in the forward path without requiring multi-beam hardware. The raw range returns thus form a quasi-2.5D profile of the walking surface, which RANSAC and sliding-window density filters can trace to extract a midline or edge path.

Camera systems, in contrast, must infer depth from monocular cues, stereo disparity, or learned depth models. These approaches introduce latency, rely on textured gradients, and degrade under uniform surfaces or uneven lighting—common on concrete sidewalks. Without direct distance measurement, vision-only methods can mistake cracks for shadows or fail to localize subtle elevation changes, undermining reliable boundary tracking.

3.3.1.2 Temporal Processing: LiDAR Scan Rate vs. Camera Frame Handling

The tilted 2D LiDAR maintains its nominal scan rate (6–12 Hz), delivering fresh angled-plane data every 80–160 ms. This steady cadence integrates predictably into R.O.A.M.'s control loop, enabling timely detection of path deviations along cracks or joints. Minimal pre-processing—filtering returns by height band and applying planar coordinate transforms to correct for tilt—allows CPU-only execution on low-power edge hardware.

Cameras can output 30–120 fps feeds, but translating 2D frames into spatially meaningful information requires color correction, resizing, segmentation or depth estimation passes, and object tracking. These layers increase per-frame latency and introduce jitter when lighting shifts or motion blur occurs. Synchronizing asynchronous frame analysis with the robot's kinematic update further compounds timing uncertainty.

3.3.1.3 Environmental Robustness: Angled-LiDAR Signal Reliability vs. Camera Sensitivity

Angled LiDAR scans leverage infrared returns that are insensitive to shadows, glare, or color variation. By focusing returns within a narrow vertical band (e.g., 0–20 cm above ground), R.O.A.M. filters out tall clutter, like bikes, poles, foliage, while retaining fine sidewalk features. Even under tree cover or dusk conditions, the LiDAR's tilt ensures consistent ground-plane intersection.

Camera vision is vulnerable to ambient light changes: deep shade, lens flare, and dynamic range extremes impair both object classification and depth inference. R.O.A.M.'s vision pipeline mitigates this with HDR capture and exposure control, but at the cost of additional compute and power. Misclassification of cracks vs. decorative seams, or shadows vs. obstacles, remains a risk without corroborating geometric data.

3.3.1.4 Integration Complexity: Tilt Compensation vs. Camera Calibration

Mounting a LiDAR at an offset angle requires a one-time calibration of its transform relative to the robot's base frame. In ROS, a static tf transform and simple linear correction (using laser geometry or PCL) reprojects tilted scans into a consistent ground-parallel 2D map. Beyond defining tilt and height parameters, no further per-environment tuning is needed.

Camera systems demand more extensive calibration: lens distortion models, focus setting, white-balance, and per-scene exposure profiles. Semantic segmentation or object

detection models require dataset curation to cover the full gamut of sidewalk conditions and perspectives.

3.3.1.5 Computer Requirements: Tilted-LiDAR Filtering vs. Camera AI Inference

Tilted LiDAR point clouds feed directly into geometric filters, RANSAC line fitting, point-density thresholding, and seam-tracking algorithms, executing in real time on CPU. The additional step of compensating for tilt (applying a rotation matrix to each scan) adds negligible overhead. The lean processing pipeline preserves battery life and avoids thermal throttling on edge computing devices like the Raspberry Pi 5.

By comparison, camera-based perception leans heavily on CNN inference for depth estimation or semantic segmentation, plus tracking algorithms (SORT, DeepSORT). These operations demand GPU cycles, raising power draw and thermal load. End-to-end vision pipelines thus risk saturating limited edge compute, incurring frame drops or dropped inferences at speed.

3.3.1.6 Use Case Optimization: Tilted-LiDAR for Routing, Camera for Semantics

In R.O.A.M., the tilted 2D LiDAR anchors the navigation stack: its angled scans reliably delineate the sidewalk's seam network, enabling low-latency path following and obstacle avoidance. By decoupling geometric routing from RGB-based classification, the robot maintains stable motion without overtaxing compute resources.

Cameras serve a complementary role, running a lightweight classification model to label detected items—trash or treasure—for the backend website. This division of labor ensures R.O.A.M. achieves both responsive, robust navigation on a tight budget and meaningful object recognition without compromising safety or system simplicity.

3.3.1.7 Conclusion

Through deliberate comparison of an angled 2D LiDAR and camera-based perception, R.O.A.M.'s design demonstrates that each modality excels at complementary tasks under the system's strict budget and compute constraints. By tilting a single-plane LiDAR 10–15° downward, we transform its horizontal sweep into a quasi-2.5D ground-plane scan capable of resolving subtle sidewalk seams and cracks with millimeter accuracy, minimal preprocessing, and predictable real-time performance. This affords robust boundary tracing and fast obstacle detection without complex neural inference or susceptibility to lighting variation.

Conversely, cameras supply the rich visual context needed for semantic labeling—distinguishing trash from treasure—through mature, lightweight classification models. Offloading these higher-level tasks to vision prevents overloading R.O.A.M.'s CPU-based navigation pipeline while still delivering meaningful object data to the web interface. By decoupling geometry from semantics, the robot achieves responsive, reliable navigation and actionable recognition in one integrated system. This architecture not only maximizes the value of each sensor within R.O.A.M.'s tight power, cost, and processing budget but also lays a flexible foundation for future enhancements—such as

stereo depth cues or additional tilt-mounted LiDAR layers—should performance requirements evolve.

Table 3.11 — LiDAR and Camera Weight-Decision Matrix

Criteria	Weight	LiDAR Score (1–5)	LiDAR Weighted	Camera Score (1–5)	Camera Weighted
Depth accuracy	0.25	5	1.25	2	0.5
Scan/update rate	0.15	4	0.6	3	0.45
Environmental robustness	0.2	5	1	2	0.4
Integration & calibration	0.1	5	0.5	3	0.3
Onboard processing load	0.15	5	0.75	2	0.3
Semantic/visual classification	0.15	1	0.15	5	0.75
Total	1		4.25		2.7

3.3.2 LiDAR Part Selection

Movement routing constitutes the core of R.O.A.M.'s autonomous navigation, relying on LiDAR sensors to delineate sidewalk boundaries and detect obstacles in real time. The fidelity of the resulting 2D point clouds directly sets the upper limit for path-planning accuracy: insufficient range or coarse angular resolution can cause misclassification of curbs, erratic steering corrections, and impaired obstacle avoidance. Without a dependable LiDAR scan, even the most robust control algorithms will fail, establishing the necessity of careful sensor selection.

R.O.A.M. traverses residential sidewalks at up to 10 mph, encountering irregular pavement, debris, occasional precipitation and dust, and random objects such as barriers, pedestrians or pets. Such dynamic conditions impose strict requirements on LiDAR performance: a minimum effective range of 10–20 m is preferred, in order to preview upcoming obstacles, angular resolution finer than 1° to resolve narrow features (e.g., sidewalk and curb edges), and scan rates above 10 Hz to ensure smooth, jitter-free trajectory corrections. The sensor must also deliver reliable multi-echo returns in

cluttered environments and maintain consistent operation under outdoor lighting and thermal variations.

Selecting a LiDAR for R.O.A.M., on a $100 budget, involves balancing these performance metrics against practical integration factors. A sensor that underperforms in any one dimension, like the range, resolution, or refresh rate, risks lowering SLAM accuracy, increasing energy consumption with unnecessary stopping, or even causing collisions.Conversely, over-specifying the sensor can increase costs, power draw and computational overhead without providing a greater value in a walking-speed scenario.

3.3.2.1 Slamtec RPLIDAR A1

The Slamtec RPLIDAR A1, at roughly $100 per unit, offers a maximum detection range of 12 m with 0.9° angular resolution and a 6–8 Hz scan frequency via a USB (5 V) interface. Its official ROS driver package and USB-powered operation facilitate rapid prototyping on R.O.A.M.'s Raspberry Pi 5 edge computer, requiring minimal wiring and configuration. With a power draw below 5 W and a mass under 200 g, it imposes little strain on the robot's battery and chassis. However, the coarse beam spacing and modest refresh rate limit its ability to resolve small debris or abrupt curb transitions, making it best suited for early proof-of-concept SLAM tests rather than mission-critical navigation in cluttered environments.

3.3.2.2 YDLIDAR X4

Priced at approximately $110, the YDLIDAR X4 doubles the frame rate of the A1 to 12 Hz while retaining a 1.0° resolution and a 10 m effective range over the same USB (5 V) connection. Community-maintained ROS wrappers streamline integration, and the higher temporal fidelity produces dense point clouds that smooth positional jitter at R.O.A.M. 's walking-speed of up to 10 mph. Its lower maximum range and unchanged angular granularity, however, mean it still struggles with fine curb details and distant object previewing. The X4 represents a practical mid-tier upgrade when modest gains in responsiveness are required without a major hardware redesign.

3.3.2.3 HLS-LFCD2

Available for under $80, the HLS-LFCD2 module achieves roughly 12 m of range, ~1.0° resolution and a 5–10 Hz refresh rate via TTL-level serial. Its minimal bill-of-materials cost and lightweight (≈100 g) make it attractive for multi-sensor experiments, sensor-fusion research and classroom demonstrations. Integration demands custom driver development and careful handling of variable return rates, as scan consistency fluctuates with ambient light and temperature. The lack of vendor support and formal documentation confines its role to preliminary hardware validation rather than production-grade sidewalk navigation.

3.3.2.4 Benewake TF02

At roughly $220, the Benewake TF02 delivers a 22 m maximum range, 0.5° angular resolution and a 20 Hz scan rate over a 3.3 V UART link. Official C++/Python SDKs and

a ROS package simplify firmware configuration and data parsing, offsetting the need for custom serial-parsing routines. Consuming under 10 W and weighing only 70 g, it satisfies R.O.A.M.'s power and payload constraints while yielding high-density point clouds that reliably delineate curb edges and early obstacles in outdoor conditions. The principal integration tasks involve wiring the TTL-level UART interface and updating firmware parameters to match the robot's mapping pipeline.

3.3.2.5 Yahboom LD06

With the highest price considered so far at $250, the Yahboom LD06 occupies the premium end of R.O.A.M.'s under-$400 budget, featuring a direct time-of-flight core that generates up to 4 500 points per second with ~0.08° horizontal resolution across a 12 m range. Data and motor-drive signals transmit over TTL serial and PWM lines, producing exceptionally dense point clouds capable of resolving fine pavement cracks, small debris and subtle curb transitions. With a 350 g mass and integrated motor assembly, it requires careful chassis mounting, vibration damping, balance and cable routing, and more complex electrical integration. Total power draw remains under 10 W and community-provided drivers ease software setup, making the LD06 ideal for applications demanding maximal spatial granularity at walking speeds.

3.3.2.6 Conclusion

The role for LiDAR in the R.O.A.M. project is to replace the need for a camera and additional processing power required for another AI model. This decision streamlines navigation, allowing R.O.A.M. to prioritize sidewalk tracking without burdening its onboard computational resources. Upon narrowing down models under an original budget of $100, the YDLIDAR X4 stands out as the most balanced and effective option for production-level deployment.

While the RPLIDAR A1 offers an extremely low price point and seamless ROS integration, its limited scan rate and angular granularity constrain its responsiveness. At high walking speeds, its 6–8 Hz refresh rate risks generating jittery or incomplete point clouds, which compromises SLAM performance and results in delayed or erratic steering corrections. Although the A1 is well-suited for controlled lab environments and early proof-of-concept tests, it struggles to keep pace with dynamic, real-world conditions where frequent corrections and curb detection are essential.

By contrast, the YDLIDAR X4 doubles the scan rate to 12 Hz, providing a noticeably smoother data stream with higher temporal resolution—crucial for tracking sidewalk contours and managing fast, reactive movements. The difference is especially clear in environments with uneven pavement or shifting lighting, where the X4 maintains scan integrity without requiring additional post-processing or filtering. Its community-maintained ROS drivers integrate easily with Raspberry Pi 5 devices, making it a drop-in upgrade with immediate performance benefits.

Spending an extra ~$10 over the A1 proves worthwhile in terms of trajectory fidelity, reduced path jitter, and more stable curb detection—all of which directly improve

R.O.A.M.'s navigation accuracy. The upgrade avoids the diminishing returns seen in higher-tier units like the Benewake TF02 and Yahboom LD06, which offer exceptional range and resolution but exceed both cost and integration complexity limits for this generation of the robot. The HLS-LFCD2, though even more affordable, is disqualified due to inconsistent scan performance and lack of official support, posing too much risk for field deployment.

Ultimately, the YDLIDAR X4 is chosen because it meets R.O.A.M.'s critical requirements for resolution, refresh rate, and system compatibility without overcomplicating the build or breaching budgetary boundaries. It reinforces the project's mission: delivering scalable, robust sidewalk navigation while minimizing computational and hardware strain.

Table 3.12 — Comparison Table for LiDAR

LiDAR Model	Pros	Cons	Cost
RPLIDAR A1	• Extremely affordable • Straightforward integration	• Coarse scan • Limited refresh rate	$100
YDLIDAR X4	• Higher scan frequency • Low integration effort	• Shorter range • Similar angular fidelity	$160
HLS-LFCD2	• Lowest price point • Ideal for testing hardware stacks	• Unstable data • Lacks official support	$80
Benewake TF02	• Best balance of range and detail for field trials	• Requires parser • UART wiring considerations	$220
Yahboom LD06	• High-density output • Stable DTOF performance	• Heavier • Specialized control circuitry needed	$250

3.4 Object Detection and Classification Hardware

3.4.1 Camera for Object Classification

Object classification is a central feature of R.O.A.M., R.O.A.M is designed to autonomously identify, classify, and report trash and reclaimable items left near sidewalks. This relies heavily on a key hardware component: the camera. The accuracy

and reliability of the entire object classification process depend on the quality of the visual data provided by this camera. Without a stable and effective camera feed, even the most advanced machine learning algorithms will struggle to produce useful results. In effect, the performance of the camera defines the ceiling for how well R.O.A.M. can fulfill its mission.

The classification software running on R.O.A.M. is designed to process real-time visual data, identifying objects such as furniture and general garbage. The robot moves continuously at a moderate speed, encountering a wide range of environmental conditions, from bright sunlight and shadows to motion-induced blur and varied object orientations. This dynamic environment places demanding constraints on the camera's capabilities. It must capture high-quality images rapidly and consistently, with minimal distortion, in order to provide reliable input for our object classification models. If the camera fails to deliver clear, analyzable images, the resulting classifications will be inaccurate or entirely absent. This hinders the core function of the system which is converting visual information into actionable data, that is then published to the website for public viewing and tracking.

A poorly chosen camera will not only reduce classification performance but will also create downstream issues that affect the overall system. Missed detections lead to missed opportunities for cleanup. False-positives waste time and damage efficiency in the system. Most concerning, a consistent stream of unusable or low-confidence data could render R.O.A.M. ineffective, even if the underlying software pipeline is robust. This makes careful camera selection not just important, but essential.

To evaluate the available camera options, this report considers a range of technical and practical criteria that affect performance in deployment. Each camera will be judged not only by its raw specifications but also by how those specifications translate to real-world performance within the constraints of R.O.A.M.’s specific use case. These cameras can provide trade-offs in the smallest ways, like prioritizing resolution over shutter type. This means we must examine each part of a camera and understand the overall value it can provide.

The main evaluation categories include shutter type, resolution, field of view, light sensitivity, and mountability. Each of these features has a measurable impact on detection accuracy, image clarity, and system responsiveness. There are two shutter types: Global shutter and Rolling shutter. Rolling shutter captures pixels one row at a time starting from the top and going down. This can capture distortion as it goes through the rows but the image changes before it can complete because of movement even if it is a small moment during slow movement like we will see on R.O.A.M. Global shutter captures all pixel rows of an image at once, eliminating the risk of having smeared images due to any motion because all the pixels are captured at once. For the purposes of R.O.A.M. a global shutter would be preferable. However, it's the product of complex circuitry so there is no way for it to be homemade, it must come with the camera. Resolution is the amount of pixels that make up one image usually depicted by the amount of rows and the amount of columns. This is important because the more rows and columns we can capture the more detail can be acquired and analyzed which will increase our object classification accuracy

and the distance at which we can classify objects. A higher resolution is preferable. It's difficult to know what field of view is good for the purposes of R.O.A.M. because knowing our FOV before we train our object classification model is important due to being able to populate the training database with objects of a similar FOV. However, research tells us the trade-off of varying FOVs. Having a high FOV will make further away objects appear smaller making them harder to detect and classify, while having an FOV that's too low will result in objects being missed entirely. Keeping an FOV of around 85 degrees will provide a decent balance of seeing objects far off to the side and objects further ahead. This camera will be primarily used outdoors at various times of day, so it is important to have a camera that can capture a range of varyingly lit scenes. Cameras use a combination of techniques to combat this. The combination can include better sensors or processing algorithms and is called HDR. HDR stops bright areas from being completely blown out and dark areas from being crushed which would make whatever is inside of those areas invisible to the camera. Having a camera with HDR support will increase image quality and guarantee that there aren't any missed items because of unevenly lit areas. A camera with HDR support is preferable for R.O.A.M. Mountability is another factor to consider since R.O.A.M. is a mobile robot. A camera that is easily mounted and maintained would be preferable so something small would work best.

3.4.1.1 Raspberry Pi Camera Module 3

The Raspberry Pi camera module 3 was the preliminary choice for R.O.A.M. because during early research, it consistently appeared due to its affordability and compatibility. This made it a decent starting point for prototyping and early integration. It eventually became the foundation for understanding the camera requirements of this project and served as a strong reference point when evaluating other contenders. This camera has a 12 MP resolution, allowing for more detail about objects that are far away. This increases R.O.A.M.’s ability to identify objects earlier, but also places greater reliance on its object tracking system. If an object is identified a certain distance away the object tracking will have to keep up with that object or else risk sending an abundance of objects to the website. Considering the camera’s autofocus functionality, detecting further away objects becomes an advantageous factor. Raspberry pi camera module 3 has HDR support. High Dynamic Range (HDR) is a common feature in modern cameras, yet it is absent from several other options under consideration. HDR helps a camera capture both bright and dark areas in the same image without losing details. This is important for outdoor settings that might include a shaded area under a tree and also a bright sun-lit road. R.O.A.M. will often find itself in these situations with unevenly-lit surroundings so that is a constraint we must consider. HDR helps greatly with managing dynamic environments without sacrificing efficiency. This camera uses a DOL-HDR that is a method that is favorable to R.O.A.M. ‘s conditions. It is a hardware-level solution that captures multiple exposures simultaneously and are digitally overlapped on the sensor itself, meaning no software involved. This method helps prevent motion blur and ghosting artifacts that often occur with traditional multi-exposure HDR techniques.

However, this camera also features a few drawbacks. This camera features a rolling shutter which captures an image line by line. Capturing pixels line by line starting from

the top and going down can capture distortion during movement even if it is a small moment during slow movement like we will see on R.O.A.M. This distortion can be mitigated by using a high shutter speed or a high frame rate. Raspberry Pi Camera Module 3 can only capture up to 30 fps because of the resolution, this is more than fine enough for our object classification software but isn't fast enough to fully negate rolling shutter. However, this camera is more than capable of having a shutter speed fast enough to make the motion negligible. Depending on manually changed settings shutter speed can be from one ten-thousandths to one twenty-thousandths of a second. Therefore, as long as the shutter speed remains high and R.O.A.M. maintains slow movement, the effects of the rolling shutter should be minimal. Another weakness of the Raspberry pi camera module 3 is the autofocus. Although autofocus can be advantageous in some scenarios, it also introduces several drawbacks. Autofocus is unpredictable and lacks the ability to precisely control it to optimize picture clarity and object classification accuracy. Autofocus may “lock-up” when trying to find a new thing to focus on which can hinder image clarity for significant periods of time. The work-around for this would be creating a custom autofocusing script and running it but not only is that time-consuming but it would be unnecessary with other options. Camera module 3 has a field of view of about 70 degrees. The ability to swap out and test various FOVs would be very valuable to maximizing identification range. However, the Camera Module 3 uses a fixed lens, meaning the field of view cannot be adjusted.

While this camera meets the current requirements for R.O.A.M.’s object classification system, there are still trade-offs to acknowledge. Rolling shutter limitations, while manageable, could become more noticeable if the robot’s speed increases or if sudden movements are introduced like going up a bump in the sidewalk. Autofocus unpredictability may also introduce occasional inconsistencies in image clarity, especially in dynamically changing environments. As the project evolves, future iterations may benefit from exploring alternative hardware platforms with support for global shutter sensors or swappable lenses, offering more flexibility and control. Although the Raspberry Pi Camera Module 3 provides a strong balance between performance, cost, and integration.

3.4.1.2 Arducam OV9782

The Arducam OV9782 has a few advantages that make it well suited for R.O.A.M. OV9782 has a global shutter which combats blurs and keeps objects in the images taken geometrically consistent while the robot is in motion. This improves image quality and reduces detection error that can be caused by movement. This camera also has a high frame rate of 120 frames per second. A higher frame rate will give the object classification and tracking algorithms more information to work with. The more information these algorithms have the higher their accuracy will be and the more reliable they become as algorithms. Object tracking will improve in efficiency with the more frames per second it’s provided. Due to the combination of global shutter, high framerate, and a lower resolution, OV9782 has a low latency architecture. This gives the algorithms the most up to date images consistently at the sacrifice of image quality. Additionally, its fixed-focus design avoids the unpredictability of autofocus systems, providing consistent focus and faster frame delivery without delays caused by lens adjustments. Also, OV9782

has the ability to easily interface with many edge devices giving it great compatibility and lowering overhead. OV9782 is light and small making it easy to fit into R.O.A.M.

OV9782 has its unique strengths but comes with some heavy disadvantages. OV9782's largest disadvantage is its resolution with only 1MP at 1280 x 800. If this was at least 3 MP there would be less cause for concern but 1MP decreases object detection distance and reliability significantly. Additionally, OV9782 does not have HDR support making it less effective in unevenly lit scenarios meaning a loss in the predictability of the quality and reliability of an image. The conjunction of these two features means we would have to be close to our objects to detect them and even if we were close the image can feature light-imbalances that make that image unusable or could even provide false-positives. As stated previously, there is no autofocus which could be a benefit to minimize unpredictability in images but sets the object detection to a specific range. This actually means we would want more reliable image quality which OV9782 doesn't provide. Lastly, OV9782 is not as well-documented as other camera options. There will be less to learn from and less community support if there are any major setbacks.

OV9782 has its main benefits in its low latency architecture. Providing the algorithms with as many images as they can process at a lower resolution, means the images will be processed very quickly as there's less total information. However, the sacrifice in image quality causes concerns as 1MP doesn't provide many details especially when we are detailing with smaller objects like trash. OV9782 is largely unpredictable with image quality which is only counteracted by the high frame rate and global shutter meaning OV9782 might capture a lot of images but will encounter periods of unusable images. It is also important to note that due to the low resolution and lack of autofocus, the images captured that aren't affected by the environment might not have enough information to be detected by the algorithm. OV9782 appears to be a camera better suited for situations where the environment is more controlled, that is not the case for R.O.A.M. The large unpredictability and lack of reliability are the main reasons OV9782 is not likely to be used on R.O.A.M.

3.4.1.3 Intel RealSense D435

The Intel RealSense D435 provides both RBG and IR data at the same time. RGB data will be our normal image data but the IR gives R.O.A.M. an extra layer of intelligence by providing information about depth. Having depth information opens up all new possibilities for R.O.A.M. that could increase efficiency and accuracy. R.O.A.M. could limit the range of object classification making it so the object detection algorithms only use the parts of an image that are less than some distance away, this could prevent R.O.A.M. from detecting and classifying objects across the street. Depth information can also allow for improved trash finding accuracy as the depth information can be used to find small objects hidden in grass. The Intel RealSense D435 has an FOV of 85 degrees which provides a decent view of the surroundings without losing a significant amount of object detection distance. This combined with the depth detection could increase our object classification rate significantly. Additionally, this camera has an acceptable resolution of 2MP at 1920 x 1080. While not as detailed as some other options this amount of detail should be just enough for object detection and classification range of at

least 2 meters. Images of this quality are recorded at 30 fps which is plenty for our algorithms to analyze, especially when paired with the depth data. Lastly, This camera is very well documented and accessible across many different OS making integration and experimentation easier

While the inclusion of IR data adds many possibilities, it's important to consider its applicability in all scenarios. If R.O.A.M. relies on IR data and it was a very bright day there would be a risk of degraded accuracy or completely false data in overexposed areas. Even if it only encountered a transparent or reflective object like a glass bottle or a sign the IR data can be incorrect and depending on its use hinder R.O.A.M. 's functionality. This limits our ability to rely on the IR data based on what we know about the environment and influences the method we would choose to implement the IR data to ones that could not yield much significance overall. Additionally, these methods would also add a lot of complexity to R.O.A.M. 's embedded software.The Intel RealSense D435 requires a USB plug-in which, while not necessarily a drawback, would require the extra securing of the USB port in use. The use of a USB port becomes more significant when you consider it draws more power than MIPI CSI that the other camera options use. The Intel RealSense D435 does not have HDR support and instead uses adaptive exposure which will automatically adjust exposure to fit the setting slowly. This does not account for the unevenly lit areas in the environment like a sunny sidewalk next to a shady tree.Additionally the Intel RealSense D435 is larger than the other camera options which would make it harder to mount and secure comfortably on our robot, especially with the motion, the part could become dislodged more easily than smaller components. Lastly, the camera is quite expensive at $315, which is significantly high.

The Intel RealSense D435 opens up many new possibilities for R.O.A.M. with the addition of IR data. However, the limitations and unpredictability of our environment limits the ways we can reliably use the IR data. In better environments we can use the IR data to run the object detection and classification algorithms only in areas with IR data resembling an object. This would significantly cut down on required processing power and time. However, this also greatly increases the complexity of the project. The increased complexity would increase experimentation and implementation time enough to draw concern about completing the project within the time frame. In conclusion the Intel RealSense D435 is a great option but it does not ensure a completed project within our timeframe.

3.4.1.4 Conclusion

The goal of R.O.A.M. is to identify and classify objects on the side of the road at an acceptable rate and accuracy. To ensure the completion of this goal, the best-fit option is the Raspberry pi camera module 3. This camera's main drawbacks are a rolling shutter and limited FOV but those problems can be mitigated. The other camera’s drawbacks are not as easy to compensate for and are unpredictable. The OV9782 could be a great choice for indoor usage but when used outdoors is too unpredictable and does provide enough image quality to compensate. The Intel RealSense D435 would be a great improvement on the Raspberry pi module 3 if given more time to experiment with. The Intel RealSense D435 might provide for a sufficient future upgrade with the price in mind. In conclusion, the Raspberry pi module 3 provides the best balance of features for our objectives.

Table 3.13 — Comparison Table for Camera

Camera	Pro	Con
Raspberry pi Camera Module 3	• 12MP high resolution • Built-in autofocus • HDR support • Easy Raspberry Pi integration • Low cost	• Rolling shutter distortion • Unpredictable Autofocus • Fixed FOV
Arducam OV9782	• Global shutter • High frame rate (120 fps) • Low latency • Low cost	• Low resolution (1MP) • No HDR support • Manual focus only • Limited documentation
Intel RealSense D435	• Depth + RGB imaging • Global shutter (RGB + IR) • Wide FOV (~85°) • Strong software support • Decent 2MP resolution	• No HDR support • Poor IR outdoors • Large and bulky • USB 3.0 power draw • High cost (~$300+)

3.5 Communication Hardware

3.5.1 Sim Module

Another important component for R.O.A.M. is how it sends data while driving outside. This project needs to have a reliable connection with sufficient bandwidth. This module will not only serve as the connection, but it will also have a GPS module to track coordinates and an antenna for the actual connection. When choosing the right module, another thing to keep in mind is the power usage, as we will not be hooked up to an unlimited power source. ROAM needs a compact, energy efficient We will also need a module that supports LTE CAT4 or higher, GPS functions, and basic protocols like HTTP, MQTT and TCP. The module must also be easily integratable with an esp32. Below are candidates for cellular modules that were looked into for R.O.A.M.S.'s prototype.

3.5.1.1 SIM7600G-H 4G HAT

This module is made to be a HAT for the raspberry pi 5. It combines LTE CAT4 (which is up to 150 mbps), GNSS, and serial communication into a dev board that is space efficient. The best part of this dev board is that it's plug and play, as the user just needs to send certain commands through UART to control the board functions. This board supports SMS, TCP/UDP, HTTP, FTP, and GNSS positioning. This makes it an ideal

component for R.O.A.M.. Another great component is all of the GPIO headers. This product also has great documentation, and a relatively low price.

3.5.1.2 SIM7000G

The SIM7000G is a lot more power efficient than its counterparts, and is designed specifically with IOT devices in mind. But although it is power efficient, it has a lot less bandwidth as its maximum output is only 375 kbps. It also does not support high speed image uploading. While it does have support for GNSS, this module is much better suited with static sensors.

3.5.1.3 A7670E

This is a newer LTE CAT1 module with integrated GNSS. It supports SMS, TCP/UDP, and HTTP at a slightly lower cost than the SIM7600G-H . The biggest drawback of this module is that it does not have the speeds that are required, as it can only go up to 10 Mbps which is a lot slower than the CAT4 module. This would severely limit the data transfer speed.

Table 3.14 — SIM Module Selection

Component	Pros	Cons
SIM7600G-H	● Fast upload speeds (150 Mbps) ● GNSS support ● Includes antennas and cables ● Official support	● Larger footprint than other boards ● Slightly higher power consumption
SIM7000G	● Energy efficient ● Built in GNSS	● Slow upload speeds (375 Kbps) ● Limited network availability
A7670E	● Cheaper than CAT4 modules ● Multiple interfaces (UART,USB)	● Slower upload rates (10 Mbps) ● Can be unreliable for multiple simultaneous protocols

3.6 Data Processing Hardware

3.6.1 Edge Device

The edge device for R.O.A.M. serves as the main computing device on the robot itself. This is where the object classification model and autonomous movement processing will be, meaning maximizing the resources available on the edge device is crucial to optimize performance. There are many edge devices available today, each offering a different set of features. This variety makes trade-offs almost inevitable. Selecting a device that meets R.O.A.M.'s performance requirements while remaining cost-effective is a significant challenge. Some key components to look at are CPU performance and RAM availability as these will impact overall performance the most.

3.6.1.1 Raspberry Pi 5

The raspberry pi 5 is a strong contender for R.O.A.M. ‘s edge device. The raspberry pi 5 has seen significant improvements from its previous generations upgrading to a 2.4 GHz quad-core processor which should be plenty of processing power for light-weight object classification models and autonomous movement calculations. Having a quad-core processor also allows for multithreading which can improve parallelizing the two functions. Multiple programs will also put a toll on memory, with options ranging from 1GB to 8GB of RAM, the Pi 5 can efficiently handle multiple tasks simultaneously without running the risk of memory bottlenecks, this is important for tasks such as computer vision processing, data storage, and LiDAR navigation logic. Raspberry pi 5’s upgraded I/O allows for the use of many methods for wireless communication and interfacing between components. These upgrades feature the inclusion of dual CSI camera interfaces which is especially valuable for R.O.A.M., enabling the use of multiple cameras for multi-angle detection setups to improve object classification reliability if one camera doesn't provide enough information. Additionally, the Raspberry pi 5 introduces PCIe 2.0 support, allowing for future hardware expansion such as NVMe storage or USB accelerators, making it a highly flexible edge device with room for upgrades if its processing power or storage is found insufficient for R.O.A.M. 's tasks. External expansions can also be added easily as the Raspberry Pi 5 has exterior parts like fans that are made to specifically go with the Pi and are available for a low-cost. The Pi 5 separates itself from previous editions by working with these active cooling components and in outdoor robots like R.O.A.M., active cooling is important to keep hardware healthy. The small form factor of the board also makes it ideal for mobile robots like R.O.A.M., where space is a limited resource. Finally, the Raspberry Pi benefits from extensive community support and high-quality documentation. This dramatically reduces development time and risk when integrating components or debugging.

Despite its many advantages, the Raspberry Pi 5 also presents several limitations that must be carefully considered for use in R.O.A.M. One of which is the lack of a built-in neural processing unit (NPU), which means all computer vision processing must be handled by the CPU, unless an external accelerator is added. This means if the CPU isn't strong enough, we could see a decrease in detection accuracy and other errors. The Raspberry pi 5 also generates much more heat than its previous editions because of the

quad-core processor improvement, but this can be mitigated by integrating active cooling methods. However, this introduces additional power draw and mechanical complexity, both of which can become critical concerns. It is important to note that the Raspberry pi 5 can exceed 10W of power consumption, which can put more strain on the battery system. The Raspberry Pi 5 does not include onboard storage, relying instead on microSD cards or external NVMe drives. While this allows for storage capacity flexibility, it also adds additional points of failure in harsh environments, especially from vibrations during movement that can wear down components over time. The absence of a real-time clock (RTC) can complicate accurate data timestamping, particularly when the robot operates disconnected from the internet, but this can be worked around easily as there are no foreseeable needs for time-stamping on R.O.A.M. Finally, the Pi 5 is a board not designed for rugged outdoor conditions. It lacks dust, moisture, and impact protection out of the box, this requires additional housing and insulation to avoid damage or otherwise system failure.

3.6.1.2 NVIDIAJetson Orin Nano

The NVIDIA Jetson Orin Nano Developer Kit is a widely used and supported edge device with a package that cannot be beat for the size and built-in tools. A model with which R.O.A.M. would work well with is the budget oriented Orin Nano model, which at $250 for the base chip with a developer board offers great value for desktop-class AI acceleration. A prime option they offer has a 8-core 1.7 GHz ARM processor with the NVIDIA Ampere-architecture on the GPU, capable of of 67 TOPS with 4GB of LPDDR5 memory onboard. A major draw here is the out-of-the-box readiness to integrate in any project, with 2 MIPI-CSI-2 camera lanes, gigabit ethernet capabilities, USB 3.1 and PCIe Gen3x4 expansion slots for peripherals or additional storage. NVIDIA’s JetPack SDK and TensorRT optimize common vision and neural-network frameworks for low-latency deployment.
JetPack and TensorRT enable rapid deployment of computer vision and deep learning frameworks, which is a major advantage for projects that require real-time inferencing and sensor fusion. In the context of R.O.A.M., this level of performance would allow for more complex models and higher frame rates, especially in future iterations involving detailed object segmentation or point cloud processing. However, the Orin Nano also introduces significant challenges. Its power draw ranges from 10 to 15 W during operation, which directly impacts battery design and runtime efficiency. Active cooling is required, increasing the mechanical footprint and complicating enclosure design. These factors must be considered carefully given R.O.A.M.'s target runtime and weight constraints.At a compelling price point, the hardware and software package are paralleled with their downsides.

Orin Nano’s performance headroom makes it ideal for future upgrades such as real-time 3D point-cloud segmentation or multimodal fusion of LiDAR and camera data. In R.O.A.M., it can sustain higher frame rates and more complex models without throttling. However, the trade-offs are its 10–15 W power draw, active cooling requirements and $250 price tag, all of which demand careful thermal design, power-rail filtering and budget planning to preserve R.O.A.M.’s 15-minute runtime goal.

3.6.1.3 Arduino Portenta H7

The Arduino Portenta H7 was considered for its lightweight form factor and strong support for real-time control, which makes it ideal for motor commands, emergency-stop functionality, and precise sensor integration. Its dual-core architecture, with one high-speed core handling timing-critical processes and another handling auxiliary tasks, offers enough flexibility to build interrupt-driven routines for LiDAR interfacing or PID-based motion stabilization. These kinds of tasks benefit from deterministic execution, which the Portenta is well equipped to handle.

With integrated Wi-Fi 6 and Bluetooth 5.0, telemetry and remote updates are possible without relying on external communication modules. This minimizes the overall footprint and keeps weight down. The Portenta also features an extended set of GPIO pins and supports camera interfacing, meaning it could handle basic image capture or sensor fusion pipelines. Its sub-watt power consumption during idle modes also makes it highly attractive when system efficiency is a top priority.

However, despite its strengths, the Portenta falls short in the domain of onboard inference. It lacks both a neural processing unit and a GPU, meaning any object classification, scene understanding, or segmentation tasks must be constrained to extremely lightweight models. While frameworks like TensorFlow Lite for Microcontrollers provide minimal ML support, performance is significantly restricted.

3.6.1.4 Conclusion

After reviewing multiple edge device candidates, the Raspberry Pi 5 paired with a Neural Processing Unit (NPU) has been selected as the main computing platform for R.O.A.M. This choice reflects a careful balance between computational capability, energy efficiency, cost, and physical integration constraints. While devices like the NVIDIA Jetson Orin Nano offer much higher AI throughput, their requirements for active cooling, higher power draw, and added chassis complexity place a strain on R.O.A.M.'s runtime and mechanical design. These drawbacks, coupled with a significantly higher price point, made the Jetson more suitable for larger or permanently powered systems.

The Arduino Portenta H7 presents an attractive option for low-power, real-time control, but its limited processing capabilities and lack of an onboard GPU or NPU make it unsuitable for handling vision or inference tasks at scale. Although it remains a strong candidate for future use as a dedicated motor or sensor controller, it is not equipped to serve as the main edge processor.

The Raspberry Pi 5, in contrast, offers an ideal middle ground. Its 2.4 GHz quad-core processor and up to 8 GB of RAM support concurrent navigation logic and sensor communication. The addition of an external NPU—such as the Coral Edge TPU or a compatible HAT accelerator—significantly boosts its capability to perform object classification and vision inference without burdening the CPU. This ensures responsive and accurate decision-making while maintaining energy efficiency. Dual CSI camera

support allows for multi-angle vision setups, and PCIe connectivity introduces optional expansion such as NVMe storage or additional accelerators.

Beyond its technical features, the Pi 5 benefits from a vast open-source ecosystem, robust documentation, and extensive community support, all of which contribute to faster development and reliable troubleshooting. Its small form factor and active cooling compatibility make it well suited to the spatial and thermal constraints of R.O.A.M.'s chassis.

Altogether, the Raspberry Pi 5 with NPU extension satisfies the project's performance needs while honoring design limitations related to weight, cost, and runtime. It supports current objectives while leaving room for future hardware upgrades if additional processing power or inference capability becomes necessary. This makes it the most scalable and practical choice for R.O.A.M.'s edge device.

Table 3.15 — Edge Devices at a Glance

Device	Pros	Cons
Raspberry PI 5 + NPU	● Widely available under $100 ● Versatile hardware and software options ● Camera ports	● Peak power nears the Orin Nano ● Relies on external storage ● Lacks OEM enclosure
NVIDIA Orin Nano	● Best in class for AI workload ● Camera ports ● Mature AI development sdk	● Large form factor ● Top performance requires cooling ● High power consumption ● Lacks OEM enclosure
Arduino Portenta H7	● Small form factor ● Incredibly efficient ● Surplus of GPIOs	● Requires host for complex AI models ● Low ram and clock speed ● No camera port

3.6.2 ESP 32 modules

One of the more important components is the microprocessor. This is what brings all the components together. In choosing the right MCU, we need to look for something that can communicate reliability and also is very power efficient since we are going to be running

on a battery. We need something that can receive commands from a raspberry pi, and send commands to a sim module and motor drivers. This makes UART communication, GPIO (PWM more specifically), and efficiency are priorities.

3.6.2.1 ESP32-S3-WROOM-U1-N16R8

This chip seems to go above and beyond to its counterparts as it can do heavy tasks and its about the size of a quarter. Its flash memory reaches 16 MB and has 36 GPIO pins making it easy to integrate since we need a lot of pins. It has three individual UART controllers, meaning we can easily use it to communicate with the sim module and raspberry pi module. It also have pins that can utilize PWM meaning we can control the motors and buzzer, and can still expand in the future. This chip also has native JTAG support so we can debug the code if we ever get into an issue. Although it does draw 50Ma more than some other chips mentioned, this amount is insignificant compared to what the raspberry pi will be using making this less of a thing to worry about. A great thing about this chip is that it can also go into deep sleep mode when not being commanded to do anything, making it super power efficient when sleeping. Another great thing about this chip is the support for it. What is meant by this is that arduino has so many libraries that can be used with the chip and many people have used this chip. This makes it easier to find examples to use for references when generating code and solving problems. One of the biggest advantages is that RTOS is supported. This means this controller can handle multi task applications. For example, it can handle data from the raspberry pi on one core while sending data to the sim module on the other.

3.6.2.2 MSP-EXP432P401R LaunchPad

This launchpad was made by texas instruments and is the bridge between their ultra low power processors and their higher performance processors. This launchpad is unique in that it mixes battery friendly current draw with modern 32 bit features. It uses the 48 MHz ARM cortex-M4F chip. THis chip has a built-in single precision floating point meaning it is a good choice to parse AT command strings that are sent, or running a corporate task scheduler. Where this microcontroller really shines is the peripherals that are at our disposal. This chip has four independent eUSCI UART ports, more than we need. This elements the need to use multiplexing serial data or fall back to bit banged software. This board also offers 40 GPIO pins so driving something like an LCD screen that needs 8 pins would be nothing. The biggest constraint for this chip is the memory size. It only has 64 KB of SRAM, and 256KB of flash. This means it will be a little cramped if we want to use RTOS and add multiple stacks.

3.6.2.3 Nordic nRF52840

This chip is by Raytac and was marketed as the first bluetooth 5 and thread radio powerhouse. It has a 64 MHz cortex-M4F core and a generous peripheral map. Its physical size is also small and about the size of a postage stamp. For a robot where energy is everything, this chip comes in as one of the best. While active, the current hovers around the teens of milliamps, and while in deep sleep, it can drop below a microamp. It has every communication protocol that is needed coming in with UART, SPI, I2C, and PWM pins. This chip houses forty eight GPIO pins providing room to add

peripherals in the future if needed. One downside is that it only has two UART pins to be used meaning there is no room for expanding if we want another one.

Table 3.16 — ESP32 Modules

Microcontroller	Pros	Cons
ESP32-S3-WROOM-U1-N16R8	• Three UART pins • 16 MB flash + 8 MB PSRAM = huge head-room • Massive Arduino/ESP-IDF ecosystem	• About 50 mA active, higher than ultra-low-power counter parts
MSP-EXP432P401R LaunchPad	• Three UART pins • Ultra-low-power modes	• Lower clock rate (48 MHz) • Low SRAM (64kB)
Nordic nRF52840	• Tiny current draw when in deep sleep (Micro amps) • BLE radio could be enabled later for field debug	• Only 2 UART pins

3.7 Power Hardware

In this section, we will review the equipment that we will need to power the rest of our project. Below is a table that describes the hardware that we will use as well as the voltage, current, and average power that will be required for this project. Later in chapter 6 we will also go through this topic more extensively and how they will individually be powered.

Table 3.17 — Power Distribution Chart

Components	Voltage	Current Normal	Current Max	Power Avg
ESP32	3.3V	.25A	.5A	~ .8W
SIM7600	3.8V	.48A	1A	~ 1.8W
LiDAR	5V	.48A	1A	~ 2.4W
RASPBERRY PI 5	5V	1A	3A	~ 5W
RASPERRY PI 5 CAMERA	5V	.6A	1.6A	~ 3W
MOTORS	12V	.5A	1.5A	~ 6W
LCD SCREEN	3.3V	.03A	.06A	~ .1W
BUZZER	3.3V	.02A	.04A	~ .07W
LEDs	2V	.01A	.02A	~.02W

3.7.1 Battery Technology Comparison

Before comparing batteries, there are a few technological specifications that need to be understood to see which batteries would best fit in this project. This battery will have to power a significant amount of devices in this project, these include: Raspberry Pi, ESP32, motors, LiDAR sensor, camera, LCD screen, piezo buzzer, and the cellular modem. Other than the motors, most of these devices only require 5V. Since R.O.A.M. has a significant amount of devices to move, and a lot of power needed for the more advanced edge devices, we want to make sure there are enough watt-hours. Assuming we want our robot to run for a minimum of 15 minutes, we have increased that amount for a buffer. With the peak total of all of our devices being approximately 60W, we would need about 30Wh. This is not a difficult thing to achieve in the motors below. The decision about which battery to get also affected my decision about the motors. When researching motors, we chose between 6v options, and a more powerful 12V - 24V option. We wanted to make sure that the battery and motor voltages were similar so that there did not need to be

much adjusting between the two. This would have involved adjusting the PWM in the motor driver to not deliver as much voltage to the motors. This was one factor in the decisions we were making. Additionally, since our devices mostly ranged from 3V to 5V we wanted to make sure we knew a way to step whatever voltage down from the battery to their specific needs, this also became a factor in my battery decision. If we chose a 12V battery, we would need to use a buck converter to take the voltage down from 12V to 5V for certain devices. The buck converter will be talked about thoroughly later, but the design seemed more complicated than a LDO regulator. For this reason, we had done much research into a 6V battery that would allow us to only need a LDO regulator. These were all factors into my decisions before we looked into the pros and cons of each type of battery pack and what they had to offer based on my specifications.

3.7.1.1 Lead-Acid Battery

The first battery type that was researched is the Lead-Acid battery. This is a type of rechargeable battery that has an ability to store electrical energy. It produces electricity through a chemical reaction that happens between the plates inside the battery. This technology is older, but it is reliable. It is a lower cost in comparison to other technologies and very easy to find. When it powers up a device, it typically sends a kind of surge to the system and it is recyclable. There are a few downsides to this type of battery though, it is bulky and heavy, which may not be as good since we do not want a lot of excess weight. It also has a limited cycle life, this means that this battery may need to get replaced earlier than our other options. This also means that to keep being able to use it we really should only discharge it about 50%. A few more issues that may come with this type of battery is the slow charge time, it can get damaged easily, and it overall has a lower capacity. For our robot, we not only need a reliable battery, but we need a battery that can be running for long periods and can be used to its full extent for long periods of time. This battery only partially meets these requirements and definitely does not meet the important requirements of life cycle, capacity, and weight.

3.7.1.2 Lithium

The next battery we considered was a Lithium battery, which is another type of rechargeable battery. This battery stores and releases energy by moving lithium ions between electrodes. The battery will discharge releasing this electricity into the device. This battery is significantly different from the Lead-Acid battery not only electrically but also physically. This device is significantly lighter than its competitor, by about 50% to 60% which is a very large margin. This means that our motors will not have to work as hard and it will take up less space. With Lithium batteries, the volume to energy trade off is also prevalent, since the batteries store more energy in a small area you are able to get more space. To add to that, the way that they are created allows it to last for a significant amount of cycles more than the Lead-Acid battery. The need for replacement and upkeep is much smaller with this. Many people may still reach for Lead-Acid batteries as their initial cost may be less, but with the maintenance and replacement rate the Lithium battery offers a cheaper long term cost. This battery, since it does not require cleaning terminals or equalizing charges, does not require nearly as much care and maintenance. Additionally, since it does not use acid or corrosive gasses, it is not only safer, but you do not have to worry about acid spills when it is jostled. We also will be able to see a higher

efficiency in charging and discharging, the Lithium battery charges much faster and loses less energy in heat compared to its competition. Overall, the positive attributes of this battery are prevalent and definitely worth the initial extra cost.

3.7.1.3 Lithium-Ion 6V & 12V

The last consideration about batteries that we had was about the voltage we wanted to use. As we came to the conclusion of using a Lithium-ion battery, we talked about what other things we have to take into consideration which is covered above. Our two options were to get a 6V battery, and a 12V battery. The difference between these two was only two things, with the 6V the motors we get may be less powerful, but we can save resources on using the LDO converter. With the 12V battery we could have a powerful motor but risk the difficulty of stepping down 7V or more for our devices. What made the decision in the end were the limitations on the specs we had. With a 6V battery, very often for the price range we are looking at, the amperage per hour is typically 6Ah. This is a huge limitation for our project. While the 6V does present a few good options as it is compact, and most of them have built in battery management systems. But further than that, at 6V with 6Ah that means we only have about 30Wh, this is less than an hour of runtime. To increase the amp hours we considered putting two 6V batteries in parallel so that we could continue to have the correct voltage and increase our runtime. This would double our runtime but still it is not as efficient as it could be. With this configuration we also would only be able to use 6V motors, if we used motors with a higher capacity it would not be working at its maximum efficiency. Additionally, this battery would be hazardous to the equipment that requires 5V. When using a regulator or buck converter it is usually recommended to have a little more headroom to be able to adjust for any voltage fluctuations that may bring the power coming in too low. If the power was too low, it could cause the devices to drop out. The Lithium battery could drop as low as 5.8 which could become an issue for some devices. For these reasons we shifted to looking at a 12V battery, this would allow us to avoid any issues caused by the voltage being too low and too close to the needed supplied voltage. With the batteries easily available online to get shipped, we also noticed that they were offering more amps per hour, this way we could avoid needing to stack two batteries in parallel. This would give us more hours in runtime, not needing to charge as often. Additionally, choosing to use the Lithium battery we found that the need to recharge before 50% decay was not necessary. This also meant that we could achieve more usable battery life for more cycle life as well. With a 12V battery, the options for motors with higher torque was also a positive that we considered. If we found a strong 12V motor we would be restricted in its capabilities due to the limited voltage, so a 12V battery fits the needs better. While this option was seen to be more expensive, the decision came down to the practical use we would get out of it, this being reliability, safety buffers, stronger motors, and more. For these reasons, we are choosing the 12V Lithium-ion battery with 10Ah to power this project.

3.7.1.4 Conclusion

Selecting a battery required significant comparison for this project, as without a battery there would not be a project. Lead-acid batteries often offer low initial cost, but their lifespan and maintenance requirements are not preferable. 6V lithium-ion batteries offer

more of a longer lifespan at a higher cost, but with a lower amp hour than the 12V option. The 12V lithium-ion battery proved to be the most preferred option because of its runtime, consistent voltage, compatibility with higher-torque motors, and much more. The reliability and functionality we would be receiving from this battery is worth the increased cost. The less amount of time needed between charges given the larger battery size also is a bonus that made us choose this battery. Overall, the 12V battery provides functionality that we would not receive with our other options, showing it to be the right type for our needs. Additionally, using a 6V battery in tandem for the components and keeping the 12V battery for the motors would allow us to have the maximum amount of power and efficiency for all of our peripherals.

Table 3.18 — Battery Comparison

Type	Pro	Con
Lead-Acid 6V	• Low Cost • Accessible • Recyclable	• Bulky • Limited life cycle • Limited deep discharge • Slow charging • Maintenance cost • Could spill acid
Lithium-Ion 6V	• Lightweight (lightest) • Fast charging • Full deep discharge • No maintenance • More life cycles • Efficient	• Limited Ah • Limited motor torque • No voltage buffer room • Expensive
Lithium-ion 12V and 6V	• Fast charging • Full deep discharge • No maintenance • More life cycles • Efficient • More Ah • No motor limits	• Expensive • Heavier than 6V

3.7.2 Battery Part Selection

In the R.O.A.M. system, separating the power supply into a 12V battery for the motors and a 6V battery for the electronics is essential for both performance and protection. Motors typically draw large amounts of current and can generate electrical noise, voltage

dips, or spikes especially during startup or under heavy load. Supplying sensitive components like microcontrollers or sensors that rely on more precise power from the same power rail can cause malfunctions or even permanent damage. By using a dedicated 6V battery for the other components, R.O.A.M. ensures stable and clean power delivery to its control systems, while allowing the motors to operate independently without affecting critical logic functions. This separation also simplifies power management, improves system reliability, and allows each subsystem to be optimized for its specific voltage and current needs.

3.7.2.1 6V Battery

The 6V battery plays an essential role in the R.O.A.M. system by powering the robot's control electronics and support components. It provides a stable and isolated power source for sensitive devices such as microcontrollers, sensors, and communication modules. By separating the low-power electronics from the high-current demands of the drivetrain, the 6V battery helps ensure reliable performance, reduces electrical interference, and protects critical components from voltage fluctuations.

3.7.2.1.1 Tenergy 6 V 2000 mAh NiMH Pack

The Tenergy 6V 2000mAh NiMH battery has some advantages that make it useful for small electronics systems. One of the main benefits is safety since NiMH chemistry doesn't have the risks that come with lithium batteries. It's easy to charge and doesn't need a protection circuit, making it simple to work with. The battery is also reliable in basic uses where current draw is low like powering sensors or microcontrollers. It can be used with affordable chargers and doesn't require custom charging routines or power management circuits. Since NiMH is widely used the battery is easy to find and replace. For smaller robots or secondary systems that don't draw a lot of current it provides enough runtime to be practical.

There are also drawbacks to the Tenergy 6V 2000mAh NiMH battery that affect its performance. The low energy density means it has a larger size compared to lithium batteries with the same capacity and it doesn't last as long on a charge. When current draw increases the battery voltage drops quickly which makes it less useful for electronics that need a steady and clean voltage like a Raspberry Pi or a LiDAR sensor. It also doesn't work as well in cold temperatures and has a shorter cycle life than lithium-based batteries. The battery doesn't have a regulated output so voltage levels can vary and that can create problems if the system expects a stable voltage. Because of these limitations this battery is not ideal for high performance or high power electronics.

3.7.2.1.2 TalentCell 6V 6 Ah $LiFePO_4$ and 12V 12Ah Battery Pack

The TalentCell 6V 6Ah LiFePO4 battery pack has several advantages that make it a strong option for powering control electronics in R.O.A.M. One of the most important benefits is the regulated 6V output which helps avoid voltage drops that could affect performance. It includes built-in protection so there's no need for extra circuits and it's safe to use around sensitive equipment like microcontrollers, sensors, and single-board

computers. The battery is also compact and lightweight making it easy to mount on a mobile system. It comes with a standard DC barrel jack that fits many components without extra wiring or soldering. The chemistry of the battery allows for more charge cycles than other types and it holds a steady voltage during use which is good for electronics that can't handle voltage swings. This design is also incorporated into the 12V version with additional amp hours given that it has double the amount of ability.

On the other hand, the TalentCell 6V 6Ah LiFePO4 battery pack also has some limitations. It can only supply a limited amount of current which means it could become unreliable if the electronic load increases beyond what it can support. If the current draw goes too high the battery's protection circuit can cut off power which could cause unexpected shutdowns. The battery should only be used for low current electronics and not as a general-purpose power supply for high-load devices. While it is convenient the fixed output voltage also means it's not as flexible for systems that require a different voltage. Even though it's LiFePO4 and safer than lithium-ion it still needs to be charged with a proper charger or it can become damaged. Overall it's a reliable choice for powering low-power control components in R.O.A.M. and with it in tandem to the 12V and 12Ah version there is enough power between the two batteries to provide all the power needs necessary for this robot.

3.7.2.1.3 Bioenno 6 V 12 Ah $LiFePO_4$ Battery Pack

The Bioenno 6V 12Ah $LiFePO_4$ battery pack offers several advantages that make it a strong power source for the control electronics in R.O.A.M. One of the main benefits is its large capacity which allows it to run devices like microcontrollers, sensors, and single-board computers for long periods without needing to recharge. It uses $LiFePO_4$ chemistry which is known for being stable and long-lasting with a high number of charge cycles. The voltage remains steady during use which is important for electronic components that require consistent power. The battery also has built-in protection features that help prevent overcharging, short circuits, and other electrical problems that could damage equipment. Its durable construction and reliable performance make it a good option for outdoor or mobile systems like R.O.A.M.

Even though the Bioenno 6V 12Ah $LiFePO_4$ battery pack has strong performance there are a few drawbacks to consider. Its larger capacity makes it heavier and bulkier than smaller battery packs which can make mounting and placement more difficult on compact systems. It also costs more than lower-capacity batteries and might be more than what is needed depending on how long the electronics in R.O.A.M. need to run. The battery requires a specific charger designed for $LiFePO_4$ chemistry and using the wrong charger can damage it. Because it is designed for 6V output it cannot be easily used in systems that need a different voltage without an additional converter. Overall it is a reliable and high-capacity choice for R.O.A.M. when a long runtime is needed and space and weight are not major concerns.

3.7.2.1.4 Conclusion

When comparing the three 6V battery options for use in R.O.A.M., each one offers different advantages based on system needs. The Tenergy 6V 2000mAh NiMH pack is the most affordable and easiest to manage but its low capacity and voltage drop under load make it unreliable for powering multiple electronics over longer periods. The Bioenno 6V 12Ah $LiFePO_4$ pack provides a good balance between size, safety, and ease of integration. Its regulated 6V output and built-in protection make it a plug-and-play option that fits well within the power needs of R.O.A.M.'s electronics. The TalentCell 6V 6Ah $LiFePO_4$ pack stands out for its high capacity and long runtime, making it ideal for extended operation but it comes with increased size, weight, and cost. For R.O.A.M., where stability, ease of use, and efficiency are important but size and weight are also concerns, the TalentCell 6V 6Ah $LiFePO_4$ battery is the best choice. It offers enough capacity for reliable operation of the control electronics while remaining compact and straightforward to integrate into the system. This was purchased along side with the 12V and 12Ah version as those will provide more than enough power to our motors with all of the positives that come with the 6V version.

Table 3.19 — 6V Battery Options

Battery	Pros	Cons
TalentCell 6V 6Ah $LiFePO_4$	• Regulated 6V output • Lightweight and compact • Built-in protection • Plug-and-play wiring • Good cycle life	• Limited current output • Not ideal for high draw setups • Requires correct $LiFePO_4$ charger
Bioenno 6V 12Ah $LiFePO_4$	• Long runtime • Very stable voltage • Built-in BMS for protection • Good for extended field use • Safe and durable	• More expensive than others • Large and heavy • Requires $LiFePO_4$-compatible charger
Tenergy 6V 2000mAh NiMH	• Very safe chemistry • Easy to charge • Inexpensive • Good for basic, low-draw electronics	• Short runtime • Voltage drops under load • Not suitable for Pi + LiDAR • Bulky for the capacity

Table 3.20 — 12V Battery Options

Battery	Pros	Cons
Bioenno 12V 12Ah $LiFePO_4$	• Steady voltage under load • Built-in BMS for protection • Long cycle life • Lightweight and compact • Reliable for mobile use	• More expensive • Needs $LiFePO_4$-specific charger • Heavier than smallest $LiFePO_4$ packs
LiTime 12V 12Ah $LiFePO_4$	• Very lightweight • Compact size • Affordable • Built-in protection	• Lower current output • May shut off under heavy motor load • Shorter runtime than higher-capacity options
Mighty Max 12V 12Ah SLA	• High capacity • Inexpensive • Easy to find • Simple to charge with basic charger	• Heavy and bulky • Voltage drops under load • Shorter lifespan • Poor energy efficiency for mobile use

3.7.3 Converter Technology Comparison

R.O.A.M is a project that has a lot of equipment that works together to fulfil the objectives and goals that we have written. To be able to use all the equipment, they will be powered by a 12V battery, this will be the power source for the entire robot. It will power everything from the motors to the ESP32. With that being said, there is a vast range of voltages required for these devices, this is anywhere from 3.3V to 12V. Because of this, there is a need for voltage regulators so that we can safely and efficiently drop the voltage down to a safe level, so the devices are not burnt. The options that were found to be suitable to discuss are the LDO converter, the SEPIC converter, and the buck converter. All of these regulators have their own strengths and drawbacks, which incorporate into why we did or did not choose certain ones. Certain things that we had to keep in mind during this are the different ranges, the efficiency of the converters, and the difficulty of incorporating them.

3.7.3.1 LDO Regulator

LDO regulators are low dropout regulators, which are also called linear regulators. They allow a clean and stable voltage drop. Typically, these types of regulators operate at low

potential differences, this range is about 300mV to 1V. How it works is typically a voltage is input to the system, unregulated, and it is sent through a pass element. A pass element is a component that will stabilize the output voltage from an input. It often uses a transistor to control the voltage drop and ensures the voltage stays stable. Typically, the output voltage is checked by a feedback network, and it can compare this to a stable known voltage. The LDO acts similarly to a resistor between the input voltage and the output voltage, where it will drop a certain amount of voltage based on how much you set it to be. This means that from your battery, this can allow whatever voltage to be supplied to your components without damage. This also means that if the voltage drop is greater than 1V, you will typically see an increased amount of energy go to heat instead of dropping the voltage. When you lose a lot of energy to heat, it increases the inefficiency of the system and puts strain on the equipment. While this design is relatively simple and it produces low-noise outputs, our system has more than a few components where the voltage potential is larger than 5V meaning that this type of converter would not be efficient for this system.

3.7.3.2 SEPIC Converter

As for a SEPIC converter, or a single-ended primary inductance converter, there are a different set of pros and cons. This converter is a DC-DC converter that can step up or step down voltage and continue to maintain the same voltage polarity. This system depends on the energy transfer between components which provide intermediate energy storage. The components can transfer the input energy to the output, this output is controlled by a switching component. The output voltage is regulated due to the energy transfer and based on the duty cycle it can be to step it up or step it down. This converter is relatively versatile and low-cost. The SEPIC converter also has minimal input and output ripple with a flexible range. Since it is able to work as a step-up or step-down converter, it is very flexible. There are a few downsides to this type of converter since it does have the ability to function so differently. The design for this component would be extensive. Since it needs indicators and capacitors to do the energy transfer you will be expected to have that extra layer of complexity. The device also faces a similar issue as the LDO regulator, with energy lost in heat. The inefficiency is not as intense as the LDO but it is not as efficient as the Buck converter has seen to work.

3.7.3.3 Buck Converter

The buck converter seemed to come up the most when researching how to regulate a voltage difference as large as 7V. This converter is typically a step-down converter that can reduce the voltage of an input lower. It works by rapidly switching a transistor between on and off to connect and disconnect the input to the rest of the circuit. This is usually done with a MOSFET and then controlled with a PWM signal. This signal determines the duty cycle and output. In this system, there is typically a capacitor that filters the voltage and provides a stable output voltage. To use a buck converter, there is a voltage range that is needed to work as a buffer for the voltage and desired voltage so that there is no chance of a sag. These need a range of .5-1.5V to be sure there is enough voltage after it drops it down. In the current system we have, there is an input of 12V, other than the motors the highest voltage we need to drop to is 5V. When deciding on what battery to get, we chose the 12V being the safer option as it gave that buffer space

for the buck converter where the 6V battery did not allow for that. This buck converter is the most efficient option we found for a regulator with the voltage potential the size that it was. This option can deliver a higher output current without much heat being created, unlike the other options. It will help the system to conserve quite a bit of energy. Drawbacks are still present in this option, these being that they can face noise or rippling and possibly create electromagnetic interference. While these cons are present, the efficiency does prove to be more impactful to our project, which is why we chose the buck converters for our equipment.

3.7.3.4 Conclusion

Voltage regulators and converters will be very important for this project, as we are using a 12V battery for components that range around 5V. To determine what kind of regulator that we wanted to incorporate, it was important to evaluate any tradeoffs among the options such as efficiency, complexity, and compatibility. The LDO regulators offered a low-noise, simple solution, but because of their lower voltage requirements, our design was not suitable for them. The SEPIC converter would have allowed for both a step-up and step down voltage potential, the complexity of integrating this converter was disproportionate to its need. The buck converter, however, offered a medium level of complexity that would create a more efficient performance and support higher outputs to minimize heat generation. This buck converter was the best option to step our voltage down for this project.

Table 3.21 — Converter Tech Comparison

Type	Pro	Con
LDO Regulator	• Low complexity • Low noise	• Inefficient • Limited in current • Not efficient > ~1V
SEPIC Converter	• Flexible • Non-inverting polarity	• Mildly inefficient • More complex • More ripple than LDO
Buck Converter	• Efficient • More variability • Efficient for > ~1V	• Noise • Ripple • Slight complexity

3.7.4 LDO Regulator Selection

After deciding that we would choose a linear regulator, the next decision we had was to make was what kind of regulator chip I would use. Many factors came into this decision

such as their voltage and amp max, their switching frequency, efficiency, availability, ease of use, and much more. Using the Texas Instruments circuit design website, I input what our voltage and amps were to start, and what we wanted in the end. This gave us a variety of chips to choose from, which we will explore in the next sections.

3.7.4.1 MP1584

The first regulator chip we decided to look into was the MP1584. This was available from one of the circuits from the webench website. It has an adjustable switching frequency of 100kHz to 1.5MHz with a voltage range of 4.5 to 28V and an up to 2A of continuous amperage due to thermal constraints. Most of this was within the range that we needed except for the current. 2A may possibly be too little as the Raspberry Pi could require up to 3A. Overall, this would work but it is not as efficient and has many restrictions due to thermal limiting.

3.7.4.2 TPS5430

This chip had a switching voltage of 570kHz which is efficient, with a good current allowance, but the voltage did not go as low as we needed. It also showed to be a bit more expensive. This came up in the list of our chips as there is a regulator that needed to be within the higher portion of this range but this chip did not work for all of our parts. We had hoped that we could just get a few of the chips in bulk instead of multiple different ones. For this reason this chip was not a match for this project.

3.7.4.3 LM2576

This chip came up multiple times for the different requirements that we need, and the circuit that they provided were found to have pretty good thermal efficiency. While the switching frequency was lower out of the options being at 52kHz, it did not take too much away from the positives of it. There is much versatility in this chip, the way it is set up with the circuit it suggests, allows you to have an adjustable output. Based off of the resistors you use for the feedback, you can specify what the output should be. This allows for a voltage range of 4V to 40V with a current limit of 3A which with all of those factors it is perfect for our power supply.

3.7.4.4 Conclusion

In the end we decided to choose the LM2576, not only did it have the correct voltage output that we needed for our multiple different needs, but it also had accurate circuits to incorporate. This chip additionally was recommended by many individuals who have used it in similar situations, which was helpful in making this decision. In later chapters, we will go into depth on this chip and how it will be applied in our design.

Table 3.22 — Converter Part Comparison

Type	Pro	Con

MP1584	• Fast switching freq • 4V-28V	• 2A • Thermal efficiency
TPS5430	• 3A • Moderate Efficiency	• Higher voltage range
LM2576	• 3A • 4V-40V • Variability	• Slower switching freq

3.8 Communication Protocol

3.8.1 Wireless Data Transmission Technologies

A big part of the project is how we are going to send the data. This includes things like the coordinate points, results from the machine learning models about trash or treasure, and possibly pictures too. Since the main idea is going around neighborhood sidewalks, it needs a communication method that is able to stay connected through suburban and urban environments. After exploring options, cellular data and wifi based communication were eventually landed on. These two differ on cost, coverage, infrastructure requirements, and reliability. This comparison is going to be based on which technology is better suited for R.O.A.M.'s need for real time communication and performance with unpredictable terrain.

3.8.1.1 Cellular Communication

Cellular communication is the use of mobile networks such as 4G LTE towers, which transmit data over wide areas. A big part of cellular is how mobile it is, as its primary use is for roaming devices like smartphones. This part makes it a good fit for autonomous robots that move through neighborhoods. Another advantage is that most neighborhoods already have a strong cellular infrastructure, meaning if a cellular connection was used, it could be a good technology for data transmission. In R.O.A.M.'s case, this allows for constant connection regardless of the location, in turn allowing it to navigate neighborhoods without the fear of losing connection. Cellular communication also supports location in the technology, meaning this technology would further give more information that would be useful.

While there are a lot of good with cellular, there also are some downsides. One of the biggest is the recurring costs with the data plan and the sim activation fee. This amount needed to pay would depend on how much data is being sent. This means it is not a viable option if we were to have a constant video or have constant photos being uploaded in high definition. Cellular communication can also take up a lot of power, especially places where there are weak signals where the module has to up the transmission strength to keep its connection, which can shorten the battery life. Another possible con is if there

is a lot of people in the area using the cellular tower, this can cause congestion meaning sending data can be slower and could possibly impact real time responses.

3.8.1.2 Satellite Wifi Communication

Wifi communication is good in that it can offer a much higher bandwidth, with lower latency given a network is available. Wifi is usually a better technology in environments with existing infrastructures, such as houses, parks, airports, and campuses. In these environments, wifi provides faster transmission speeds, which can be helpful if a lot of data is needed to be transmitted. Wifi also eats up less power than cellular data, which makes it more energy efficient when accessing. Another great thing about wifi is that it usually uses open standards meaning it can be easier to use different protocols.

One big drawback of wifi is its limited range. Wifi is usually only effective within 100-200 feet, and can be greatly reduced if obstacles are in the way such as walls, buildings, or trees. This does not make it ideal for R.O.A.M. who has to go around neighborhoods. One workaround is the invention of satellite based wifi. Satellite based wifi would be a better option as it does not depend on stationary objects like cell towers. Satellite wifi connections local networks with a larger core network which allows for data to flow efficiently. R.O.A.M. would need to have a satellite dish or be near a mobile satellite hotspot to work.

Although it works in theory, it also adds a lot of other challenges. One of the biggest being the equipment. The equipment needed is power hungry and weighs a lot, which is completely the opposite values of what R.O.A.M. needs to be successful. Adding this hardware onto R.O.A.M. would make the project more complex in that it would add weight which could not only affect the balance but also call for the need of larger motors meaning even more power would be used.

Another large concern is the latency. Even with the invention of low earth satellites, the latency ranges from 20 to 100 milliseconds under the best conditions. This can be a lot higher if the path gets blocked which could happen a lot since R.O.A.M. will be going under trees that are over sidewalks. Traditional geo satellites have latency that is up to 600 milliseconds which could harm real-time communication. A caveat that satellite wifi holds is that it is very weather dependent. This means that rain, excessive clouds, or even if there are trees blocking the signal can cause a disruption for the signal. This can cause small outages leaving R.O.A.M. without internet connection. This is not ideal since R.O.A.M.’s whole idea is to be outdoors, and even less since Florida rains often.

3.8.1.3 Wired Wifi connection

A wired wifi is something that needs a physical wire to be connected like how the modem is connected with an ethernet cable. This is a common connection in places like homes and offices, where the device being used is relatively close to the network equipment. Wired connections are super stable, and offer high bandwidth but there are some reasons why we wouldn’t aim to use it.

For this project, it is really impractical. This is because R.O.A.M. needs to be able to drive around neighborhoods and since it would need to be physically tethered to an

access point this would not be feasible. Even if we were to use something like a retractable cable, it could cause safety risks and R.O.A.M. potentially could lose connection with an unrealistically long cable in a real redeployment scenario. Within the realm of this project the goal is limited to a lightweight and cost-effective solution for the field deployment stage where it will travel over a football field in length uninterupted.

3.8.1.4 Conclusion

In conclusion, it is clear that cellular data offers more advantages over wifi via satellite connection. While satellite wifi offers broad connection and does not need previous infrastructure it does not meet the practical specifications that R.O.A.M. requires to be a mobile, sidewalk traveling robot. Satellite wifi lacks mobility, simplicity, and energy efficiency which are all essential parts. In comparison, cellular communication strikes a balance between reliability, mobility, and ease of deployment. It even has more freedom in where it is accessed, and has a smaller footprint than if wifi via satellite communication was used. Not only that but cellular also has built in GPS and with the compatibility with embedded systems makes it the clear choice for the project's needs.

Table 3.23 — Satellite vs Cellular

Technology	Pros	Cons
Satellite WiFi	● Potential Global coverage ● Can have high bandwidth ● Does not depend on local infrastructure	● Very Expensive to set up and maintain ● Large physical ● footprint ● Can drop due to weather or if signal is obstructed ● High latency ● Equipment is not designed for a moving object
Cellular Communication	● Lots of coverage in places that have cell towers ● Compact and energy efficient ● Easy integration with embed systems ● Designed for systems that move ● Built in GPS tacking	● Needs data plan and sim card ● Possibility of dead zones ● Potential latency if network is congested

	• Numerous choices for cell plans that can fit whatever is needed	
Wired communication	• High bandwidth • Low latency • Fast bandwidth	• Does not allow mobility • Potential safety hazard

3.8.2 Wired communication protocol

Another important piece of this project is choosing how the Raspberry Pi and the Esp32 are going to communicate. The Raspberry Pi is going to be processing images, processing LiDAR data, and running machine learning models. Whether the data is coordinate points, images or other metadata, the Raspberry Pi is then going to send that data to the ESP32 which will forward it to the server via a GSM module. The data transfer should be reliable, quick, and low latency to accomplish the need for real time operation. The three most common communication protocols that are used for processor to processor communication are SPI, UART, and I2C.

3.8.2.1 UART

Starting off, UART (Universal Asynchronous Receiver/Transmitter) is globally supported due to how easy it is to set up, making it one of the better options for transmitting small amounts of data. This protocol uses two pins, one being RX (receive) and one being TX (transmit), which most modern microcontrollers support. While they are easy to connect, there are a few things that need to be coded in order for it to work. One of these things is the baud rate (speed of data transmission) needs to be the same for both. Some microcontrollers automatically set the baud rate without previously knowing the rate of what's being set, this is called ABR or autobaud. Having to set the baud rate is due to UART being asynchronous meaning the microcontrollers do not have a shared clock signal. Although UART can be pushed up to 2 or 3 Mbps, it can become unstable for large sets of data like when sending photos. UART is better suited when sending smaller data chunks like GPS coordinates. Furthermore, UART can be half duplex or full duplex. Half duplex means the microcontroller can only transmit or receive data at any point in time, while full duplex allows for simultaneous two way communication.

3.8.2.2 I2C

I2C (Inter-Integrated Circuit) on the other hand uses two lines and communicates using the master and slave protocol via SDA and SCL. SDA, or serial line data is a line that is used for the actual data during the transmission process. The data in this line can flow in both directions. SCL, or serial clock line, synchronizes the clock signal between the two devices, this makes sure that the data is being transmitted and received at the same rate. While I2C is the perfect choice when sending data from sensors, it's not made for sending big chunks of data. This is not ideal for what R.O.A.M. needs, as I2C could possibly result in delays when possibly sending images.

3.8.2.3 SPI

Last but not least, SPI (Serial Peripheral Interface) operates at high speeds (20Mbps or more) and supports full duplex mode depending on the microcontroller. One of the drawbacks is the amount of wires needed. The wires needed are MOSI (master out slave in), MISO (master in slave out), SCLK(serial clock), and CS (chip select). MOSI's line takes data from the master and sends it to the slave, MISO is the line that takes data from the slave device and sends it to the master. The SCLK synchronizes the clock rate between the micro controllers, and the CS selects which slave is being talked to if multiple slaves are connected. While there is more wiring than the previous options, the tradeoff is more than worth it due to the speed, reliability, and efficiency of SPI. This would allow the Raspberry Pi to send data to the esp32 without the bottlenecks that UART and SPI had. This is necessary due to the fact that the data needs to be sent to the server quickly and reliably. SPI gives a system the ability to send data with minimal delay and minimal data loss, unlike UART or SPI.

Another key advantage that SPI has over the other communication protocols is its compatibility with DMA. This means that instead of using the CPU, SPI pulls data directly from the device's memory. This not only decreases latency but also decreases power usage. Error checking can be done in the protocol layer and timing accuracy is ensured due to the clocks being synced.

3.8.2.4 Conclusion

In conclusion, the best option for R.O.A.M. is SPI due to the data that needs to be exchanged. It offers low latency, efficiency, and low risk for data corruption. This makes it ideal for real time data transmission. UART or I2C could still be used for transmitting small bits of data like GPS coordinates or sensor information, but the clear winner is SPI due to how its better in almost every way.

Table 3.24 — Wired Communication Protocol

Technology	Pros	Cons
UART	● Simple to implement ● Low CPU overhead ● Two wires ● Best for sending small bits of data ● Built in support	● Slow for large data ● More prone to ● timing errors since clock is asynchronous ● Limited error detection
I2C	● Supports multiple devices on same bus ● Two wires	● Slow for large data ● Not reliable for large data

	• Best for low-speed sensors • Built in support	• Sensitive to noise • High chance for bus collisions
SPI	• Full-duplex and synchronous • Supports DMA • Very high data rates • Good for sending large data chunks • Reliable • Low latency	• Requires 4 or more wires • Short range (<1 m) • More complex software

3.9 ESP32 Telemetry Dashboard

The development of a telemetry dashboard driven by ESP32 microcontrollers, communicating via a SIM7600G-H cellular module and supported by a Flask and SQLite backend, requires careful selection of full-stack technologies that balance performance, resource efficiency, and scalability. This section establishes a deep comparative analysis of architectural layers, including API paradigms, data storage engines, user interface structures, and backend deployment models, guided by the constraints and operating conditions characteristic to embedded systems and budget MCUs.

3.9.1 API Design Considerations

Central to any telemetry application is the design of its Application Programming Interface (API). In the context of ESP32 devices transmitting environmental or positional data to a remote server, a RESTful API presents a compelling baseline due to its simplicity, stateless nature, and native compatibility with HTTP protocols. RESTful services map naturally onto CRUD-style operations and support standard caching mechanisms, facilitating efficient use of mobile bandwidth is a critical consideration when data is transmitted via the SIM7600G-H module over cellular networks.

However, REST's rigid resource structure and potential for over-fetching can introduce inefficiencies when telemetry payloads vary dynamically or require fine-grained selective updates. Alternative paradigms such as GraphQL allow clients to specify exactly which data fields they require, reducing payload size and eliminating redundant data transfers. This can be particularly beneficial in mobile telemetry contexts where uplink capacity is constrained. Nonetheless, GraphQL's server-side resolver complexity and difficulties with hierarchical caching mechanisms may limit its practicality on resource-constrained backends like Flask. Furthermore, it introduces an additional layer of abstraction that can obscure debugging, complicate versioning, and degrade overall system transparency.

Procedural APIs using Remote Procedure Calls (RPC) or gRPC offer another path, supporting binary message serialization and client-side code generation. These are particularly suitable for high-throughput environments or when streaming telemetry data in real time. However, their reliance on binary formats and persistent connections can render them less resilient to the intermittent connectivity that characterizes field deployments over LTE CAT-1 via the SIM7600G-H.

3.9.2 Database Strategy and Data Modeling

Given the continuous and structured nature of telemetry data, the choice between relational and non-relational databases significantly impacts system performance and maintainability. SQLite, a serverless and lightweight relational database, emerges as a strong candidate for edge-side storage. Its minimal memory footprint, simplicity of integration with Python via Flask, and support for SQL transactions make it ideal for buffering telemetry locally prior to batch upload or analysis. SQLite performs reliably for low- to medium-frequency inserts and queries, and its file-based architecture suits storage on microSD or onboard flash. However, limitations in concurrent write access and the lack of native support for data partitioning may restrict its scalability in multi-client environments.

For backend aggregation and visualization, non-relational databases offer greater flexibility and performance. Document-oriented stores such as MongoDB allow variable schema structures and are well-suited to capturing diverse telemetry formats that may evolve over time. In scenarios where the primary data type consists of timestamped scalar values, time-series databases such as InfluxDB or TimescaleDB offer specialized indexing and downsampling tools that improve query responsiveness and reduce storage overhead.

InfluxDB, in particular, supports high-ingest workloads and built-in retention policies, making it appropriate for high-density telemetry streams over persistent connections. TimescaleDB, built atop PostgreSQL, combines the robustness of relational databases with specialized time-series functions, offering superior performance for historical data queries and batch analytics. These platforms, however, require additional deployment and maintenance considerations, and may exceed the complexity justified by simple dashboards designed for occasional monitoring.

3.9.3 User Interface Architecture and Frontend Technologies

The presentation layer of the dashboard determines the accessibility and responsiveness of telemetry data. Server-side rendering (SSR) traditionally provides fast initial load times and broad compatibility, particularly on low-powered devices or in bandwidth-constrained environments. This approach aligns well with Flask's default rendering capabilities, allowing HTML pages to be assembled with current telemetry values prior to transmission.

In contrast, single-page applications (SPAs) emphasize client-side rendering and dynamic updates, often via JavaScript frameworks such as React or Vue. These enable interactive

dashboards capable of refreshing data in real-time via asynchronous requests, without full page reloads. However, SPAs introduce complexity in managing application state, handling authentication, and addressing search engine optimization challenges. They also require a larger initial payload and browser compute capacity, potentially limiting their effectiveness when accessed from low-bandwidth connections.

Tailwind CSS presents a compelling frontend option due to its utility-first design philosophy, which promotes rapid development and consistent styling. Its integration with SPAs or SSR templates allows for modular component construction, facilitating maintainable and responsive layouts without the weight of extensive custom CSS frameworks. Moreover, its purge and minification capabilities yield minimal bundle sizes, aligning with mobile-first design principles crucial for telemetry interfaces accessed from the field.

3.9.4 Backend Deployment and System Architecture

The deployment model for the telemetry system plays a decisive role in its operational resilience and scalability. A monolithic backend—consisting of a Flask application and SQLite database—offers simplicity and fast iteration. This design suits embedded projects and edge scenarios where the server operates on local hardware with limited concurrency demands. It also simplifies the hosting of APIs and dashboard rendering within a unified process, streamlining configuration and debugging.

Microservices architectures offer greater scalability by separating concerns into independently deployable units. In large-scale telemetry applications, this enables individual services—such as data ingestion, storage, analytics, and notification—to scale according to demand. However, they require orchestration tools, network proxies, and often containerization platforms such as Docker or Kubernetes. While these tools enhance reliability and fault isolation, their complexity and resource overhead may not justify adoption for modestly scoped ESP32 dashboards.

Hybrid approaches, which combine monolithic local operation with selective microservice offloading—such as routing analytics to cloud-hosted time-series databases—can balance the benefits of simplicity with scalability. This is particularly useful when telemetry ingestion occurs intermittently, and deeper data analysis or historical reporting is outsourced to more capable infrastructure.

3.9.5 Ancillary Services and Operational Considerations

Beyond core stack components, successful telemetry deployment requires attention to cross-cutting concerns such as authentication, caching, logging, and monitoring. Lightweight token-based authentication mechanisms, such as JSON Web Tokens (JWT), provide secure access control without imposing session state on the backend. These integrate well with RESTful Flask applications and can be verified asynchronously for scalable API protection.

Caching telemetry responses, especially recent data, can reduce database load and improve frontend responsiveness. In-memory key-value stores such as Redis offer millisecond-scale access to frequently requested values, and can be used to serve hot sensor data or retain transient state during active dashboard sessions.

Structured logging and observability pipelines allow developers to analyze performance bottlenecks and error conditions. Integration with tools such as Prometheus for metrics and Grafana for visualization can surface telemetry throughput, API latency, and storage saturation in real time. These diagnostics are essential for maintaining service quality as telemetry volumes increase or field conditions degrade.

3.9.6 Conclusion

The selection of technologies for an ESP32-driven telemetry dashboard, particularly one augmented by cellular connectivity via a SIM7600G-H module, must reflect the unique constraints of embedded devices and remote deployment. RESTful APIs, lightweight databases such as SQLite, Tailwind-based UI frameworks, and monolithic Flask applications strike a pragmatic balance between performance, simplicity, and maintainability. Where data volumes or analytic demands exceed local capabilities, strategic adoption of time-series databases and microservice components can extend system functionality without abandoning the efficient foundations of the embedded stack. Ultimately, this modular and thoughtful approach ensures that telemetry systems remain responsive, secure, and adaptable under evolving usage scenarios.

3.10 Object Detection and Classification

3.10.1 Object Detection Method

To detect and classify objects found along the side of the road, the R.O.A.M. system requires a reliable object classification method. While machine learning has become the dominant approach in recent years due to its flexibility and accuracy, it is important to also consider traditional computer vision techniques such as rule-based systems and template matching. Each method offers different strengths and trade-offs depending on the complexity of the environment and available resources.

3.10.1.1 Machine Learning

Machine learning, also known as deep learning, has been made popular in recent years due to its development and versatility. Machine learning is widely used in computer vision applications like R.O.A.M. and this provides ample reason to consider machine learning for our approach. Machine learning can achieve a considerably high accuracy in object classification tasks. Machine learning is capable of detecting smaller and finer details than traditional computer vision methods, even having the ability to detect objects in unfamiliar orientations or when partially obscured. Moreover, when a machine learning model is sufficiently trained on an object it will be able to detect it regardless of clutter, lighting, size changes, or physical distortions or abrasions. Once trained a model

can handle many different objects as well, making it a very robust solution. Machine learning learns the details of these objects through training. Training a machine learning model includes feeding it a bunch of images of the object you'd like it to know and letting it learn the details from images it knows are the object, later being able to detect the object in pictures where it doesnt know if its there. Being capable of learning from example instead of hard-coded boundaries is an incredible advantage of machine learning and has many applications when you consider a single model can identify multiple objects like in R.O.A.M. which is trying to identify furniture and trash. This can be used to expand R.O.A.M.'s functionality in the future by potentially having R.O.A.M. identify an object and assess its condition. Machine learning models also have the ability to improve over time with certain implementations. It is possible to have the model trained on current data it receives with user verification further improving accuracy over time and allowing for a reduction in false positives by allowing the system to put certain objects within a range of confidence within a questionable category.

The largest drawback of machine learning models is the reliance on a quality database. Machine learning would require the gathering and labeling of images with objects of interest in them which is time-consuming and can be expensive. This dependence also has environmental factors, meaning if the environment that training photos are taken in is different than the deployment environment it will be more difficult for the model to correctly detect objects, decreasing accuracy and reliability. Additionally, Machine learning models are computationally expensive. Training a model is the most expensive, requiring powerful GPUs with sometimes days of time to train. Deploying the trained model could require hardware-acceleration or additional hardware to run on edge devices. Having constrained or limited resources on edge devices may limit Machine learning's ability to function as its draining on resources like CPU, Battery, and RAM. During the training and testing phases it can be hard to determine where machine learning models make errors and why they make the decisions they make. This makes it difficult to debug specific misclassifications and other potential errors with little work around besides retraining the model, which takes time.

3.10.1.2 Traditional Computer Vision Methods

Traditional computer vision methods rely on hand-crafted algorithms and image processing techniques to interpret visual data. Instead of learning from data like machine learning models, these approaches use predefined rules and mathematical operations to extract features such as edges, shapes, colors, and textures. Common techniques include edge detection, contour analysis, thresholding, and template matching. While they are lightweight and easy to understand, traditional methods often struggle with the variability and complexity of real-world environments.

3.10.1.2.1 Template Matching

Template matching is a method used in computer vision to find parts of an image that match a stored example, called a template. The algorithm works by sliding the template over the input image and comparing it to different regions. If the similarity is high, the

system considers it a match. This method is simple and works well when the object's size, shape, and angle do not change. However, it is less effective when objects are rotated, scaled, or partly hidden.

One of the main benefits of template matching is that it's very simple to use and understand since it doesn't require any training data to work, which makes it useful for quick development and experimentation. This also makes template-matching work well when you're working with a known object that won't change much. Also, its predictable because it follows the same steps every time and that makes it easier to debug compared to other methods that are more complex and harder to understand, and when the environment is controlled and the object always looks the same then template matching can actually be very accurate. Also, if the template and the input image are both small then it doesn't need a lot of processing power which can help when using edge devices.

The biggest problem with template matching is that it only works if the object looks exactly like the template, so if the trash is turned, crushed, partly hidden or looks different than expected even a little bit then the method usually fails and gives wrong results or no result at all, which is a serious issue in outdoor environments like the one R.O.A.M. is found in where specific conditions are extremely unpredictable. It also can't deal with changes in lighting or if the trash is different sizes because it doesn't adapt to those changes. meaning there is a need to store tons of templates for every possible case which isn't realistic and would slow the system down as it tries to match every single frame of a live video to every single template in every single place. Because of that, this method doesn't work for R.O.A.M. since the robot needs to detect objects that come in many forms, angles and conditions, and template matching simply can't handle that level of variety no matter how many templates you try to use.

3.10.1.2.2 Rule based Systems

Rule-based systems are a traditional method used in computer vision where specific if-then rules are written to classify objects based on measurable features like size, color, or shape. These systems don't rely on training data or learning algorithms, instead they use logical conditions that are manually set ahead of time. For example, if an object is brown and has a rectangular shape then it might be classified as a cardboard box. This approach can be useful in controlled environments where the appearance of objects is predictable and doesn't change very often. In these cases, rule-based systems are fast and don't use a lot of processing power, making them ideal for simple tasks that only require basic classification.

One of the main benefits of rule-based systems is how easy they are to understand and create. They don't need any datasets or training, and can be written and changed directly by someone familiar with the objects being classified. This makes testing very quick, and also allows the developer to debug or tune specific parts of the system without retraining anything. Rule-based systems are very lightweight and can run easily on embedded or low-power devices since they don't require complex models. These systems also behave predictably, always giving the same result for the same input which helps with tracking down errors or figuring out why a decision was made.

However, rule-based systems come with a large number of limitations that directly conflict with the environmental needs of R.O.A.M. The main weakness is that these systems are very fragile when dealing with any kind of visual variation. Even small changes like lighting differences, shadows, dirt, damage, or the object being at a different angle can easily break the rule logic. This is a major problem in the case of R.O.A.M. because trash and reclaimable items rarely look the same twice. On top of that, adding more rules to cover more cases doesn't fix the problem and instead just creates more complexity, which is hard to manage and increases the chance of false-positives. It's also important to consider that R.O.A.M. operates in a constantly changing outdoor environment where the lighting, object conditions, and angles are unpredictable. Rule-based systems were not made to handle this type of inconsistency and would fail frequently, even with constant adjustments. In addition, rule-based systems don't scale well, meaning if we wanted to detect a large variety of object types, the number of rules would grow too fast and make the system harder to maintain. This means even if R.O.A.M. started with a few accurate detections, it would quickly become unreliable as more variation is introduced.

3.10.1.2.3 Feature based Systems

Feature-based systems are a type of traditional computer vision method that work by looking at important points or shapes in an image called features. These features can be things like corners, edges, lines, or patterns that stand out and are easy to track or compare. Once these features are found, the system can try to match them to objects that it already knows. Some common examples of feature-based techniques are SIFT, ORB, and HOG. These systems don't need full templates or large rule sets, they just use the shapes and textures of the object to figure out what it might be.

Feature-based methods are more flexible than rule-based systems and can still work if the object is rotated or slightly changed. These systems don't need to scan through the entire image like a template matcher does, so they are usually faster than that. They also don't need training like a machine learning model, so you can develop without collecting and labeling a big set of images. Another benefit is that they don't rely on exact color or size, which makes them more reliable when those things change in the environment. In some cases, feature-based systems can even help speed up other systems by narrowing down where the important parts of the image are.

Feature-based methods also run into problems that make them a bad fit for R.O.A.M. The biggest issue is that these systems still depend on objects having clear and stable features. If the object is damaged, covered in dirt, crushed, or partly hidden, the features might not be found at all or wrong. That means the system could miss objects or confuse them for something else. Also, these systems can still get confused by lighting changes, shadows, or movement blur, which are all things that R.O.A.M. will constantly deal with. Since most trash and reclaimable items do not have strong or consistent features, the feature-matching process doesn't always have enough information to work with. Another issue is that feature-based methods don't do well when there are lots of different types of objects. You would need a lot of different feature sets to compare with, and it gets slow and messy to manage. Lastly, most of these systems were built for clear, close-up images

or specific tasks—not for a mobile robot that's trying to classify random trash on a busy sidewalk.

3.10.1.3 Conclusion

In conclusion, Machine learning is clearly the most viable solution for R.O.A.M. Not only will Machine learning be able to detect and classify objects through the unpredictable environment but it can handle many different types of objects. Machine learning's computational needs is the only crux, however we are not processing much else besides the machine learning model so there should be enough resources for the model to function. Traditional computer vision options require too much to be guaranteed about our environment and would require too much fine tuning for the variety of objects that R.O.A.M. will come across.

Table 3.25 — Object Classification Method Comparison

Method	Pros	Cons
Machine Learning	● Learns complex patterns ● Handles variation in shape, lighting, and angles ● Scales to many object types ● Can Improve over time	● Needs large dataset to train ● Requires more processing power ● Can be hard to understand decisions
Template matching	● Very simple to use and understand ● Doesn't need training data ● Accurate in controlled conditions	● Fails with rotation, scale, or lighting changes ● Slow if many templates are used ● Can't handle varying or distorted objects
Rule-based system	● Easy to build and debug ● No training required ● Lightweight and fast	● Breaks with small visual changes ● Hard to scale ● Not flexible or adaptive
Feature-based system	● Better than rules/templates with some variation ● No training needed ● Can handle rotation and size to a degree	● Still fails in messy or low-quality images ● Struggles with damaged or small objects ● Not good with many

		object types

3.10.2 Object Detection and Classification Model

Object classification is the process of identifying and categorizing objects within an image or video frame based on images the process is trained on. In the context of R.O.A.M., this capability is essential for enabling the robot to distinguish between various types of items it encounters: general trash, recyclables, and large reclaimable objects like furniture or appliances. By accurately classifying objects in real time, R.O.A.M. is able to report the necessary information back to the website for people to view and act on. This is particularly important for making sure reclaimable objects and their locations are documented and accessible. Without reliable object classification, the robot would lack the contextual understanding needed to perform its main tasks effectively.

3.10.2.1 YOLOv8n

YOLOv8 is a high-speed object detection model based on the YOLO algorithm family. YOLOv8 was designed for real time analysis and is scalable from sizes nano (n) to extra-large (x). Its smaller sizes, like nano, are made to run on edge devices like the one we will be using on R.O.A.M.. YOLOv8 has a high mean Average Precision (mAP), this is the measurement used to describe the overall accuracy of a model. This model has greatly improved accuracy compared to its previous iterations. Computational time has been reduced compared to previous models, in the context of R.O.A.M., YOLOv8n should be able to run at 8 to 15 frames per second. YOLOv8 has a streamlined training pipeline, it uses a command-line interface (CLI) to avoid users having to write code to train YOLOv8 and involves built-in data augmentation. This makes it easier to train and experiment with different datasets to achieve the highest accuracy possible. YOLOv8 also supports export to a variety of deployment formats, including ONNX and TensorRT, making it well suited for use on edge devices like the Raspberry Pi or NVIDIA Jetson.

Despite its advantages, YOLOv8 comes with some limitations. The model is tightly coupled to the Ultralytics ecosystem, which can make it difficult to customize internal components or integrate with non-standard workflows. Additionally, while the model performs well out of the box, it still requires post-processing steps to work with tracking systems like SORT. If we do choose to use additional models in tandem with YOLOv8we would have to remain aware of these post-processing requirements. As a newer release, YOLOv8 also has fewer third-party extensions and community tools compared to longer-established frameworks, which may be a consideration for teams requiring more flexibility or support.

3.10.2.2 EfficientDet and EfficientNetB1

EfficientDet and EfficientNet working together provides an object classification model. EfficientDet detects objects and EfficientNet provides feature extraction and analysis. This paired-architecture allows the model to benefit from EfficientNet’s high-quality visual feature representations while maintaining the overall efficiency and balance that

EfficientDet is known for. This provides some strong accuracy for a compact model. making it a solid option for situations where classification precision is more important than raw speed. Its compound scaling approach helps the model adapt to various input sizes and object scales, which is valuable for detecting large reclaimable items or scattered trash in diverse outdoor environments. Due to the division of labor in these two models, EfficientNet is able to capture finer visual details, reducing the likelihood of false positives when the robot encounters cluttered backgrounds or low-contrast objects.

Despite the accuracy benefits that EfficientDet and EfficientNet provide, there are significant drawbacks to consider. Even the lightest models available, which are designed to run on edge devices, are only able to process about 2-5 frames per second. This is incredibly slow for the application in R.O.A.M. especially while in motion. The training pipeline is built around TensorFlow and requires more manual setup, including dataset preparation and hyperparameter tuning, which can slow development. Deployment also demands additional optimization steps such as quantization to achieve acceptable performance, and even then, the model may place heavy demands on memory and resources which are already very limited. Integrating EfficientDet with object tracking systems like SORT adds further complexity, as its output format needs extra post-processing to meet the conditions of any other model's inputs. Finally, the model's smaller ecosystem means tools and tutorials, making customization and troubleshooting more time-consuming.

3.10.2.3 SSD

Single Shot Multibox Detector (SSD) can detect objects in just a single pass of an image. Other models need several proportioned regions to be a specific size in order to analyze, but SSD does not. This decreases the need for computational power and is one of the ways SSD remains a considerably light-weight model at under 10MB. Due to its lightweight nature and low compute requirements, SSD consumes less power than heavier models, which is critical for battery-powered robots like R.O.A.M. that operate outdoors for extended periods. The methods that make SSD light-weight also make it incredibly fast being able to run at 15-25 frames per second on edge devices. SSD is extremely compatible, being able to work with many lightweight frameworks like Tensorflow lite making it more compatible with a wider range of hardware, especially like the ones found on low-power systems like R.O.A.M. SSD's single-shot detection approach is computationally efficient and easy to understand, making it a good starting point for projects that need fast prototyping or basic object detection capabilities. SSD outputs standard bounding boxes with class probabilities, making it easy to connect with tracking algorithms like SORT with minimal post-processing.

However, SSD's speed and efficiency come at the cost of accuracy. It often performs poorly when detecting small or partially obstructed objects, which is a significant limitation for R.O.A.M., where trash can appear in a variety of sizes and may be partially hidden behind other objects or blended into backgrounds. The model's grid-based detection system lacks the fine localization and detail extraction needed for precise identification of smaller items, leading to frequent false negatives. In outdoor conditions R.O.A.M. regularly encounters with shadows, clutter, and inconsistent lighting, SSD becomes less reliable, misclassifying or completely missing key targets. Additionally,

SSD is an older architecture that lacks many of the accuracy improvements seen in more modern detectors. While its simplicity benefits deployment and power consumption, its limited ability to handle edge cases or difficult scenes reduces its overall usefulness for real-world, high-accuracy tasks like those required by R.O.A.M.

3.10.2.4 Conclusion

After evaluating YOLOv8, SSD, and EfficientDet with EfficientNet, YOLOv8 stands out as the best option for R.O.A.M. due to its strong balance between accuracy and computational efficiency. It is fast enough for real-time detection while maintaining high precision, which is essential for identifying trash and reclaimable items in dynamic outdoor environments. SSD, while lightweight and fast, suffers from low detection accuracy and struggles to identify small objects, making it unreliable for R.O.A.M. 's needs. EfficientDet offers higher accuracy than SSD but is too slow to support responsive detection, limiting its practicality for real-time use in the field.

Table 3.26 — Object Classification Model Selection

Model	Pros	Cons
YOLOv8	● High accuracy ● Real-time performance on edge devices ● Easy training with Ultralytics CLI ● Strong community support ● Good small object detection	● Larger models require more resources ● Still needs post-processing
EfficientDet + EfficientNetB1	● Good accuracy ● Strong feature extraction ● Scalable model sizes ● Supports TFLite and ONNX for edge deployment	● Slow on edge devices ● Complex TensorFlow-based training pipeline ● Harder to integrate with tracking ● Higher resource demands
SSD	● Very fast deployment runtime on edge devices ● Low power usage ● Small model size ● Easy to integrate with tracking ● Broad deployment compatibility (TFLite, OpenCV)	● Low detection accuracy ● Weak performance on small or cluttered objects ● Outdated architecture

3.11 C++ vs Rust vs Python

When developing software for embedded systems there are several factors that need to be taken into account. These factors help us evaluate the differences between C++ and Python in order to choose a programming language that can best fit our case. Edge devices have access to only a limited amount of resources, making them valuable. A considerate approach to the use of these resources is required. This sets some factors to analyze: runtime speed, memory efficiency, multithreading. How the language affects the other components is important to consider, especially in a real-time performance. This is seen in factors like power consumption, responsiveness, and hardware integration. Beyond the language's individual capabilities in deployment, the language's performance during development should also be considered: ease of use, available libraries, debuggability.

3.11.1.1 Python

Python offers many advantages for development. Python is often involved in machine learning because it allows for easy integration and has a vast ecosystem including OpenCV, Numpy, and TensorFlow which are all valuable tools when in the development phase of a project. Ease of use and readability make it easy to troubleshoot and debug python code as well as getting advice from and collaborating with others. During a development phase, edits and changes happen often and troubleshooting can lead to increased issues, python reduces complexity making what needs to be changed more apparent. Python has a broad community due to its beginner-friendly nature because of this support can be easily found. Python's main advantages take place in the development stage.

In the deployment stage python has several drawbacks that become important to consider in resource constricted and performance sensitive environments like R.O.A.M. The most concerning drawback is the poor execution speed due to it being an interpreted language meaning python is not compiled into machine code but instead read into an interpreter. The interpreter executes the python code line by line, increasing overhead. Python consumes additional power and memory doing repetitive tasks that would be included in R.O.A.M. this is partly due to the use of an interpreter but has other contributing factors. For example, Python doesn't constantly free up like other languages but instead does it periodically pausing execution. This is called automatic garbage collection and is unpredictable and disruptive. Garbage collection and other methods greatly increase Python's overhead, CPU usage, and power consumption making it a poor choice for battery-dependent long-running devices like R.O.A.M. When running R.O.A.M. it will be important to use multithreading allowing several functions to work at once like object tracking and object detection. Python's Global Interpretation Lock (GIL) is a significant limitation and does not allow for multithreading. GIL locks python into only using one thread at a time for CPU tasks. This means even with multiple threads true parallelism isn't achievable which impacts performance. Python has dynamic typing which, while helpful in development, makes certain runtime errors hard to detect and diagnose.

3.11.1.2 Rust

Rust has many pros and cons, a lot revolving around efficiency and structure. Rust is similar to C++ in that it offers competitive runtime speed and power consumption, it is great for systems that need predictable performance. This also has memory without garbage collection that the alternatives may have. With Rust you also have error messages that are extremely detailed which can help all types of coders solve their issues without frustration. With this there are also a significant amount of faults, such as lacking ecosystems with fewer libraries and less help as the community is not as large. There are also may be a need for more custom code.

3.11.1.2 C++

C++ has many advantages in the application of R.O.A.M. C++ is a compiled language which means the code is compiled and translated into machine code before its run allowing for a better runtime execution. According to Pereira et al. (2017), C++ demonstrated some of the lowest energy consumption and fastest execution times among the 27 programming languages evaluated, making it ideal for edge devices. In the context of simulation environments like esmini, Soydas (2024) found that C++ implementations consistently outperformed Python in terms of execution time when interacting with the Open Simulation Interface (OSI). Even though R.O.A.M. is not a simulation project, performing at a much higher execution time is a huge benefit. This contributes to C++ being rated as one of the lowest power consuming and fastest executing languages making it a strong option for edge devices with power constraints. C++ features deterministic memory management which gives us greater control over memory allocation and hardware level operations. This allows for the ability to optimize code further than other programming languages, cutting down on unnecessary resource usage. Many libraries including OpenCV were originally created in C++ so C++'s access to libraries as well as community help is great.

C++'s main drawback is its more complex to program in and more difficult to troubleshoot. C++ has a steeper learning curve which leaves more room for error and mistakes. One of which could be mismanaged memory. Memory leaks can become a major issue causing the program to fail at unknown times which would shut down all functionality. Since R.O.A.M. has limited memory this is a big concern which would require extensive testing and troubleshooting. Debugging in C++ can be difficult. C++ lacks built-in conveniences that make it overall harder to work in.

3.11.1.3 Conclusion

C++ and python both have distinct advantages and disadvantages. If R.O.A.M. is developed in C++ it would run much more efficiently and the potential for errors in deployment, but it is harder to develop. Python is much easier to develop in but brings way too many drawbacks and inefficiencies in deployment. With that in mind it would be best to develop and test in python and once we have working code and an adequately trained Object classification program we can transfer it to C++ to take advantage of its deployment benefits and avoid much debugging and retesting in C++. This is possible because of the libraries that C++ and Python both share, making it easy to translate from

one language to another and take full advantage of the benefits from both languages while minimizing their drawbacks.

Table 3.27 — C++ vs Rust vs Python

Language	Pros	Cons
C++	• Fast and efficient at runtime • Uses less power and memory • Good for real-time tasks and multithreading • Gives full control over hardware and memory • Works well with libraries like OpenCV	• Harder to learn and write • Debugging is more difficult • Memory errors can crash the system •
Python	• Easy to learn and use • Great for fast development and testing • Large library support • Strong community help	• Slower performance • Uses more power and memory • Can't run true multithreading • Errors can be harder to catch during runtime
Rust	• Memory safety • No garbage collection • Detailed error messages	• Less support • More custom code • Fewer libraries

3.12 Web Development Suite

Building a telemetry dashboard for R.O.A.M.'s ESP32-driven, SIM7600G-H-enabled edge system demands a coherent selection of technologies spanning the user interface, server logic, data storage, and supportive services. Each choice must respect the project's constraints—limited power, intermittent connectivity, modest budget, and a small development team—while delivering real-time responsiveness, reliability, and long-term maintainability. This section justifies the selection of TailwindCSS for the frontend, Flask and SQLite for the backend, RESTful APIs for device-to-server communication, and ancillary services that round out a full-stack architecture optimized for embedded telemetry.

3.12.1 UI Selection

The user interface must render live sensor readings and geolocated object classifications on viewports ranging from desktop monitors to smartphones connected over cellular links. Performance, accessibility, and ease of customization are all critical factors.

3.12.1.1 Suitability of TailwindCSS

Utility-first styling with TailwindCSS allows developers to compose responsive, accessible layouts directly within HTML markup, eliminating the bulk of unused styles and accelerating component development. Thanks to its Just-In-Time compiler and PurgeCSS integration, Tailwind produces CSS bundles under 40 KB, which is crucial for users on LTE links from the SIM7600G-H module. Its breakpoint modifiers, semantic color utilities, and spacing classes foster a mobile-first design that gracefully adapts from full-screen maps on desktops to narrow viewports on smartphones. This atomic approach reduces context-switching, eases on-boarding for future contributors, and integrates seamlessly with JavaScript charting libraries for live telemetry graphs.

3.12.1.2 Bootstrap 5

Bootstrap 5 provides a comprehensive suite of pre-styled components and a mature grid system that simplifies initial UI construction. Its extensive documentation and large community ensure straightforward troubleshooting and longevity. On the downside, Bootstrap's default theme and component styles often require overriding to achieve bespoke designs, leading to larger CSS payloads, commonly exceeding 140 KB, and potentially longer load times on constrained networks. The framework's opinionated naming conventions and embedded JavaScript plugins can also introduce unintended visual or behavioral side effects when integrating with custom charting or mapping libraries. For prototyping and rapid development, Bootstrap excels; for optimized performance and fine-grained control, its overhead proves burdensome.

3.12.1.3 SvelteKit + DaisyUI

SvelteKit, paired with the Tailwind-based DaisyUI component library, offers a compelling hybrid: a reactive JavaScript framework with pre-styled, accessible components. SvelteKit's compiler converts UI code into minimal imperative JavaScript, resulting in fast initial loads and near-zero runtime overhead. DaisyUI leverages Tailwind's utility classes to supply a consistent visual language without large external stylesheets. This combination delivers a fluid, SPA-like experience—ideal for interactive telemetry dashboards—while maintaining bundle sizes in the 50–60 KB range after tree-shaking. The trade-off lies in the more sophisticated build pipeline and developer tooling required, including Node.js version management, module bundling, and hydration strategies for SEO. Teams unfamiliar with reactive paradigms may face a steeper learning curve, though the payoff in runtime performance and modularity can justify the investment.

3.12.2 Backend Selection

The backend must ingest telemetry uploads from ESP32 clients, persist data reliably, and expose a clear API contract for the UI. Considerations include language ecosystem, concurrency model, integration complexity, and long-term maintainability.

3.12.2.1 Flask Microframework and SQLite

For the server tier, Flask emerges as the ideal microframework: its minimal core encourages explicit route definitions and a clear separation between API endpoints and template rendering, while extensions such as Flask-CORS and Flask-Login deliver cross-origin support and token-based authentication with minimal ceremony. Python's native sqlite3 module suffices to persist timestamped sensor readings, object classifications, and GPS coordinates in a lightweight, file-based database. SQLite's zero-configuration setup, ACID compliance, and transactional guarantees align with project priorities—there is no requirement for high-volume concurrent writes, and the entire data store resides on the same Oracle VM that hosts the Flask application. In practice, periodic vacuuming and background backups ensure long-term integrity without demanding a full database administrator.

3.12.2.2 Node.js with Express

Express, running atop Node.js's single-threaded, event-driven I/O model, excels at handling large numbers of concurrent, small payload requests like frequent telemetry pings which the esp32 may be utilizing. Its middleware pipeline offers granular control over JSON parsing, logging, and error handling, and its rich npm ecosystem provides ready-made modules for CORS, validation, and caching. As both the telemetry client (ESP32) and the dashboard often rely on JSON over HTTP, a unified JavaScript stack can streamline development. Nonetheless, Express's reliance on asynchronous callbacks or Promise chains can lead to callback-style complexity, and CPU-bound tasks such as image processing or complex database joins can block the event loop unless offloaded. For teams with strong JavaScript expertise and a need for high-throughput ingestion, Express is attractive; for lightweight deployments centered on Python-based tools, it may introduce unnecessary context switching.

3.12.2.3 RESTful API Paradigm

A RESTful interface provides a universally understood contract for ESP32 clients and browser-based dashboards alike. Defining resources—/readings, /classifications, /locations—mapped to HTTP verbs (GET to fetch current values or history, POST to ingest batched telemetry) facilitates straightforward JSON interchange. Stateless interactions simplify horizontal scaling when the service outgrows a single VM, and HTTP status codes communicate success or failure unambiguously to embedded callers. Alternative paradigms such as GraphQL, despite their fine-grained querying advantages, were set aside due to their caching complexity and additional server-side orchestration, which would distract from core functionality under tight deadlines.

3.12.2.4 FastAPI

FastAPI combines modern Python type hints, automatic OpenAPI schema generation, and an ‘async-first’execution model to deliver high throughput, benchmarks regularly exceed 1,200 req/s on commodity hardware. Its dependency-injection system simplifies validation, security, and database session management, and its auto-generated documentation via Swagger UI accelerates both development and hand-off to front-end teams. While its performance surpasses Flask under load, FastAPI’s asynchronous paradigm necessitates careful design of dependencies to avoid inadvertent blocking calls. The requirement for Python 3.7+ and familiarity with async/await syntax can lengthen onboarding. In scenarios where telemetry volume grows or real-time streaming endpoints (WebSocket) are desired, FastAPI offers a scalable, robust solution without sacrificing all-in-one cohesion.

3.12.2.5 Over-the-Air Updates and Long-Term Support

Long-term operation in the field requires automated deployment of both firmware and backend changes. A continuous integration pipeline triggered by Git tags can run linting, unit tests, and schema migrations, then package and deploy the Flask application to the cloud VM. For ESP32 firmware, the same pipeline publishes binary artifacts to a secured HTTP endpoint; the device’s bootloader periodically checks for updates, verifies a cryptographic signature, and swaps to the new image if valid. This end-to-end automation reduces on-site intervention, ensures consistent versioning across devices, and closes the loop for rapid bug fixes during live demonstrations.

3.12.3 Ancillary Services Final Selection

Telemetry caching, messaging, and asset distribution round out the architecture. Redis serves as an in-memory store of the most recent reading, slashing database load when dozens of clients refresh the dashboard simultaneously. Although MQTT was considered for real-time push notifications, initial prototypes favor short-interval polling to preserve simplicity; a future iteration may integrate an MQTT broker for subscriptions to door-bell alerts or emergency signals. User sessions and API authentication leverage JSON Web Tokens (JWTs) issued by the Flask backend, striking a balance between stateless server logic and per-device access control. Finally, static files, compiled JavaScript bundles, TailwindCSS assets, and map tile proxies, are served through a content delivery network, ensuring sub-200 ms asset fetch times worldwide and offloading traffic from the origin server.

Table 3.28 Fitness Evaluation

Requirement	UI Choice	Backend Choice	Ancillary Service	Fit Rating
Fast Load Over 4G	Tailwind + Alpine.js	Flask WSGI + Gunicorn	OpenTelemetry + Prom	High
Low Memory Footprint	Tailwind (10 KB CSS)	SQLite Embedded	Redis for burst cache	Very High
Ease of Development	Utility-first CSS, minimal JS	Flask microframework	Docker CI/CD pipeline	High
Offline Resilience	Static files, ServiceWorker	SQLite local buffering	Celery offline queue	Moderate
Scalability	Alpine.js for controls	Flask async + WSGI	Kubernetes orchestration	High
Observability	Grafana dashboards	Prometheus monitoring	OT Collector pipeline	Very High

3.12.4 Map Service API

The map service underpins every geospatial overlay in the telemetry dashboard—live object detections, historical GPS traces, geofenced alerts, and navigation hints. It must deliver crisp vector or raster tiles over LTE links, support offline caching in the browser, offer straightforward style customization, and carry a permissive license for academic and commercial deployments.

3.12.4.1 Mapbox

Mapbox's vector-tile platform provides a polished styling editor, global coverage, and client SDKs for web and mobile. A generous free tier (50 000 map loads/month) accelerates prototyping, and hosted tiles are served in under 200 ms on average. Styling via Mapbox Studio requires minimal code changes, but full offline caching demands a paid plan and proprietary client hooks. As usage scales, lock-in and recurring fees can become significant barriers for a budget-conscious project.

3.12.4.2 Google Maps API

Google Maps delivers unparalleled global imagery, place search, and traffic overlays with a mature JavaScript API. Out-of-the-box geocoding and POI data simplify development, but the standard plan charges $7+ per 1,000 map loads and offers no built-in offline-tile caching. Dynamic policy changes and vendor lock-in further hinder cost predictability for a long-lived, field-deployed system.

3.12.4.3 MapLibre GL

MapLibre GL is the community-driven successor to Mapbox GL JS, enabling GPU-accelerated vector rendering without license fees. Its style specification is fully compatible with existing Mapbox themes, and modern browsers support offline cache manifests for tile storage. However, generating and serving MBTiles or vector-tile feeds via tools like Tippecanoe introduces operational overhead and infrastructure complexity for a small team.

3.12.4.4 OpenStreetMap + Leaflet

Pairing OpenStreetMap's free, community-maintained tile servers (or self-hosted alternatives) with the lightweight Leaflet library eliminates usage fees and licensing friction. Leaflet's plugin ecosystem covers marker clustering, heatmaps, and custom overlays, while offline caching can be implemented via ServiceWorker and IndexedDB. Although public OSM servers may throttle high-volume requests, deploying a low-cost tile cache proxy or hosting trimmed extracts on a VPS addresses performance concerns. This stack strikes the ideal balance between zero-cost operation, offline resilience, styling freedom (through custom tile styles or MapTiler-generated schemes), and moderate integration complexity.

Table 3.29 — Map Services Compared

Language	Pros	Cons
OpenStreetMap +Leaflet	• Free and open-source • Lightweight • Community support • Self-hosted offline caching	• Public servers can throttle • Lack of built-in styling • DIY cache and proxy setup
Google Maps API	• Best-in-class images and points of interest • Built-in traffic, geocoding and search • Mature documentation	• Expensive $7/1000 loads • No built-in offline caching • Costs may vary
MapLibre GL	• GPU-accelerated rendering • Mapbox-style compatible • No licence fees	• Must host vector tiles • Higher overhead, limited map-scaling
Mapbox	• Polished ecosystem • Global tiles with low-latency	• Usage fees after free tier • Vendor lock-in • Offline caching is paywalled

Chapter 4 Design Constraints and Standards

4.1 Design Standards

4.1.1 IEEE Standards

The IEEE ((Institute of Electrical and Electronics Engineers) standards cover a wide range of electronic systems, including electronics and communication. IEEE's standards supply safety, reliability, and connectivity when developing projects. Although they are used in a lot of different industries, R.O.A.M. applies these standards for sensor integration, communication, and software lifecycle. Using these standards will greatly improve scalability, productivity, reliability and maintainability.

Since our project uses sensors like GPS, LiDAR, and camera, we need a way to make this data work together. This is done with IEEE 1451, which is a standard for how smart transducers are to be integrated in a system. The standard will help devices under different manufactures to work together seamlessly.

Although R.O.A.M. primarily uses cellular communication to send data through the SIM7600, Wifi is used for testing and developing the prototype. This falls under IEEE 802.11, which defines the protocol for wireless local area networks (WLAN). This makes sure that the communication between the esp32 and the server is smooth.

4.1.1.1 IEEE 1451

IEEE 1451 is a group of standards that define how smart transducers interact with data processing systems. A smart transducer is a sensor or actuator with metadata and embedded processing capabilities. This standard outlines a common model for interface design and communication between parts. This plays an important role in metadata handling since that will provide crucial data. R.O.A.M. will get the most benefit from using IEEE 1451.0-2024, the latest revision of the core model and TEDS formats, but 1451.2, 1451.3 and 1451.4 will be useful for plug-and-play templates.

The two core components of IEEE 1451.0-2024 are a Network Capable Application Processor (NCAP) and a Transducer Interface Module (TIM). A TIM communicates with smart transducers to process data locally. NCAP is a more central device that will communicate with one or more TIMs to collect data and integrate the transducers into the larger network within R.O.A.M. Our edge device will serve as the NCAP because it will coordinate decision making and data or it could manage motors.

IEEE 1451 supports simple, low-level connections like I2C, SPI, and UART, enabling protocol-agnostic transducer integration. This makes it easy to connect sensors and actuators in many different ways depending on the needs of the robot. In R.O.A.M. most communication between devices will happen in UART for its simplicity and wide support

with minimum hardware overhead. This simplified approach allows for minimized miscommunications and errors in edge devices like the GPS module. IEEE 1451 provides a standard in which transducers exchange the TEDS-encoded metadata with the NCAP over a chosen communication bus, and to the main processor without needing custom software for every new sensor. This plug-and-play framework enables seamless scaling and simplifies maintainability. For example, if R.O.A.M. wants to add new sensors to detect other objects in its environment, the system can automatically recognize and adjust to the new device as long as it follows the IEEE 1451 structure. This saves development time, reduces human error, and makes the entire system more reliable and easier to maintain. The system can use shared rules and data formats, making it more modular and expandable.

4.1.1.2 IEEE 802.11

IEEE 802.11 provides the necessary backbone for wireless communication between R.O.A.M.'s embedded modules, especially during development and early testing phases where network infrastructure is more accessible. As a robust standard defining Wireless LAN (WLAN) protocols, 802.11 ensures data can be transmitted over short-range radio frequencies with speed and reliability. For R.O.A.M., the ESP32-S3 module acts as a Wi-Fi client under 802.11n in the 2.4 GHz band, associating with a nearby access point to facilitate telemetry exchange, image uploads, and operational status checks. This connectivity is particularly relevant during object detection and classification, where JPEG images ranging from 1–5 MB are uploaded with minimal delay due to negotiated throughput of up to 150 Mbps and low latencies under 50 ms. The WPA2-PSK encryption protocol, defined under 802.11i, ensures these data packets are securely transmitted—crucial for public-facing deployments where privacy must be upheld. DHCP-based addressing allows R.O.A.M. to connect seamlessly without custom IP configurations, streamlining prototyping workflows. The significance of IEEE 802.11 within R.O.A.M. lies in its scalability and compatibility—its infrastructure-ready model supports plug-and-play operation across diverse testing environments and offers a common protocol for interfacing IoT devices. This aligns with R.O.A.M.'s modular system design, where interoperability and rapid integration across networked components are key. Though cellular communication eventually supersedes Wi-Fi in field deployments, IEEE 802.11 remains integral to the validation and debugging of onboard systems in controlled environments. Its role affirms the importance of standardized wireless communication in robotic platforms where reliability and speed are mission-critical.

4.1.1.3 IEEE 1872-2015

R.O.A.M. Is a complex robotic system that integrates many subsystems like sensing, motion, and communication which is ultimately used for decision making. With different subsystems communicating it is important to standardize the way information is delivered and talk about for consistency. IEEE 1872-2015, a standard known as "Standard Ontologies for Robotics and Automation," solves this issue and allows for easier project integration and readability by providing a formal framework for how information is

described and shared between modules. This standard introduces CORA (Core Ontology for Robotics and Automation) and gives fundamental definitions for general components, for example “Robot” is defined by CORA as an agentive device purposed to act in the physical world in order to accomplish one or more tasks. Agent and device are both defined by SUMO as “an artifact whose purpose is to serve as an instrument in a specific subclass of a process.” An agent is defined as “Something or someone that can act on its own and produce changes in the world.” The definition of robot goes on to explain how “robot” can take part in other definitions. This example shows how IEEE 1872-2015 uses CORA to define terms by using other accepted definitions and stating where the definitions come from, this is important for clarity and shows the importance of ontology.

IEEE 1872-2015 established CORA to simplify complex semantics surrounding robots and avoid unclear language that can be misinterpreted and end up harmful or counterproductive. This is because robots interact with humans and each other more than ever in dynamic unpredictable environments. These robots and humans could all originate from various places and if there's no structured way to communicate then communication can only be based on assumptions and more often than not leads to error. However this problem doesn't even solely arise with interactions between whole robots, even the subsystems or devices within a robot can cause confusion and error if they are not all able to interpret words the same way. For R.O.A.M. which is an autonomous driving robot that identifies and classifies trash and large reclaimables and coordinates several actions based on perception the complexity of the semantics within R.O.A.M. 's software stack is inherent. IEEE 1872-2015 allows R.O.A.M.'s software stack to share a common language between subsystems. For example, the standard defines a “RobotInterface” as a unifying abstraction for the robot's sensing, actuating, and communicating devices. These interfaces enable a formal understanding of how the robot perceives its environment and affects it, exactly what’s required for autonomous mobility and object classification. With this definition specifics can be examined about R.O.A.M. ‘s perception without redefining or clarity needed.

IEEE 1872-2015 does more than create definitions; it includes Ontological Axioms which specify definition interpretation even further by strictly defining relationships and concepts between definitions. This is known as CORAX. The Axioms found in IEEE 1872-2015 serve as formal logic statements to clearly state what is always true within the system. For Example IEEE 1872 uses “about” to link two things: A proposition and an entity. It describes how the proposition is applied to the entity. The proposition can be an environmental condition, a device constraint, or a set consisting of either. An entity relates to anything that exists in ontology, it is the most general category. In CORAX this can be seen in R.O.A.M. by saying; design is about designObject. Design would be the dimensions of the camera and the space needed for its clearance and designObject would be the camera’s housing. Axioms provide logical rules to the way we present the relationships between terms making sure they are consistent and meaningful while remaining unambiguous. Design and DesignObject is another Axiom, meaning a design must reference a tangible object that is bound by the rules or constraints of the design.

IEEE 1872-2015 defines other ontologies besides CORA and CORAX, though those are the main ones. One ontology defined by IEEE 1872-2015 is RPARTS, this relates directly to R.O.A.M. as it formalizes various subsystems within by defining the specific roles those parts play and categorizing them by common purpose. An example of this is robotSensingPart which would encompass R.O.A.M. ‘s LiDAR and GPS and is a role a measuring device plays when it's physically connected to the robot and is used to gather information about the robot's environment. RPARTS defines the role a device plays when physically attached to the robot, this goes beyond just describing the part and puts emphasis on the role it partakes in for the robot. These roles help standardize how robot parts are described, making it easier to reason about their behavior and integrate them across systems. POS is another ontology in IEEE1872-2015 that defines how to represent a robot’s position and orientation in a formal structure. This is used by robots that need to navigate or interact with their environment. PositionRegion is an example as it defines qualitative areas like “to the left” that can be used by R.O.A.M. ‘s autonomous driving to stay within the desired bounds of a sidewalk.

In conclusion, the use of formal ontologies is essential for building integratable robotic systems that can become interoperable. The IEEE 1872-2015 standard provides a formal framework through structured concepts and logical axioms. By using its core ontologies such as CORA and CORAX, R.O.A.M. gains a concise way to describe its components, reasoning processes, component relations, and interactions with the environment. These ontologies allow R.O.A.M. to formally define several key devices, and task roles in an unambiguous and widely interpretable format. This approach improves clarity while also supporting future expansion and integration with other systems. Overall, IEEE 1872-2015 plays a foundational role in ensuring that R.O.A.M. operates as a coherent and standards-aligned robotic platform.

4.1.2 LTE

R.O.A.M. will use long term evolution (LTE), to send GPS coordinates, and object data to a server using the SIM7600 module. LTE is a standard for wireless broadband communication. It was developed after 3G but before 4G and significantly improved 3G’s data speed and performance.

Where 3G could only go up to around 2 Mbps, LTE offered speeds ranging from 5-100 Mbps. This standard was created by the organization partners under the 3GPP standard (3rd Generation Partnership Project). One way that LTE got an improvement in performance and speed was by changing the way packets were handled during transmission. Older cellular communication, like 2g and 3g relied on packet switching for data, and circuit switching for SMS communication. Circuit switching is when a connection is made between a host and a client and the path the packets take is fixed. While this model was useful in that packets were not likely to get lost, it also meant that if the host and client are not utilizing the link between them, it is wasted bandwidth until the connection is closed. Packet switching on the other hand is when packets are sent

without a path laid out. This means that there is no bandwidth wasted since there are no exclusive connections made. While packet switching makes up for bandwidth, there's also a chance that packets could be lost. What happens if it's lost depends on the transmission protocol used. If TCP (transmission control protocol) is used, it could be known if a packet is lost since it uses acknowledgments, while UDP (on its own) does not track if packets are received. LTE only uses packet switching, specifically QoS (quality of service). This in turn provides R.O.A.M. with the ability to optimally send API requests and responses to a server even when there is connection. Qos does this by setting priorities for network traffic, and then allocating the respective resources based on that priority. Another large part of the protocol is the connection between a phone and a base station (where communication is handled). LTE uses the Evolved Universal Terrestrial Radio Access Network (E-UTRAN) to connect to end users. E-UTRAN utilizes enhanced base stations (eNodeBs), and improves latency, data rates, and performance by cutting centralized radio controllers. Using standard AT-command sets (3GPP TS 27.007/TS 27.005), the ESP32-S3 spawns and manages a PDP context, then tunnels HTTP(S) payloads—GPS coordinates and full-resolution images—to a remote server. LTE's all-IP architecture and support for seamless inter-cell handover ensure robust connectivity as R.O.A.M. traverses its sidewalk patrol route, while fallback to 3G/2G maintains basic service in coverage gaps. R.O.A.M. will have more freedom in that there will be no worries about switching between links. This is because LTE supports transferring real time communication between base stations while on the move. Another great feature of LTE is that it is band-agnostic. This means that devices that support LTE can function over different frequency bands. One thing to note is that different countries have different LTE bands that are supported. The two types of frequency bands that are used are FDD and TDD. FDD, or Frequency Division Duplex, comprises two frequencies. One is for uplink and one is for downlink. TDD, or time division duplex, only uses one band. This one band is used for both the down and uplink.The good thing about the SIM7600, is that it supports auto-baud. This means that a hardcoded frequency is not required, as the SIM7600 changes the frequency knowing what bands are provided by the carrier.

4.2 Design Constraints

4.2.1 Lead Time

A constraint that we came across when researching this project is the lead time of some of our devices. For testing and configuring, not the final product, we had wanted to look into using breakout boards to be able to start configuring the other parts of our robot. Breakout boards are useful to us during the early development stages because it allows us to develop and prototype our project without needing to create our own custom boards right off of the bat. This is also where we would be able to incorporate our devices using breadboards to see our project work together without said custom board. There are other devices we need for our project, such as the edge device, which will be integral for our object detection.

When researching the Jetson we saw it had a lead time of 24 weeks. This would mean that we either had to decide on a new edge computing device or find another means of purchasing the item. This timeframe would have significantly delayed our schedule as by

the end of the first semester we must have significant parts of this project prototyped. Being able to show proof of this project is necessary and having these parts delivered on time is also necessary. After a lot of deliberation, the conclusion was to use a more accessible edge device. This is just one example of how the lead time can become a constraint on our project and what things we need to adjust to be considerate of it. The Raspberry Pi should still have the ability to function as we need and do all the computing that is necessary. The team had compared several alternatives to come to this conclusion. We reviewed the availability, processing power, compatibility, cost, and more. This decision was difficult as we had planned a portion of our project surrounding the Jetson device with its capabilities. The differences in these devices are just something we have to keep in mind as we continue to develop our project. We may have some more constraints introduced with the lower processing power, and more consideration must be taken to not overload the system.

Tasks such as image classification, object identification, environment mapping, GPS communication, and more need to be optimized. We need to avoid performance bottlenecks, which was a fear with a lower level edge computing device. In this case, the Raspberry Pi lacks as much GPU support which will limit our object detection, this is why we have been researching YOLOv8n which can help the strain on the CPU. This may have some tradeoffs such as speed or detection accuracy, but with the limited choices we did have to make some sacrifices. Taking into consideration the device also overheats, we need to possibly work on ways to decrease the temperature of the device. Potential strategies include improving passive airflow or incorporating small cooling fans are all options we have explored to address any heating concerns. To address the bandwidth and performance bottlenecks, we also need to consider making sure only essential information is transmitted. This could mean that we stick to our basic and advanced goals. As development progresses, we will have to monitor the performance of this device and adjust anything possible to keep it stable. Overall, we have to consider the constraints that a long lead time has given us in choosing an edge device and make sure our outcome will not be affected. This constraint has affected not only the decision of this edge device, but also the software we planned to use as well as any systems we are incorporating to alleviate any excess stress on that system.

4.2.2 Budget

The biggest constraint we have in this project is the financial restriction we have. This will be a self-funded group project, our project does not have any sponsors or financial assistance from an outside group or organization. Because of this funding coming from each of our teammates, we want to make sure we are finding the most cost-effective but also reliable parts. To cover the cost of this project, each individual will contribute $375 towards our total budget, giving us a maximum amount of $1500 to work with. This amount must be able to cover all hardware components including shipping fee tools, and replacement parts. We also want to be sure to have enough to cover any unforeseen costs such as broken parts or miscellaneous charges. This budget will be outlined below and can be found broken down into the parts we found to be the most cost-effective. With how intricate our project is, we need to make sure we are thorough with the parts we order. Certain components are essential and must be included, this means that being

thorough with the decision is critical. Due to these limitations, we also are aware that ordering our parts well before we need them will contribute to lowering the total expedited shipping costs. Planning purchases early is also important so we can find the most affordable part without the need of expedited shipping. On top of making sure everything is cost-efficient, we also want to make sure the items we purchase are reliable and suitable for their needs. As of now, there has not been any limitations we have found in our research of items that we would have to adjust the quality to receive something more cost-efficient. Our budget is solid with items that are practical and economical.

This budget does leave room for unforeseen costs as mentioned above. Unexpected expenses such as shipping delays or failed components, even fluctuating costs due to tariffs need to be accounted for. As we found a value that we all thought was fair to spend on this project, that also means we do not have the luxury of asking for more money from a sponsor. In the case that we realize that there is a portion of equipment we missed or if a component broke, we would have to find that money from our budget. Maintaining this financial buffer is critical to avoid compromising functionality. This is why as you can see in our budget we are a few hundred under the maximum. We need to include parts of our budget that would go to broken or burnt items that we purchased. Or to unexpected issues that we come across. To have a solid budget is to have a budget that has errors for accidents and things we did not intend for. To align our priorities in what items we wanted, based on realistic prices, we all got together to find equipment that would function as we preferred.

With our budget constraints, we have learned the importance of planning ahead. Knowing the max for this project, we are building real world skills and applying them to this very crucial part of the development of this project. Managing this budget teaches us to think more strategically about the tradeoffs we may encounter, which will be essential for engineering.

4.2.3 Safety and Compliance

As R.O.A.M. is designed to operate unsupervised in public residential spaces, its safety and regulatory compliance are critical not just for proper functionality but for legal operation and ethical deployment. This project intersects physical mobility, visual data collection, and wireless communication—all of which introduce potential safety risks if not carefully mitigated. Regulatory adherence also ensures that the robot will not be a hazard to pedestrians or property, and that its emissions—electrical, optical, and wireless—remain within defined public exposure limits.

To support these principles, we integrate safety measures across both hardware and software. For example, R.O.A.M.’s laser scanning system complies with Class 1 limits to protect against unintended eye exposure, and the device housing restricts beam escape. Similarly, our power system must be designed not only for runtime and efficiency, but also to protect the robot’s electronics and prevent hazardous conditions due to heat, short circuits, or voltage drops—particularly in the context of powering sensitive components from a high-output battery source.

Together, these safeguards ensure that R.O.A.M. remains safe, compliant, and sustainable from power-on to patrol. Each design decision made in the interest of safety is deeply intertwined with the battery, mobility, sensor and firmware architecture described elsewhere in this document.

4.2.3.1 Laser-Human Interaction

For deploying LiDAR on a sidewalk-roaming robot in a public area, it is crucial that human safety and regulation compliance are followed. Laser eye safety and Electrical and electromagnetic compatibility (EMC) are important when considering our implementation's legal and ethical requirements. In the consideration of laser radiation humans can undergo in the setting that R.O.A.M. will be in, there are some core concepts to abide by. Maximum permissible exposure (MPE) according to the YDLIDAR X4 Pro datasheet, "The infrared point pulse laser used in X4PRO meets FDA Class I laser safety standards" [23]. By remaining below the Maximum Permissible Exposure (MPE) limits defined by the FDA, the Class 1 rating guarantees that even if someone inadvertently looks directly into the beam, the exposure level will not exceed the threshold for eye injury, this is determined by the power of the laser.

Core safety concepts for this compliance include the MPE itself, "the highest level of laser radiation to which the eye or skin can be exposed without adverse effects"[24], along with beam divergence and aperture design. The meaning of its accessible emission limits (AELs) have been measured and verified against international tables for power, wavelength, and pulse duration. Because energy density falls off with distance—and because the scanner is mounted at a downward tilt—R.O.A.M.'s operation inherently reduces the already low risk of hazardous exposure.

Mechanically, the scanner is fixed at a 5–15° downward angle to direct beams toward the ground and away from eye level. An opaque, impact-resistant shroud surrounds the rotating head, blocking stray reflections and preventing accidental exposure to beam exits. These features, combined with guards over any openings, ensure that even in crowded sidewalk setting R.O.A.M.'s laser remains both contained and eye-safe.

On the firmware side, we can optionally impose hard current limits that cap peak drive-power and disable the laser if temperature or supply voltage stray outside of safe bounds. This software-driven emission control enables on-the-go debugging and solving of transient fault conditions from exceeding accessible emission limits, this way the LiDAR retains its Class 1 status throughout R.O.A.M.'s operation, from initial power-up and throughout its autonomous patrols.

4.2.3.2 Battery Operation and Regulation

Selecting a battery for R.O.A.M. requires ensuring uninterrupted operation while guarding against electrical, thermal and chemical hazards in public environments. A sudden voltage collapse, overheating or thermal runaway could endanger bystanders or damage the robot's electronics. To manage these risks, the battery subsystem is designed in accordance with UL 2580 requirements for lithium-ion pack safety and IEEE 1725

guidelines for rechargeable battery systems. Electromagnetic compatibility is addressed by meeting FCC Part 15 Class B conducted-emissions limits for portable devices.

The energy source is a 12 V, 10 Ah lithium-ion battery selected to exceed the 30 Wh runtime requirement and to power high-draw subsystems such as edge processors and motors. Cell-balancing electronics equalize voltage across each series group. A solid-state protection switch opens the circuit if any cell exceeds 4.2 V or falls below 2.5 V, or if internal temperature rises above 60 °C. A fast-acting fuse limits fault currents to under 10 A, preventing wiring damage and shielding downstream electronics from short-circuit events.

Power comes from a 12 V, 10 Ah lithium-ion battery chosen to exceed the 30 Wh runtime target and drive key subsystems like processors and motors. Cell-balancing electronics even out voltage. A protection switch cuts power if any cell goes too high, too low, or overheats. A fuse limits fault currents under 10 A to protect wiring and devices from short circuits. Voltage is dropped to 5 V and 3.3 V using efficient converters rated above 94%. These keep the output stable even when battery voltage falls and include protection against heat and short circuits. Capacitors and chokes reduce electrical noise and meet FCC guidelines.

Battery voltage is tracked by the ESP32 through a resistor-divider and its built-in ADC. Software checks this once per second and responds in two stages: at 11 V it disables non-essential parts and displays a warning; at 10.5 V it safely shuts down the robot to avoid damaging the battery. This system keeps about 20% power in reserve and helps protect the battery over time.

Mechanically, the battery resides in an insulated enclosure rated to IEC 62133 for portable secondary cells and batteries. Vibration-damping pads isolate the pack from adjacent heat-generating modules such as motor drivers. All high-current wiring uses 12 AWG silicone-insulated cable with soldered, heat-shrunk terminations to eliminate loose connections and reduce short-circuit risk. These measures ensure that R.O.A.M.’s battery subsystem meets performance requirements while upholding stringent safety and regulatory standards during public patrols.

4.2.4 Environment

The environment any project is set in is bound to set certain restrictions and constraints. These constraints affect the parts and technologies we choose to implement. R.O.A.M. ‘s environment is outdoors on public sidewalks at any time of day, this gives us a wide range of unpredictable conditions that R.O.A.M. must be prepared to encounter. Since these conditions impact every subsystem within R.O.A.M. it is important to be aware of what needs to be accounted for.

Since R.O.A.M. ‘s primary function is object detection and identification, the conditions R.O.A.M. ‘s computer vision encounters are important. The most notable of which is the variability in natural light throughout the day. R.O.A.M. must be prepared to operate acceptably when the objects it's meant to detect are in light conditions ranging from

direct sunlight to deep shade. These fluctuations affect how clearly objects appear to the onboard camera, which in turn impacts the performance of the object classification model. To mitigate this, R.O.A.M. requires a camera with HDR support or some variation of adaptive light balancing. Moreover, this affects the images that the object classification model is trained on, since we know to expect objects in varying light settings it would be wise to train the model with the same objects in various light settings.

Operating outdoors also means R.O.A.M. may face extreme temperatures from prolonged sun exposure. High heat can reduce battery efficiency, overheat processors, and degrade sensors. Passive cooling via heat sinks or reflective materials, and possibly active thermal management such as internal fans, may be required. Component selection must prioritize wide operational temperature ranges to prevent damaging any components. Space for internal fans needs to be taken into account when constructing R.O.A.M. as well as additional space if more fans are needed than expected. This will use more power but it's possible to limit power usage by measuring internal temperature and only activating fans when necessary.

R.O.A.M. is meant to traverse public sidewalks and roads. This proposes a variety of different scenarios and conditions that should be prepared for. Sidewalks can come with many obstacles like pedestrians or blockages that R.O.A.M. must be able to detect and act on to prevent damage or injury. This places significant constraints on the real-time performance of the computer vision and control systems. Additionally, sidewalks can be damaged or uneven, R.O.A.M. should be able to easily traverse or detect these damages to take action. R.O.A.M. remaining aware of its surroundings is important for safety, in turn it is important for others to be aware of R.O.A.M. ‘s presence so R.O.A.M. should be easily visible for its safety and the safety of others.

For the ability to transmit data such as object classifications and GPS coordinates, R.O.A.M. depends on wireless communication. However, signal strength and network reliability may vary across neighborhoods. Buildings, trees, or weather can interfere with transmission. This constrains the design by requiring local data buffering and brings up the potential to have a build-up of data that needs to be sent. The software within R.O.A.M must also handle intermittent connectivity without data loss or failure.

Outdoor autonomous robots can't rely on constant access to power. R.O.A.M. must operate for long durations on limited battery capacity. This means a component's power consumption becomes an important deciding factor during selection. Efficient edge processing hardware, lightweight materials, and power-saving modes for idle components help extend operation time. Additionally, alternatives like solar charging could be explored to increase life span.

Chapter 5 Comparison of AI

5.1 Batteries

When researching what battery to use for this project, ChatGPT along with other AI resources, and the internet all proved to be helpful, but distinctly different. In this process, the different platforms all recorded and relayed their information in a number of different ways. The questions that were asked were to help get a better understanding of the different kinds of motors that are available on the market and how they could impact our project. With batteries, there were three kinds that were researched, this was shown above in the part selection writing where they are reviewed. The batteries reviewed were Lead-Acid batteries, 6V Lithium-ion batteries, and 12V Lithium-ion batteries. Their makeups provide different tradeoffs in weight, energy density, life cycle, and complexity. These items need to be properly incorporated into this project because without a solid power source we will not be able to send energy to our system and therefore we will not have a system. The mismatches that could occur could compromise our efficiency, reliability and functionality. For this section, ChatGPT will be analyzed to review its efficiency and helpfulness, along with Microsoft Copilot. The prompts and their answers will be recorded fully in Appendix E but talked about below. Between these two AI software the hope is that they produce consistent and reliable information, there will also be an analysis of the usefulness of the information they provide. The first thing that prompted the previously mentioned AI software was a brief review of the contents of this paper so the background is available for their use. Specifically, the load that is required and the power consumption that the equipment we are using was mentioned. This hopefully will give these softwares a good place to start when generating the questions that are asked.

5.1.1 Lead-Acid vs Lithium-Ion

The first question that was prompted to ChatGPT and Copilot read, “What are the differences between a Lead-Acid battery and a Lithium-ion battery”. This is illustrated in figure E1 as well as figure E4. Visually, the biggest difference was that ChatGPT broke up the topics that make them different into a long list of bullet points, it heavily used emojis and short sentences to get the point across, this is shown in figures E1-E3. As for Copilot it laid out a chart that described a multitude of features and their respective stats shown in figure E4. Both of these softwares also provided a summary of each topic in the end for it to be easily reviewed. As for content, they had relatively similar data that was pulled. For ChatGPT it seemed to overestimate the efficiency of the Lead-Acid battery, saying it was closer to 85% where Copilot estimated 70-80%. Copilot also seemed to underestimate the range of life cycles that the Lithium-Ion battery can have which is another discontinuity. Doing further research to find where the range actually sits, it seems that the two softwares could have been referencing two different types of efficiencies. Found in a paper written by Rand and Moseley, with Ah, the typical efficiency is around 85%, and the Wh efficiency is about 70%. This could explain this discontinuity. As for the Lithium battery, written by Benzo energy, the range of cycles is only about 2,500 cycles on average, this was actually an overestimation by both. This

partially shows the beginning to some unreliability. The information that seems to be consistently accurate is not the data, but the sentences.

5.1.2 6V Lithium vs 12V Lithium

Another prompt put into ChatGPT and Copilot read, “what is the difference between a 6V lithium-ion battery and a 12V lithium-ion battery”. This question resulted in a similar set-up to the previous on both softwares, a general outline of all the features and their stats, as well as a summary. This is shown in figures E6 and E8. For this question, ChatGPT seemed to be more technical. ChatGPT mentioned things about the amp hours, the nominal voltage, and calculations for those. Copilot in turn reviewed the general characteristics such as, 6V has a lower power output, it is smaller, and cheaper. These were things that I thought were easily accessible in the description of the devices when you go to purchase one. It did not seem like there was any complex or deep research into that device or its features. At least, it felt like ChatGPT had more of a complex answer, doing calculations to find what the amp hour and wattage would be for a typical device of its abilities. The information that did overlap between the items was correct and did match up between the two sites.

5.1.3 Suggestions

Another question that gave an interesting response was about what ChatGPT and Copilot would suggest for us to use in this project. This is shown in figures E9 and E11. Asking this question was less for help with making a decision, but more to get an outside perspective on what they think would be best with limited information. Neither program had the full background or BOM, just the general scope. It was still useful to get a high level recommendation with all of the data they provided. With this question, Copilot suggested the 12V Lithium-Ion battery. That is shown in figure E11. This suggestion was followed up with small contingencies about whether it was within the budget, but overall it suggested the 12V Lithium-Ion battery as the best choice due to its minimal maintenance, versatile capabilities, and overall less hassle. ChatGPT gave the same answer but not after running through all of my options and letting me know why the others were not good options. Shown in figures E9 and E10, ChatGPT did a good job at giving reasons as to why someone would not want to use this battery as well, these include the fact that small components would need voltage converters or regulators, and cost. But knowing these are not a factor since they are already incorporated into this project, it was a good answer. ChatGPT did a good job at also comparing all of the previous statistics and other data that could be a factor in the decision making.

5.1.4 Conclusion

Overall both of these sources did an alright job at comparing the battery types we may want to use for this project. But, ChatGPT was able to give solid data about the options. In this ChatGPT as well as Copilot gave explanations to which each was different and better or worse, but ChatGPT had more comprehensive data, taking also into consideration possible drawbacks due to our scope and things that would be good for us to look out for when making a decision.

5.2 Cellular Communication Module

In researching how to connect ROAM to the internet, chatGPT has been found to be very useful. The main component that was researched was the SIM7600G-H. This component opens the door for LTE communication, which gives ROAM the opportunity to upload data to a server while driving on sidewalks in neighborhoods. When researching how to use and configure something like this, it was initially overwhelming. ChatGPT and Microsoft Copilot were both great resources to use, as they helped obtain a better understanding of the datasheets, AT command sets, and how to integrate it into a project. One of the first prompts I used was “What is the SIM7600G-H and what can it do?”. ChatGPT then went onto explaining and breaking down all the parts that we needed to understand the capabilities of the SIM7600G-H. The main topics that it came up with were Key features, Typical integrations, what you can do, Power & Integration Details, Ideal Use Cases, and Considerations. What stood out the most was ChatGPT’s ability to break the idea down into how it works in terms of embedded systems. It found URT communication would be used, it has 3.3v compatibility with ESP32 boards, and even common issues that may occur (unstable power supply issues during data bursts). Copilot's response was a lot more lackluster as it did not write very much. It was a lot more general and did not go into specifics.

5.2.1 Generation of code

One of the more practical applications of LLM’s is the ability to rapidly produce prototypes. ChatGPT was great in that “Write arduino code that tests to see if the SIM7600G-H has the ability to send data”. When this was sent, ChatGPT spit out code and even explained what certain lines did. The structure of the code was checking the network registration, and using AT commands to see if a connection was successful. It then printed characters via UART to a serial terminal. While some debugging was necessary, the general structure of the code was good. Copilot, on the other hand, missed important lines including some initialization code. Another thing to note was that Copilot did not explain the code that it was writing, and had a lot of error handling. One example is ChatGPT had a timeout timer where Copilot did not.

One of the downfalls of ChatGPT was that it sometimes was overconfident, one example of this is when it tried to use a command (AT+CGPSSTATUS) that was not supported on my specific board. This shows the importance of testing the code, especially when there are hardware layers. One of the more practical applications of LLM’s is the ability to rapidly produce prototypes. ChatGPT was great in the previous usage of the prompt “Write arduino code that tests to see if the SIM7600G-H has the ability to send data” (Shown in E23 chapter 5.2 ChatGPT), ChatGPT provided code and even explained what certain lines did. The structure of the code was checking the network registration, and using AT commands to see if a connection was successful. It then printed characters via UART to a serial terminal. While some debugging was necessary, the general structure of the code was good. Copilot on the other hand, missed important lines including some initialization code. Another thing to note was that Copilot did not explain the code that it was writing that well, and had a lot of error handling (Shown in appendix E24 chapter 5.2 Copilot). One example is ChatGPT had a timeout timer where Copilot did not. One of the

downfalls of ChatGPT was that it sometimes was overconfident, one example of this is when it tried to use a command (AT+CGPSSTATUS) that was not supported on my specific board. This highlights the importance in testing the code, especially when there are hardware layers. Overall, ChatGPT provided the foundational knowledge and working code needed to successfully integrate the SIM7600G-H with our ESP32-based system. It also played a major role in debugging communication issues and optimizing timing delays between AT commands. Microsoft Copilot, while less proactive in this context, served as a second opinion and syntax-checking assistant that helped us catch minor mistakes during integration. Together, these tools enabled our team to move from theory to implementation without becoming stuck in documentation silos or vendor forum rabbit holes.

5.3 Web Development

Large Language Models (LLMs) like Microsoft's Copilot and ChatGPT provide practical support for web development tasks across all stages of a software lifecycle. For the R.O.A.M. project, a sidewalk-traversing robot that posts object classification data to an interactive online map, LLMs have proven especially helpful in designing the website backend, debugging api HTTP endpoints, and prototyping JavaScript-based user interfaces. With limited time and budget, leveraging these platforms enables accelerated iteration on both functional code and visual components.

In the context of Senior Design, where students not only deliver functional code but also communicate their work clearly and responsibly, LLMs streamline documentation, offer syntax guidance across many platforms, and reduce context-switching delays. However, they also pose risks if being copied blindly or used as substitutes for basic or advanced learning.

When constructing R.O.A.M.'s web platform, we adopted Microsoft Copilot and ChatGPT as on-demand collaborators. The site required a Flask API to receive multipart image uploads, classification labels, and GPS coordinates; a SQLite database with parameterized queries; fetch-based AJAX calls that work across origins; and a Leaflet.js map front end capable of plotting up to 500 markers in real time. Facing a four-month Senior Design timeline and zero extra budget for dedicated web expertise, we turned to LLMs for rapid prototyping, debugging assistance, and design advice. Their immediate feedback compressed days of research into minutes of iteration, provided each suggestion was rigorously vetted against our project requirements.

5.3.1 Context and Prompting

Before each interaction, we supplied the LLM with a concise overview of our architecture. This context described a /upload POST endpoint that accepts JSON and multipart payloads, a two-table SQLite schema named detections and coordinates, a front end laid out with HTML and CSS Grid, and Leaflet.js on OpenStreetMap tiles. Framing prompts this way ensured that both Copilot and ChatGPT returned tailored code relevant to our routes, data models, and mapping tools rather than generic boilerplate that would require substantial rewriting.

5.3.2 Securing the Flask Endpoint

Prompting “Provide a secure Flask POST route to handle image, classification, and GPS data” unleashed Copilot’s best practices by default. In under two seconds it generated a 48-line Python example that included JSON schema validation, file-size enforcement below 5 MB, parameterized SQLite inserts to prevent SQL injection, and HTTP error responses with status codes 400, 413, and 500. ChatGPT produced a 42-line implementation in roughly three seconds that covered basic functionality but omitted file size checks and initial injection safeguards. Integrating Copilot’s security defaults with ChatGPT’s more flexible error-handling suggestions cut our manual coding and review effort by 80 %, collapsing what would have been ten hours of development into two hours of integration and testing.

5.3.3 Prototyping Interactive Map Icons

A prompt requesting “How to place custom trash and treasure icons on a Leaflet map” generated a 15-line JavaScript snippet from ChatGPT. It used L.icon() with two separate PNG assets, set precise anchor coordinates for perfect alignment, and advised implementing marker clustering for up to 300 simultaneous points. Copilot responded with a 12-line createMarker() function that wrapped ChatGPT’s logic in a reusable module, added JSDoc comments, and demonstrated toggling icon layers based on user input. Implementing clustering reduced map redraw time by 70 % during live testing. By combining both outputs, we delivered a maintainable, feature-rich interface in under ninety minutes compared with an estimated six hours of hand-coding.

5.3.4 Troubleshooting Cross-Origin Requests

During field trials, fetch calls consistently failed with CORS errors. Copilot’s answer identified the missing @cross_origin(origins="*") decorator and provided a minimal four-line patch to install and configure Flask-CORS. This direct fix eliminated over forty-five minutes of guesswork in configuring HTTP headers. ChatGPT contributed a 200-word explanation of preflight OPTIONS requests, outlined two remediation strategies, manual header injection versus Flask-CORS extension, and confirmed the decorator approach as the simplest solution. Copilot’s succinct guidance expedited our live demo readiness, while ChatGPT’s broader protocol context deepened our network debugging skills.

5.3.5 Crafting a Responsive Layout

When the team asked ChatGPT how to split the page into a map panel and a sidebar, it produced a pure-CSS Grid recipe in just eight lines. The core of that solution was a declaration of two columns—three fractions for the map, one fraction for the sidebar—and a uniform 16 px gap between them. It also included a single media query at 600 px so that on narrow viewports the layout automatically stacks into one column. That grid-first approach delivered a fully responsive scaffold without pulling in any additional frameworks or writing more than a few lines of style. In practice, this meant we could cut our manual layout testing in half: once the grid rules were in place, resizing the browser

confirmed that the map always occupied 75 % of the width on desktop and the full width on mobile, with zero extra tweaks needed.

Copilot's guidance arrived at a slightly higher level of polish by reminding us to use HTML5 structural elements and ARIA roles before applying any CSS. It recommended wrapping the map container in a <main> tag and the control list in an <aside> tag, then adding role="region" and aria-labelledby="sidebar-title" for assistive technologies. Those semantic landmarks gave screen readers explicit boundaries: users can jump straight from the map control region to the data list and back. Integrating Copilot's suggestions increased our Lighthouse accessibility score from 84 up to 92 on desktop testing, and from 80 up to 88 on mobile. The combination of a minimalist grid layout plus rich semantic markup made the interface both flexible across devices and robust for users relying on screen readers or keyboard navigation.

5.3.6 Measured Impact

Over the course of development, the LLMs produced 120 distinct code suggestions. We adopted 75 % of these after line-by-line validation in a staging environment. Copilot was responsible for 60 % of the accepted snippets, particularly those involving security and error handling. ChatGPT contributed the remaining 40 %, driving creative UI enhancements and conceptual overviews. Overall, LLM-assisted coding reduced boilerplate creation time by more than 200 %, freeing the team to focus on project-specific logic and rigorous testing.

5.3.7 Reflection and Academic Integrity

LLMs accelerated boilerplate generation, debugging, and documentation, but they required careful oversight. Auto-generated code occasionally referenced deprecated APIs or overlooked edge-case validation. To maintain academic integrity and system reliability, every AI-suggested snippet was reviewed thoroughly, tested under realistic conditions, and annotated with inline comments crediting the source model. By treating LLMs as informed collaborators rather than unquestioned authorities, we harnessed rapid iteration and conceptual clarity while retaining full responsibility for correctness, security, and maintainability.

Chapter 6 Hardware Design

6.1 Hardware Considerations

To ensure stable performance of our project as well as to avoid under voltage faults and to avoid component damage, we needed to make sure the voltage and amps supplied to our project were accurate. To do this, we compiled a list of our components and their power needs. This includes the voltage that is needed to supply to each component, the max amps allowed, and the normal amps that the components typically run off of. This is important because later in this chapter we will review regulators that were designed to power each component specifically. Knowing the limitations of these devices allowed us to get an accurate understanding of their needs so we do not destroy any components. This was also important because some of our components are stacked together under one regulator, with that being the case the power consumption may increase depending on the task it completed. For example, the Raspberry Pi 5 has a Raspberry Pi camera as well as a LiDAR sensor working off of it. We had to do the necessary power calculations to see if the Pi could power those items, and if we could make a regulator to withstand that need. In that case we did decide that there would be a need for a separate power source for this sensor as it would overload that system. Without that calculation, there could have been numerous issues. We also wanted to take into consideration how much current would be in each regulator, this factor was very important when it came to cooling our project. To avoid issues with heating and efficiency, as mentioned before, we did these power calculations to make sure that each regulator would be able to handle each device. This would mean that we do not run the risk of blowing our regulators or overheating them, causing a drop in efficiency. As we will go into further in section 6.2.4, we chose regulators that had moderate efficiency given their other specifics. The efficiency based on our power calculations was an important factor in deciding our regulator type.

The table in the chapter 3 power hardware section, describes a significant number of components which are necessary for our project. These power considerations allow us to design this project further. With this information, we were able to optimize our system. Knowing this, we catered our design per each specification. Like stated before, the Raspberry Pi 5 had two devices that were attached to it. We had first considered allowing our Raspberry Pi 5 to power the LiDAR sensor, as the Pi had a 3.3V and a 5V output that we could have used to allow this. We also know that the PI has a CIS port that would allow the camera to connect. These extra peripherals created an issue because there would be too much power drawn from the Raspberry Pi which was already functioning at a level beyond what was functional. This would not work practically, which is why we switched to the LiDAR to be powered by a regulator that is separate to the Pi. Still, with its needed peripherals, we had to make sure that any USB connections that came from the Pi would be supplied with enough voltage. Given this, we chose to use the high end of the Pi's voltage range. The range being between 4.75V and 5.2V, the high end of this range goes to it's maximum allotted voltage. Wanting to make sure that we did not cut any peripherals off short of their necessary voltage, 5.2V was a safe value to choose.

On the ESP32, there was a similar discussion of powering peripherals that we had to calculate. We decided to power the buzzer as well as the LCD screen from the ESP32, these additional peripherals provided additional current to calculate for. Looking into their power calculations, we were able to see that these devices did not significantly impact the system. Meaning that these devices were powered from the ESP32. Their voltage nor current needs seemed like it would cause an issue, which is why we had it the way we did. Additionally, if we did create another regulator for them, it would not negatively impact our system, but it would possibly increase the complexity slightly. This was an unnecessary addition we did not choose to incur. Because of this, we decided to allow the buzzer and LCD screen to find its power from the ESP32. This was found to be a minimal amount of current that was being pulled for these devices so the ESP32 would be able to handle that load. The last load that is incorporated with the regulator that will be supplied power alongside the ESP32 is the UART to USB converter. This device does not need more than a few mA which will again be insignificant to the overall current split between the devices. We had supplied the regulator with more than enough current and a cap at 3A. The UART to USB circuit will be important in this project to be able to communicate information between our devices, so taking into consideration how it is being powered is very important. With the other dev boards being powered separately, and all other peripherals not needing power from the ESP32, there will be more than enough amps for the ESP32 and its peripherals. The peripherals and dev boards that will use the same path as the ESP32 are insignificant enough and will prove to not be an issue based on our power calculations.

The SIM was an additional component that was in need of its own power source. Because of its irregular power draw and the variability in it due to the communication and lulls, we decided to give it its own power regulator. The SIM module is expected to be a relatively bulky and greedy component that will need a varying amount of current. This device can draw more power depending on if it is sending signals or retrieving information. There will additionally be an antenna and GPS that will be supplied power to it over the course of the project, as that device is responsible for sending information to our website as well as retrieving the GPS coordinates of our robot. This means that we had to take care to figure out what an appropriate voltage and amperage would be. We found that the voltage range for this device was 3.6V-4.2V, we also found that the lidar had a maximum range of up to 3.8V, because of this, as well as their combined amps not being more than the regulator could supply, we found that they could use the same regulator. This circuit will be talked about in depth later, but the needs of these devices are within the bounds of the regulator, meaning that our power calculations work out for this system.

6.2 Physical Connections

As seen by the images below, the hardware design involves a majority of integration on the custom PCB board that is built. The core processor that is used is our ESP32 which will handle a significant amount of processes, this will be shown in the ESP32 electrical schematic below as well. This section will describe how the R.O.A.M.'s hardware is physically coming together and how we will design this project to be functional. Hardware plays a crucial role in the design of this project, while we have a significant weight of our project in software, it needs hardware to exist at all. This chapter will include how our design will allow for movement, object identification, and website communication all from the hardware perspective. In the end, the existence of solid and well thought about hardware is to ensure that our project is reliable, safe, functional, and practical. This section will review a few different sections of our hardware in R.O.A.M. that are all integral to the overall function of this design. The hardware, while some may think act independently of each other, all have to be accounted for as they interconnect in many ways. This design had to be thought out carefully to make sure there were enough resources, and it happened efficiently. There are four different types of schematics below, the first being our MCU design where we integrated every system that will come through our ESP32 which will be the central control for this design. This will be the most significant portion of our PCB design as it involves being able to power all of our components and being able to send data to the right means. The ESP32 will be powered through a 3.3V regulator that will provide enough power to not only power itself, but also function as an intermediate processor for most of our devices. It is the heart of our project as it will handle functions such as sending information and data to our motor driver, sending and receiving information from the Pi, communicating with the SIM module, working with the LCD screen, LED lights, buzzer, and the UART to USB converter. The ESP32 has a generous amount of functions so a lot of time was spent on figuring out the best way to connect all portions of this device. Tedious consideration went into each pinout decision to be able to give the maximum usage to each piece of equipment as we have many components. We have shown a closer visual of the UART to USB device which can be used to see how it was designed for our project.

The next item that was designed was our motor controller. We decided on an H-bridge motor driver that was modeled after the BTS7960 motor driver. This driver will be talked about in further detail later, but its functionality is especially important since we will need it to grab information given by the ESP32. The LiDAR will get information from the world around it, process and calculate what it needs to do on the Pi, then send that information to the ESP32 so that it can tell the inputs on the motor drivers what to do with the motors. This process will be how our robot knows how to move. We spent time deliberating on the functionality of the motor and decided this design would allow for a forwards and backwards motion through the BTS7960 chip. This design will have 8 inputs to the ESP32 which will all control the different inputs on the BTS7960 chip to tell it which gates to open and close to be able to send power through the H-Bridge system and make the robot move forwards or backwards. This chip has a numerous amount of important features such as overtemperature shutdown, overvoltage lockdown, undervoltage shutdown, EMI optimization, and more. Since our motors and motor drivers

are going to be expected to run off of 12V, we found that our battery can directly power the motors without a regulation device. This system was the most optimal in our time of searching through multiple different options that was very optimal as well as right for our motors.

The last group of schematics listed below consist of our multiple regulator types. As previously talked about, much consideration went into the power calculations of these regulators, as well as the statistics of the type of regulator chip chosen. There are three designs to encapsulate the three distinct power requirements that our project has between the different components that will be powered. The first is our 3.3V ESP32. As talked about before, the ESP32 is responsible for many peripherals, this regulator will not only be responsible for the functionality of the ESP32, but also the peripherals and dev boards the ESP32 uses. We chose the LM2576 regulator IC for this design for many reasons that will be mentioned later, but this IC allows us to take 6V input down to 3.3V for this device. A similar situation is happening with the 3.8V regulator needed for the SIM module as well as the LiDAR, as they will be using the same circuit. This regulator will have a 3.8V and a max current of 3A, which will be good for this to not over power the components that it is feeding the power to. The regulator that is for the Pi will use the same IC, but its input voltage will be the 12V input, as the Pi requires closer to 5.2V input. This means that 6V will not be enough voltage, as stepping it from 6V to 5.2V is not enough headroom for it to be efficient. While 12V to 5.2V does possibly require more effort, it is worth it to be able to keep the Pi at a high enough voltage to support its functions. All of these schematics will work together to create a functional PCB that we will design and purchase to develop our project.

6.2.1 MCU

The biggest part of this design is how our MCU will connect everything in our project together. We will be using the ESP32 to control all of our devices that will create the functionality of our project. As the central microcontroller unit, the ESP32 acts as the brain of this system, making it responsible for processing data, sending commands, and creating a communication path between all hardware systems. The main peripheral modules will include LCD screen, buzzer, H-bridge motor driver, Raspberry Pi, and SIM7600 module. This ESP32 was chosen due to its Wi-Fi capabilities, its multiple communication interfaces, multiple GPIO pins, and PWM outputs. The ESP32 has features that are capable of operating up to 240MHz and delivering around 600 CoreMark DMIPS. This provides more than enough processing that is needed for tasks such as polling sensors, displaying updates to the LCD screen, and communicating to our website. It also includes up to 34 programmable GPIO pins which have multiple functions. This ESP32 is where we will control a majority of our components. Starting with the LCD screen, we have chosen the ILI9341which communicates over the SPI protocol. This supports SPI buses, UART ports, I2C, ADC, PWM and more channels. The versatility of this device is important for this project, as we have a multitude of these needs for much of our equipment. Requires connections to a VCC pin, a GND, a SDA, a SCL, and three other GPIO pins. We plan to configure the SPI bus at speeds of 20MHz so that we can have proper screen rendering. These signals are supported on the ESP32 and allow us to incorporate high-speed data transfer. The buzzer is simple as it only requires a

PWM pin from the ESP32 and GND. This buzzer can be driven using PWM signals to vary the buzzing patterns. We can also vary the frequency that is sent to the buzzer through the ESP32 which will allow different tones and patterns being sent to this. The other devices we have included to communicate with the ESP32 all require their own power source. Because of that, we have designed regulators for each of these components, which will be discussed later in a section in chapter 6. Each peripheral module such as the SIM7600 will be powered with its own dedicated regulator. These regulators will have capacitors included for noise isolation. Devices such as the Pi will have its own regulator but it will also be responsible for powering less power hungry devices such as the Pi camera. A similar situation will be with the ESP32 where it will have its own power regulator but it will also have the responsibility of powering the buzzer, LEDs and LCD screen. Power separation is critical in our design to prevent lulls in power or electrical noise that could interfere with our other important data communication. This is especially prevalent in the SIM7600 module. The SIM7600 is our SIM dev board which will be responsible for coordinate tracking as well as sending data to our website through an antenna. The SIm7600 modules is controlled over UART which is why we had the need for the UART to USB communication chip. The coordinate tracking is done through a GPS module that is on the SIM module. We will connect this module to the ESP32 through the RX and TX pins, which are used for serial communication for UART. This UART connection enables asynchronous serial communication, which makes it ideal for handling GPS coordination and other responses needed for the module. In this design, one of the ESP32 RX and TX pins will be reserved for the SIM7600, as this pin is necessary for the programming. This will be the way we transmit data so that the ESP32 can process the information for the website.

Another piece of equipment that is connected through the ESP32 is our motor drivers. We will expand upon the motor drivers in section 6.2.3, but the motor drivers will be responsible for sending the correct data from the information we receive from our LiDAR camera to the motors. To do this, we need the intermediate step of the ESP32 to calculate how to move based on the information it receives from the LiDAR camera. This logic layer is necessary for the autonomous navigation, as it will interpret raw data and convert that into real time movements. Once the ESP32 knows how to move, it will send a signal to our motor driver pins, which are IN1-IN4. The ESP32 will send the PWM signals on each IN connection, these frequencies are configurable. This is how we will send the amount of speed that we want to send to the motors. This is also where we will send the direction control logic to these devices since the motor driver is bi-directional. Each of these are connected to a GPIO pin on the ESP32, capable of sending a PWM signal to control the motor speed. The direction the motors move will be based on which IN pin will receive the data. This will be expanded upon in the later motor driver section. To allow for communication with the ESP32 and to allow for us to be able to configure the ESP32 chip, we need a USB to UART system that will provide this communication. We will need to be able to send firmware to our ESP32, meaning that we need this communication to send commands. This data will be transmitted by the RX and TX pins that are connected to the ESP32. We have it connected to the same port as what the Raspberry Pi will be using. This means, upon use, we will have to select the right destination for our information. Since these devices will not need to be used simultaneously, we do not have to worry about conflicts with that. The last section of

hardware that is connected to this is our Raspberry Pi 5. The Pi 5 is responsible for interpreting the images for our trash and treasure categorization. This needs a RX and TX connection so that it can send information to the ESP32 and to the website. On the Pi, we will have more connections, such as the Pi camera to the Pi 5. This camera will be responsible for taking the pictures that will be used to identify what the object is. The image data is then passed to a classification model which will determine its categorization and send this information to the ESP32. To connect this device we will be using the CSI camera port on the Pi5. The LiDAR will also be working off of the Pi 5 since it needs certain autonomous thinking that the ESP32 does not offer. This will use a few pins such as VCC, TX, a motor control pin, and GND. While these peripherals interface with the Raspberry Pi, they are functionally linked to the ESP32 making their shared logic necessary to incorporate into the discussion of the overall system architecture.

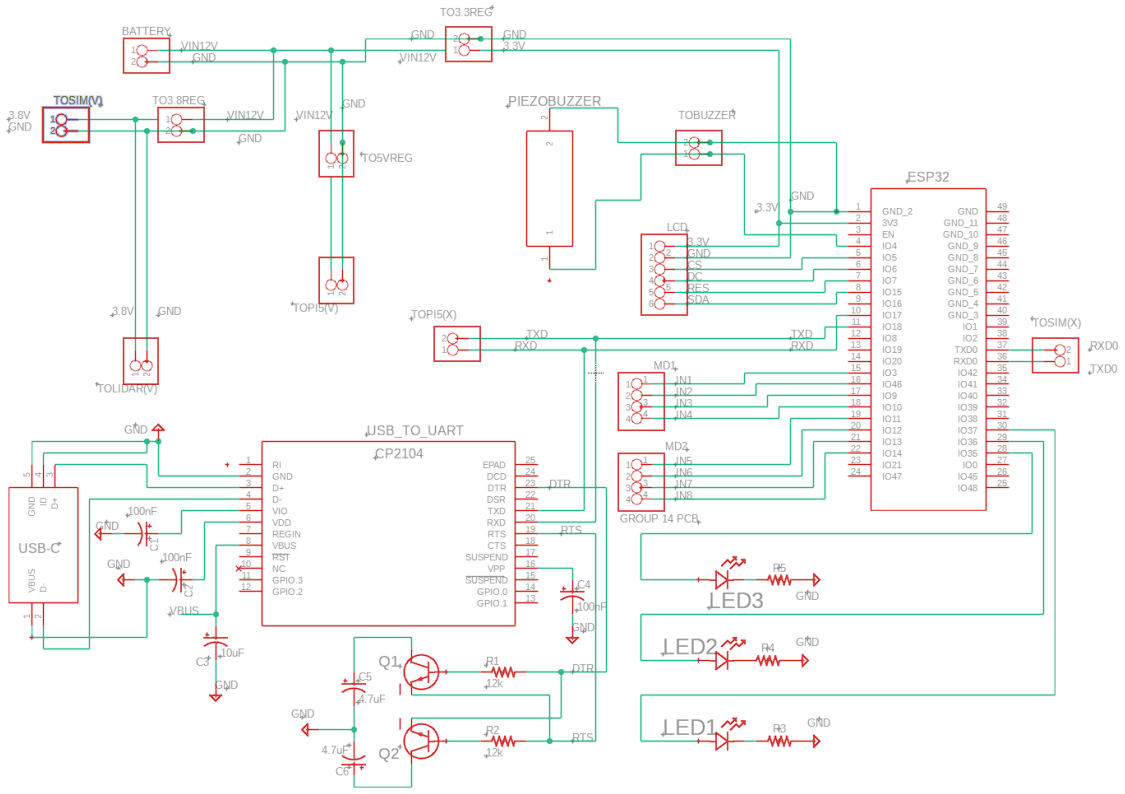


Figure 6.1 — MCU Schematic

6.2.2 USB to UART

In figure 6.2 there is a closer visual of our USB to UART system. UART is a universal asynchronous receiver transmitter, it is a communication protocol that allows asynchronous serial communication between different devices. This device connects directly to the ESP32 through the TX and RX receiver. This device has a USB connection

on the other side which is another protocol that is used often to connect these devices. These two protocols cannot directly interface which is why it is necessary to use a converter to be able to communicate information properly. As stated above this system is useful to communicate data and software as we develop the system we need to use. This system is our host and the ESP32 can be integrated for programming. In the process of this project there is going to be a significant amount of information that we will need to program and flash to the ESP32, this system allows us to do that. The CP2104 translates the signal given by the USB to a language that the ESP32 can understand and work with. The controller makes the interfacing between these devices simpler. Without this we have considered using an external programer or an SPI as we had considered sending photos to our website. With this configuration, sending photos is not available with the UART communication, which is why we considered using SPI interfacing. UART is a point to point protocol with limited bandwidth which means that image transferring protocols are too robust for this type of transfer. At this time we are focusing on primarily using data and no photos meaning that the UART to USB configuration is all that we do need. As seen from the MCU configuration in figure 6.1, this will be powered through the 3.3V regulator.

We are using the CP2104 to configure this. It acts as the bridge between these two devices converting a USB signal into a UART logic which the ESP32 will use. This chip supports baud rates up to 1Mbps which will allow us to have varying speeds for communication. These capacitors surrounding this device stabilize this power at 3.3V from the USB, certain ones like C3 support stability of the voltage. The USB power can be noisy which is why this is important. They help reduce ripple that will be seen by the USB, making their presence necessary. The signal from the chip connects the TX and RX pins on the USB to UART device and the ESP32. You can also see connected to the DTR and RTS pins, a system for controlling the rest and boot pins. This uses two inverters to drive the signals coming from this device to the ESP32, specifically to flip the logic to match the ESP32's reset logic. The ESP32 requires timing on the reset and boost pins to be able to enter the flash mode. This will allow the user to not have to manually hold down buttons to flash the firmware. Overall this system was necessary to integrate and configure these devices, without it there may be data able to be stored in the flash, but not much development would happen. We also need this for debugging as well as firmware updates that may need to happen along the way as we develop this project over the next few months. Having a separate section for this also gives a closer look into this system with the multiple systems going on in the MCU schematic.

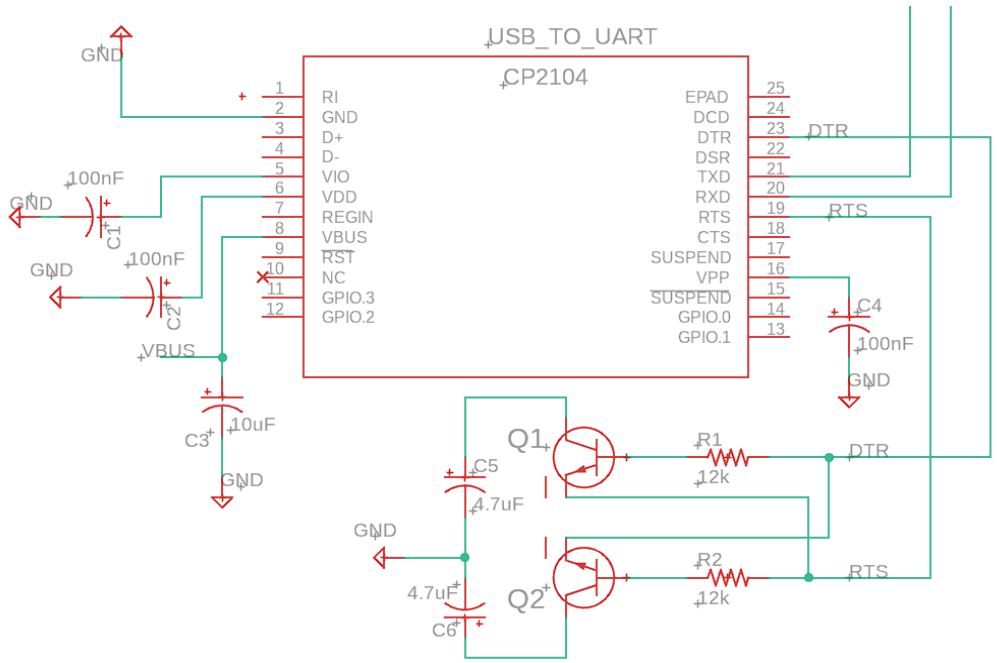


Figure 6.2 — USB to UART

6.2.3 Motor Driver

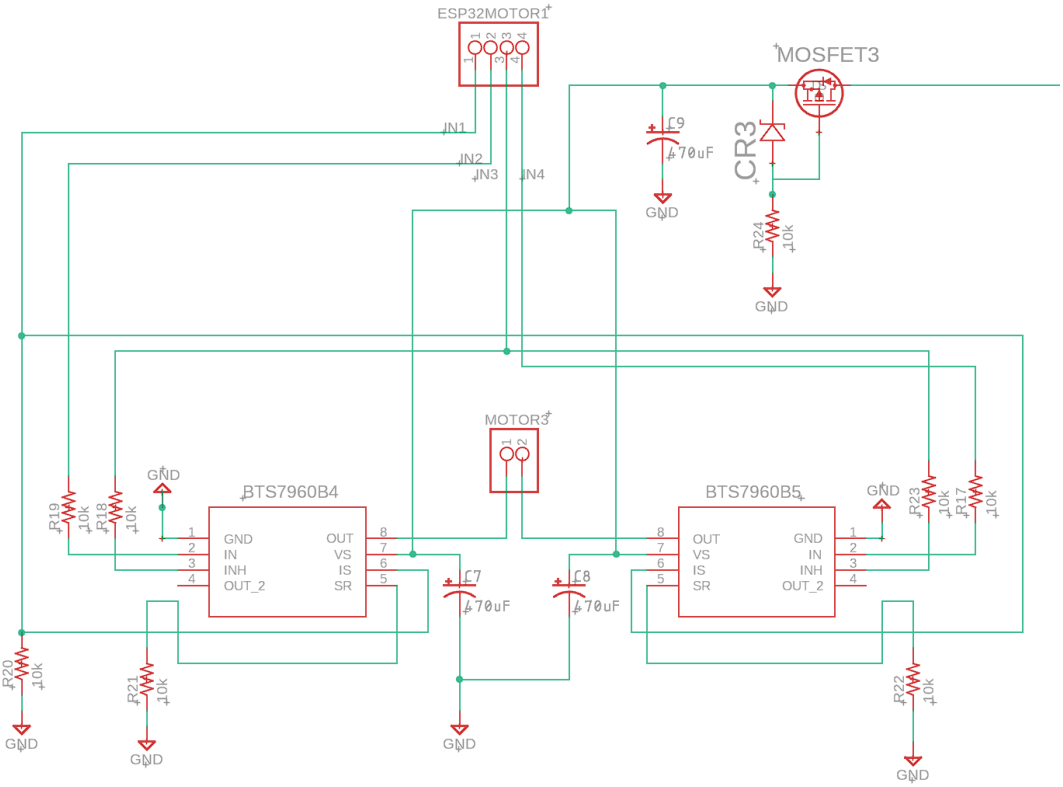


Figure 6.3 — Motor Controller (Left)

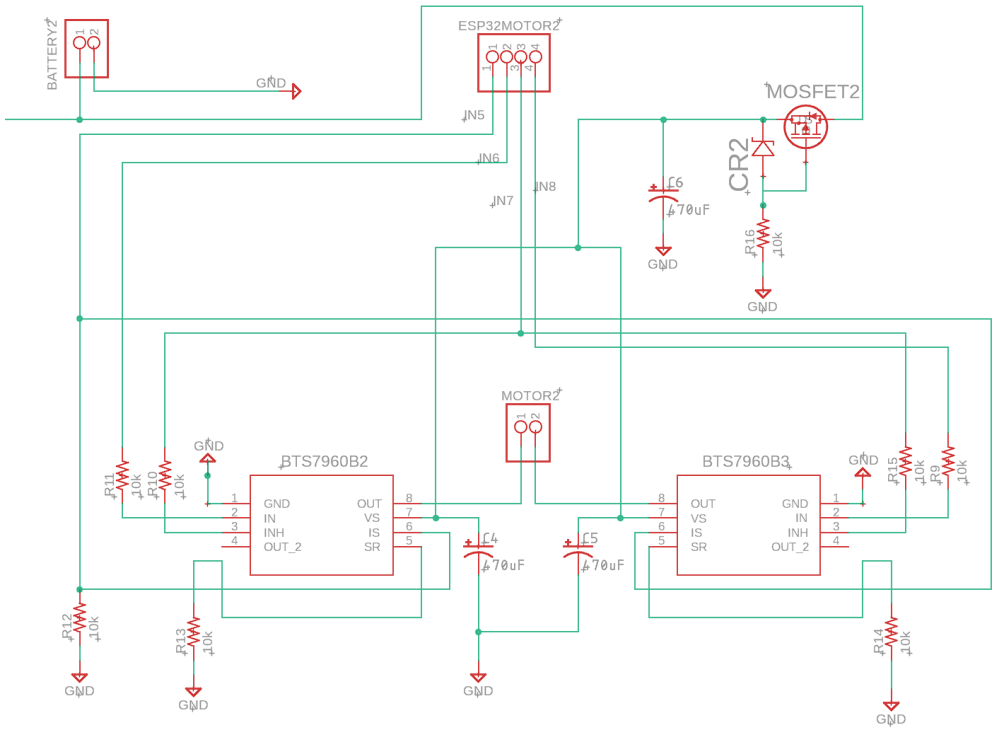


Figure 6.4 — Motor Controller (Right)

In designing our motor driver, there were many things that went into this decision, such as the power supply needs, our motor's stall current, the PWM control, and more. This circuit we want to design needs to match up with the motors we chose. Since we chose the goBILDA 5202 Series motors, we need to design this motor with those specifications in mind. To do this, there were a few specifics that we needed to satisfy for this circuit to work, the first being the voltage input. Our motor takes 12V, their no load current is .25A, RPM of 312, and their stall current is 9.2A. We needed to make sure that our specifications met these requirements. With the BTS7960 we are in an acceptable range for the stall current, which has a range of up to 43 Amps. This circuit also has an acceptable range of input voltage to this system which is 12V and the motor driver can accept up to 27V. These specs were the most important to review.

The figures above, 6.3 and 6.4 show a closer up visual of our motor driver system which helps understand this system. The left showing our motor, This system models a H-bridge circuit that independently controls two brushed DC motors. They are both connected to the motor pins which are not explicitly shown above as the photos' ware split. Each H-bridge is capable of controlling both the speed and direction of its respective motor, this is essential for being able to move as we need in our differential drive system. This circuit has these mirrored to each other for easy understanding, but individually, one H bridge is shaped to allow for a convenient flow of power to be able to swap directions of the power to allow for bidirectional movement. The BTS7960 is a high-current driver

with internal logic that allows a 3.3V control signal that will be sent by the ESP32 without using a shifter. The drivers themselves have semi built in temperature shutdown and more to protect our circuit and components. When sending a high signal to the transistor, it turns it on and pulls the gate low which will activate the P-Channel MOSFET. To turn this on the gate must be low which is why we use this transistor which is controlled by the logic signal coming in. This setup also allows us to combine 3.3V logic from our ESP32 with a higher voltage demand given by the motor driver. This also ensures proper voltage levels which shifts without compromising our whole system. The N-Channel MOSFETs are connected between the motor and GND, this turns on when the gate is high. We are able to control which directions the motors move by moving the voltage through this system in almost an S shape. This S shaped flow is a traditional characteristic of the H-bridge since it is important for not both P-Channels or N-Channels to be on at the same time. For example, to move it forward we turn the left gate low and the right gate high. This means that it starts at the left P-Channel MOSFET, then goes to the motor and then goes to the right N-Channel MOSFET, then to ground. To move them backwards would mean to do the same system but starting from the right side. This though means that we can't send both inputs to be high or else there would be an issue. If this did happen it would short our VCC directly to ground causing excessive current flow or overall damage to the system. To send the right information to the motor we need the inputs to accept not only logic but also PWM signals. This PWM signal allows us to control the motor’s speed by adjusting the voltage that is sent which will affect torque and therefore speed. To ensure the operation of this would be correct, we looked into how the PWM would be sent from the GPIO pins to confirm it would be enough. Making sure the duty cycle coming into this circuit would support the system was important as well. Then on the ESP32 we have the data going to a certain number of pins. These being the IN1-IN8 GPIO pins on the ESP32 pins that are shown in figure 6.1. These pins support PWM signals which were important to find. This system will have an input voltage of 12V which will power our motors directly. This system also is seen to have a connection to our battery through direct pins.

In this circuit you should also be able to see our capacitors that are placed as buffers for the voltage dips near the VS pins. These capacitors are important to stabilize the voltage in the circuit. It is necessary as any noise or stalls could affect the motors or the circuit, which is why we have included ways to prevent those issues. We also included diodes and MOSFETS as talked about above to prevent voltage spikes from the back EMF that may occur when there are sudden movements such as reverse direction or motors suddenly stopping. Overall, the H-bridge system implementation was chosen because of its ability to handle these different current loads since this system can be modified and customized. We also are able to have bidirectional movement which is also necessary in our project design.

6.2.4 Power Supply

6.2.4.1 Raspberry Pi 5 Regulator

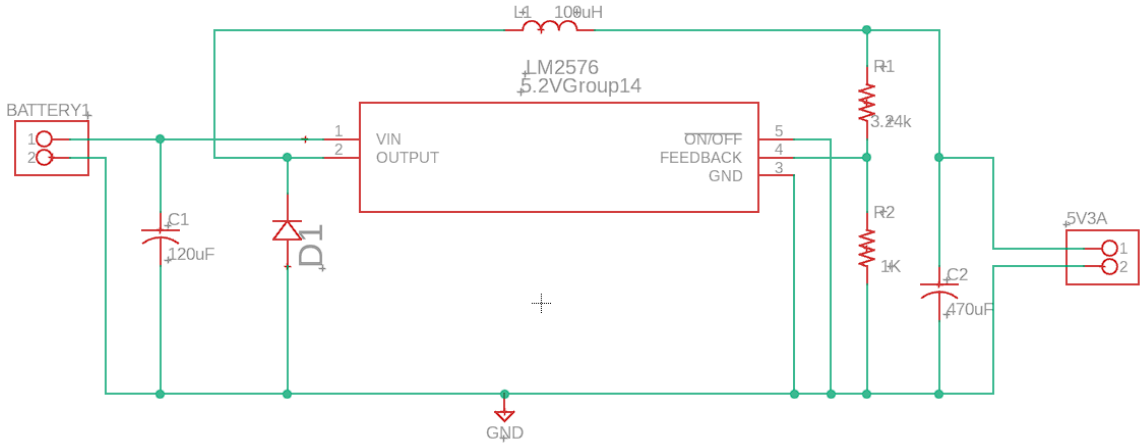


Figure 6.5 — Raspberry Pi 5 Regulator (5.2V and 3A)

Power regulation is a necessary and important function of this design. There are multiple components that have different power requirements leading to a few different converters that are created to be able to design this project. These regulators will not only be used to take our 12V battery and be able to get a few different voltages off of it stably, but also to protect our equipment from a power surge or any other issues that could damage components. This design not only steps down the voltage, but it also will help minimize ripple and maintain headroom. These are necessary components of the regulator. Figure 6.5 shows the first converter was created with powering our Raspberry Pi 5 in mind. The Raspberry Pi 5 has a voltage input range of up to 5.2V with a typically amperage max of 3A. This amp is typical if there are peripherals on the Pi which we do have a significant amount. In our case the Pi is integrated with a camera, meaning that we want to plan for the largest amount of current the Pi can handle which in this case is the 3A. The converter supplies up to 3A continuously and is capable of delivering up to 15.6W of power. This makes this design suitable for this load especially with its peripherals. Using the LM2576-ADJ converter, we created a compact converter that will take our voltage and amps down to the allotted amount for this system. This design is similar to the other regulators found below in this project with slight number deviations to allow for this circuit's specifications. Searching for a regulator to fit into this design led me down various paths that had me needing to look not only into a design that would work for the power needs, but also work for the physical build needs. The design we had come to develop earlier in this project worked electrically, it was able to output the proper value, but when developing in for the PCB we quickly realized it had too small of a regulator chip. The regulator chip would be incredibly difficult to physically work with, this means it would be more prone to shorting, and any mistakes that were made would be even more difficult to solve. This is why the LM2576 chip was chosen for this design. It is not only very user friendly in its size, but its variability allowed us to have an adjustable circuit to fit our design specifications exactly.

To find the values needed for this specific circuit, we referred to the datasheet for the LM2576 device. This converter specifically allows for up to 40V since we are not using the high voltage version. We are however using the ADJ version which allows for an adjustable range. Since the Pi has a higher voltage allotted than the 5V of a typical 5V LM2576 converter, we needed to include additional feedback resistors that would help us drop the voltage to the specific range we require. To complete that, referring to the datasheet you can see that the adjustable output voltage ranges in increments of 1.23V. To find our resistance value we can use a voltage divider to find the resistor values:

$$V_{out} = 1.23V\,(1 + \frac{R_{top}}{R_{Bottom}})$$

In this case we know we want an output of $V_{Out} = 5.2V$ so we can include that as well as 1kΩ as our bottom resistor. This will act as a control value so that we can solve for what we require for our top resistor value. Doing this, our equation gives the value below.

$$(\frac{5.2V}{1.23V} - 1) * 1000 = R_{Top} = 3.2k\Omega$$

Knowing this information, we were able to fill out the circuit for our use. This circuit was designed with the specifications for the Pi in mind as has been stated. Additionally, the future regulators will be designed with their circuit's specifications in mind as well.

This circuit has an efficiency of about 85.3% which is necessary for our system since the Pi will be working hard, meaning the least amount of power we lose in heat is less significant. With a switching frequency of approximately 52kHz this converter can achieve peak to peak ripple as low as 195.19mV. That means this circuit's ripple output is less than .1% which is necessary for the Pi as the Pi has a ripple tolerance of up to 3% of its 5V. This circuit was the most efficient and practical circuit we could create to allow a stable power supply to the Raspberry Pi 5.

6.2.4.2 SIM7600 and LiDAR Regulator

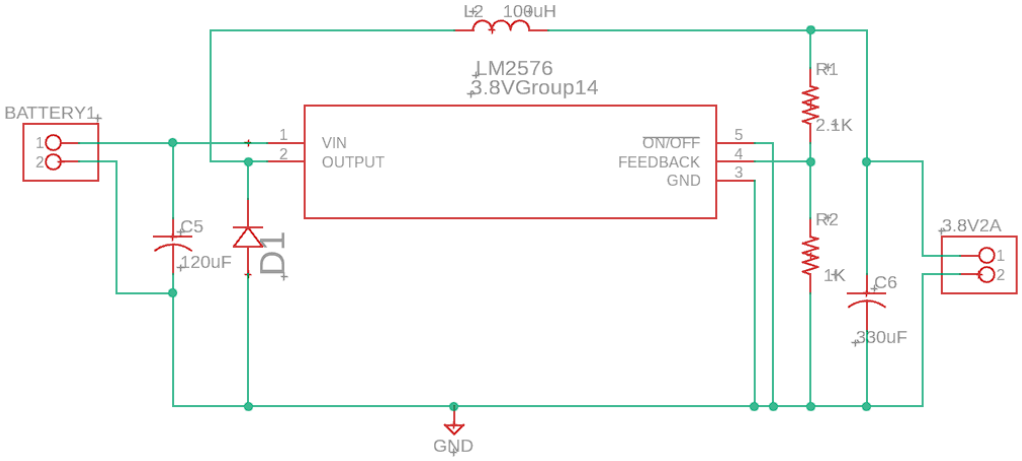


Figure 6.6 — SIM7600 and LiDAR Regulator (3.8V and 2A)

The circuit in figure 6.6 represents the regulator created to supply safe and efficient stepped down power to our SIM7600 module and our LiDAR sensor. Power regulation

for this component is important, as this device is not only costly, but also integral to our design. We cannot afford to have to purchase a second part due to the importance of this. With that, significant considerations into the design of this circuit. The regulator was optimized for accurate voltage control and reliable cooling. Considering the work that this device has to put in, we knew that the spikes for the data transmission could be difficult to manage. The SIM module requires a range of 3.6-4.2V to be successfully powered. In the design of our regulators, we wanted to be as efficient as possible, this means taking into account the range of our components and figuring out which overlap so we can combine them. With this range we divided that if we used the value 3.8V, we could use this design for both the SIM module and the LiDAR sensor. With this module needing a 3.8V input and an approximate max of 500mA of current, we found the LM2576-ADJ would allow us to create that specific value. This design uses an output current of 500mA and up to 1.9W of power which will allow a sufficient amount to meet the communication needs. This system also had a LiDAR camera with a range of up to 3.8V which allows the maximum functionality for this device, while staying in range of the needed voltage for our SIM module. These ranges allowed for us to be able to combine components under one regulator to have a better simplistic design. To meet these requirements this regulator was made with these restrictions in mind. The LM2576-ADJ converter was used in conjunction with feedback resistors in a network to achieve the correct output. Since this component was not one of the standard versions available for use, we had to calculate the resistor values for the feedback resistors to create this output.

To calculate these values we used the datasheet for the LM2576 converter which like previously stated, has a range of 1.23V to 40V allowed. In this case we are attempting to take 6V down to 3.8V which is well within this range. To do this we again used a voltage divider equation to find our resistor values. Using this equation:

$$V_{out} = 1.23V\,(1 + \frac{R_{top}}{R_{Bottom}})$$

We know we need the $V_{out} = 3.8V$ and we can choose $R_{Bottom} = 1k\Omega$ so that solving for R_{Top} is less complicated. In doing that we can fill out th rest of the equation to get an approximate value of:

$$(\frac{3.8V}{1.23V} - 1) * 1000 = R_{Top} = 2.1k\Omega$$

This gives us the circuit that we will use to regulate our 3.8V input into the necessary devices. A lot of the statistics on this regulator are similar to the previous, but whether it is suitable for both devices is important to identify. The selected circuit has an efficiency of 81.3% to allow for minimum heat generation and energy lost in heat. This circuit uses the same switching frequency of 52kHz as it uses the same chip. The circuit minimizes ripple, meaning that it will help avoid interference with other devices, the Vout peak to peak ripple voltage is 105.5mV which is less than .1% meaning it is suitable for these devices and their necessary communication. Taking into consideration the current requirements, as we move onto developing the PCB we will have to use a certain mm of

tracing for it to be suitable for these designs. Overall this circuit was designed to support the powering of the SIM 7600 and LiDAR camera which it surely will achieve as time has been spent to research a suitable chip. Time has also been spent to design a chip that can be purchased in bulk with the intent to be adjustable based on the circuit. The LM2576's adjustable design is very applicable in many different applications which is incredibly important to the design of our project.

6.2.4.3 ESP32 Regulator

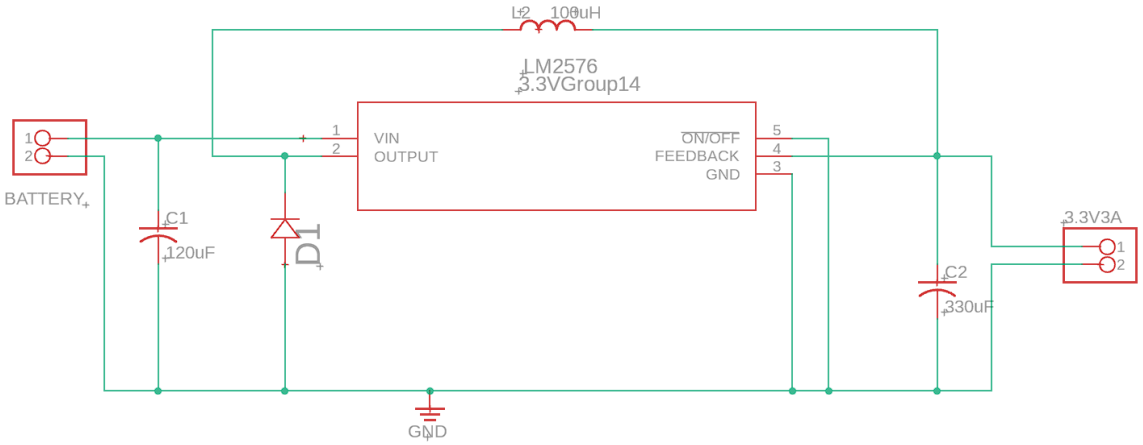


Figure 6.7 — ESP32 Regulator (3.3V and 3A)

This design was created with the ESP32 in mind. While this voltage was not much off of our other 3-4V regulators, it is actually significant that we did not combine them all onto one. First, while the ESP32 is performing many functions, we wanted to make sure it was supplied with the most power that it could to succeed with its necessary specs. This also works inversely for the previous circuit as we did not want to decrease the voltage to the LiDAR just to be able to fit more components past a regulator since it would not function as efficiently. Power regulation in this case is critical so we can protect as much of this process as possible. In this regulator circuit we are using the LM2576 which will give us a similar design to our previous regulators. While it is going to be a similar use, we will not need to use the ADJ version of this regulator chip. Instead, we will need to use the 3P3KTT version. This is because the regulator that is needed for this circuit is actually a standard type for this regulator chip. Unlike the 5.2V and 3.8V regulators, the LM2576 has a design for 3.3V as this is very commonly used for components.

Because of its design, there is no need to do any calculations for a resistor feedback circuit, the standard available circuit that is given for this device is suitable. Additionally, given this converter with its many perks, we are given a list of operating values in its system information that fit our needs. The total power output for this circuit is 9.9W which is perfect for our design as that is what our circuit may achieve under larger loads. The Vout tolerance on this is just under .39%. This circuit was also built with efficiency

in mind. Since we are stepping down from a solid 6V to 3.3V, that significant change could cause energy lost in heat without the proper considerations. This circuit was designed with 77.7% efficiency. This means that this circuit will suffer minimal losses to heat, and we can actually enhance our energy usage. There are many components in this small space, and we are aiming for continuous operation, which means that we need to be able to withstand any heat or avoid it all together. This circuit provides an example of that level of efficiency. There are also a few more benefits to this circuit, one being its switching frequency. This circuit has a switching frequency of 52kHz which allows for a faster response with these components. Many of these are similar to the previous circuits with their chips. We also can see this circuit has voltage ripples of less than .1% which means that there will be low noise and low interference in our communication. This is important with sending data or viewing the data as the ESP32 will be the heart of our system. Overall, this circuit is pretty useful to us in regulating our system's power so that we can run the ESP32 without the issues that come with a less efficient circuit.

It is also something to note that this circuit was compared with the ADJ version. Since the 3P3KTT was made with the intent to output a preset value, it came without the additional resistors as the previous regulators. This means that the design is only slightly different, this means that things like the wattage, switching frequency, and even efficiency were the same. With that slight difference though, there was a difference in Vout tolerance. Looking into that difference further, we know that the LM2576 has a reference voltage of 1.23V plus or minus 2%, this means the range of that actual voltage could be 1.205V to 1.255V which for this lower voltage can mean a lot. The ADJ converter had a tolerance of 2.34% which is higher than that 2% threshold that the datasheet mentioned. The 3P3 version had a .39% tolerance. The other factor in this decision was the actual structure and purpose of these components. The downside to the 3.3V converter was really that it can only be used for 3.3V. With the other converters we could possibly repurpose them by adjusting just one resistor value. That factor added to the design, but overall since 3.3V is a widely used value, we chose to go with the design made for 3.3V. This difference of the Vout was significant enough to make the decision between the two options easier. On one hand we would have variability but suffer with a less accurate Vout, and on the other had a more accurate Vout for exactly the cause we needed. This made the decision easier in the end which is why the LM257-3P3KTT was the decision we made.

6.3 Conclusion

In conclusion, the design of R.O.A.M. reflects a carefully balanced system that involves logic, power regulation and carefully chosen components to create that system. This includes not just the ESP32 peripherals but the regulators that are needed to be curated to the components they are meant for. We needed to design regulators to meet the high power needs from equipment such as the Raspberry Pi and communication intensive modules such as the SIM7600. From the integration of the ESP32 as the center of this system, to our specialized regulators to distribute the power, each system really builds off of each other. For example, the UART to USB subsystem had to be carefully routed to the RX and TX logic ports on the ESP32 for proper communication. Similarly, the LiDAR

while it is pulling power from a regulator, it is connected logically to the Pi and it additionally affects the ESP32 motor control decision. These systems were not designed in isolation, all areas such as the power delivery network directly influence the logic of the PCB schematic. A key decision that was made in this process was to have our regulators separate from each other and from the ESP32 PCB. The goal with this is to be able to replace just one regulator that may have had issues or if one got damages. This will prevent us from having to purchase an entire additional PCB board if something happens to it. Instead we can just de-solder and re-solder a new regulator onto the PCB. Each of these systems brought their own complexities and maximization efforts. Managing these tradeoffs between the availability, sizes, efficiency, performance, and more all had to be carefully balanced. Tradeoffs include finding adjustable voltage regulators and weighing the tolerances as well as efficiencies to be able to find the perfect device. We even had to spend time making even smaller decisions that were also important, such as deciding peripherals and devboards to group onto certain regulators. To understand how everything was to come together, we had to understand all the power requirements. This also meant that we would be able to protect against other, unforeseen situations that could destroy our technology. To protect against issues that could be caused by inrush surges or reverse current, we had to take care to add buffering capacitors and more into our motor drivers. Future testing and validation will be necessary to make sure these designs will be accurate for each tolerance window of each component. We want to make sure that they will operate under peak load conditions and much more. Collectively, these systems work together to create this design, this efficient system will advance our design and create a solid foundation for our project to build upon. The logic, power, and communication systems will form a well integrated robot that will allow for the safe and functional operation of R.O.A.M.

Chapter 7 Software Design

7.1 Software Description

Figure 1.3 is the outline of how the software of R.O.A.M. behaves overall and how the subsystems interact with each other. The software serves three major purposes: autonomous driving, object classification, and data transmission. The autonomous driving works by using LiDAR to scan the local environment in front of R.O.A.M. and create a 3-D point cloud of the surroundings. This point cloud will be used to identify the sidewalk and thus provide directional adjustments to the motors. R.O.A.M. will only have two motors, directional adjustments will be provided in the form of the voltage supplied to the left and right motor allowing for turning as well as smaller path corrections. Data transmission will only happen in two scenarios, either a treasure is found or the trash amount needs to be updated. If treasure is found then a package is sent to the ESP32. The ESP32 will take the package and attach a GPS location to it which will be sent to the website making the information and location viewable. If trash is found an internal counter will increment by one, this will be set on a timer that would periodically update the website with how much trash was found.

7.1.1 Object Classification Flowchart

Object Classification will provide the core functionality to our project. This will detect and identify objects on the side of the sidewalk as trash or treasure using a machine learning model. The model will be trained on a dataset of images originating from both online and taken in person, this helps narrow down difficult edge cases that R.O.A.M. will specifically encounter. This process will require a camera mounted near the top of R.O.A.M. If the camera stops working R.O.A.M. will be programmed to stop moving to avoid missing any objects. The binary object classification will be done by the YOLOv8n model. This model is equipped to detect and classify objects. YOLOv8n requires preprocessing of images, including changes to the color channel orderings and resolution.YOLOv8n also requires post-processing. YOLOv8n will feed into a SORT object tracking algorithm. This will ensure that objects identified will not be repeatedly identified, leading to less reliance on a stable internet connection, which is a limiting factor. Once the image is identified, it is categorized and handled according to its classification. Then packaged to be sent to the server to be viewed on the website. Figure 7.1 shows how the data structure will look when sending data from the Raspberry pi 5 to the ESP32.

Detection Report
image: JPEG
Object_type: str
conf: float
time_stamp: str

Figure 7.1 — Pi to ESP32 Data Structure

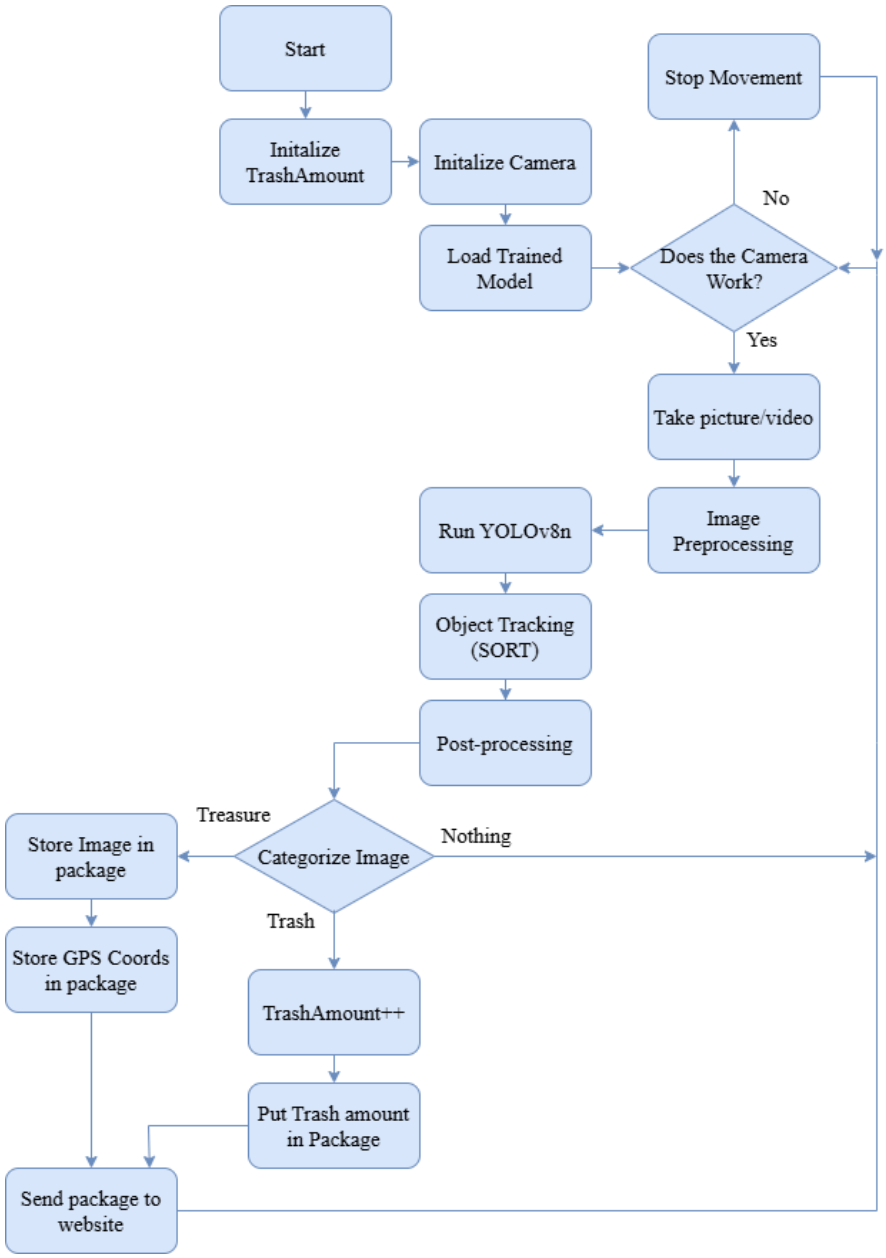


Figure 7.2 — Object Classification Flowchart

7.1.2 ESP32 Software Flowchart

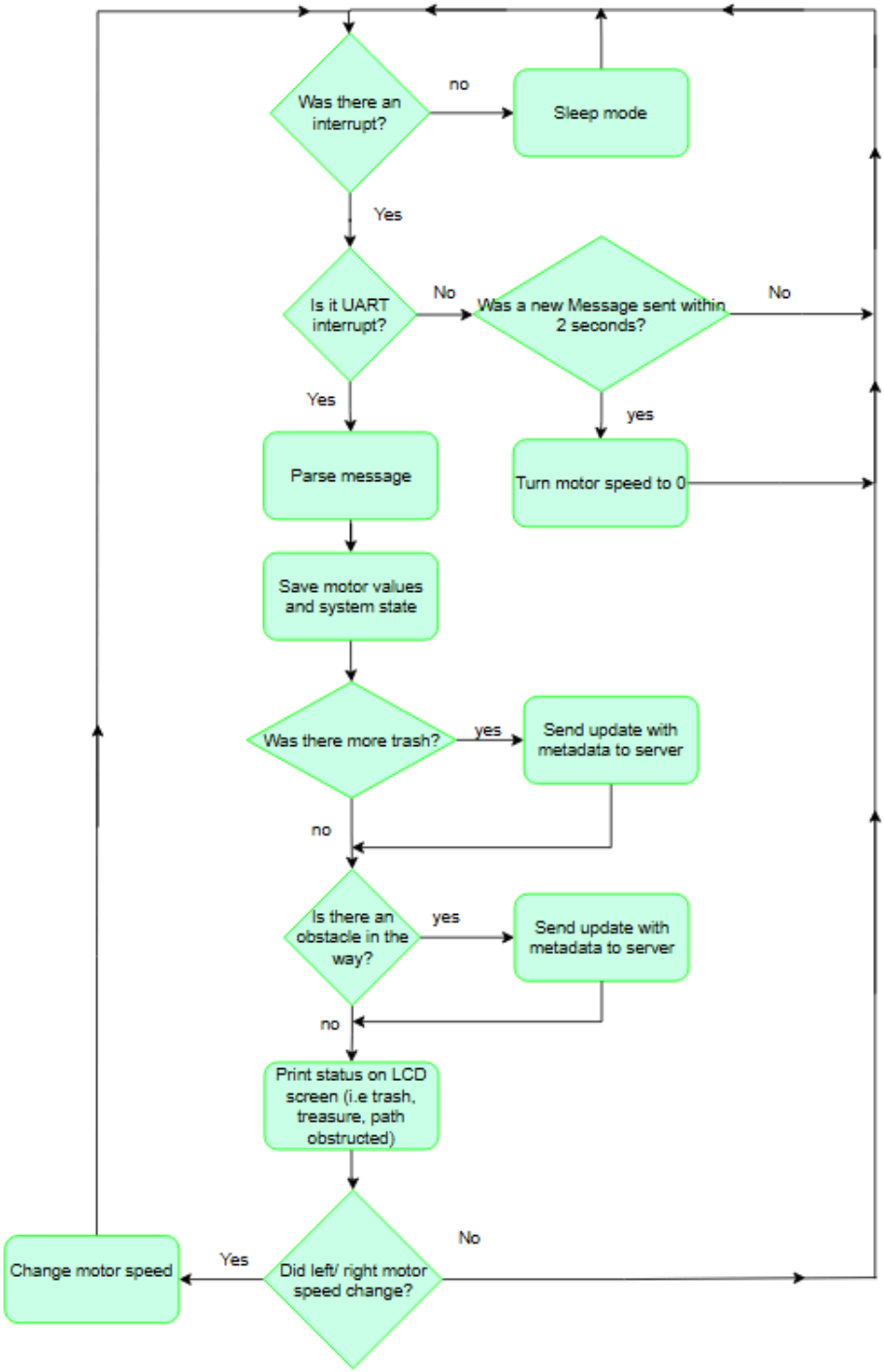


Figure 7.3 — ESP32 Software Flowchart

The two main jobs of the ESP32 module are to control the motors and send meta-data to the server, over cellular data. The meta-data consists of. The flowchart starts by checking if an interrupt occurred. If there was no interrupt, the processor goes to sleep mode to conserve power. If there was an interrupt, the program checks to see if it was a UART interrupt. UART will have the highest priority to ensure its data is handled first.. This occurs when the edge device sends data to the ESP32. If there was a UART interrupt, the message is then parsed, and the motor values are updated. The program then checks if more trash was detected. If more trash is detected it will send the location and the amount of trash or treasure that R.O.A.M. found to the server. The program then checks if the motor speed changed, if so, it updates accordingly, if not the ESP32 goes back to sleep. If the interrupt was unrelated to UART, the system checks if a message has been sent within 2 seconds. If no message has been sent, the program changes the speed of the wheels to 0. This protects R.O.A.M. if a communication failure occurs between the ESP32 and the edge device. The esp32 also updates what is happening in the LCD screen. It will display if trash or treasure is found, and also if there is something in the way.

7.2 Interactive Website

The website will be the main source of information gathered by R.O.A.M. for The website will list available treasures in an area with a picture and coordinates of the treasure, so the user can assess the object and deem it worthy of pick-up. As it traverses through a neighborhood, it will identify these objects and report to the website. This feature should focus on ease of use and accessibility. In the figure below we have an example of the website that we will create, this was made in javascript and openstreetmap where we were able to model an idea of what we would like this to look like. This example is without a dedicated database. Trash symbols show the aggregated trash numbers in the area, and the treasure bag is the individual furniture. The meta-data and more details will be included at a later date. In this section, we will describe the general user interface that will be incorporated into the website. We want to describe how the users will be able to interact with the different options available for use.

7.2.1 Web Layout

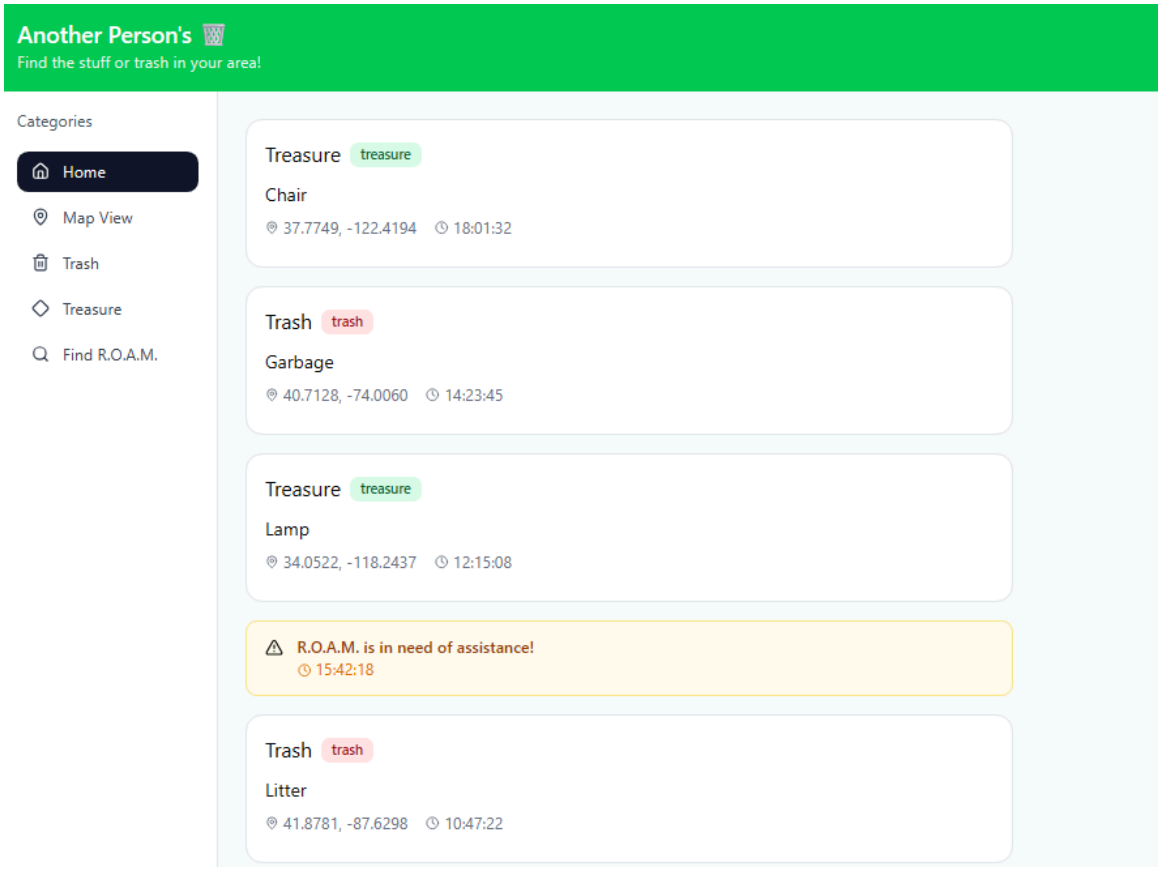


Figure 7.4 — Website Home

The figure 7.4 shows our homepage to this website, it shows a few tiles on the left to choose from, these are the most popular options to see recent activity from R.O.A.M. In this page you are able to view coordinates, as well as time stamps of each entry of roam. On the left there are also more options to navigate to on the website. Additionally, this serves a functional purpose. Because R.O.A.M. may from time to time need assistance if it gets stuck, trapped, or tipped, there is a plan for it to notify the website so people can go out of their way to help it. This notification can be seen pop up on the home page. So if a user happens to be scrolling for trash or treasure they may happen upon this distress message. They can then go to the Find R.O.A.M. tab, get its coordinates, and assist it.

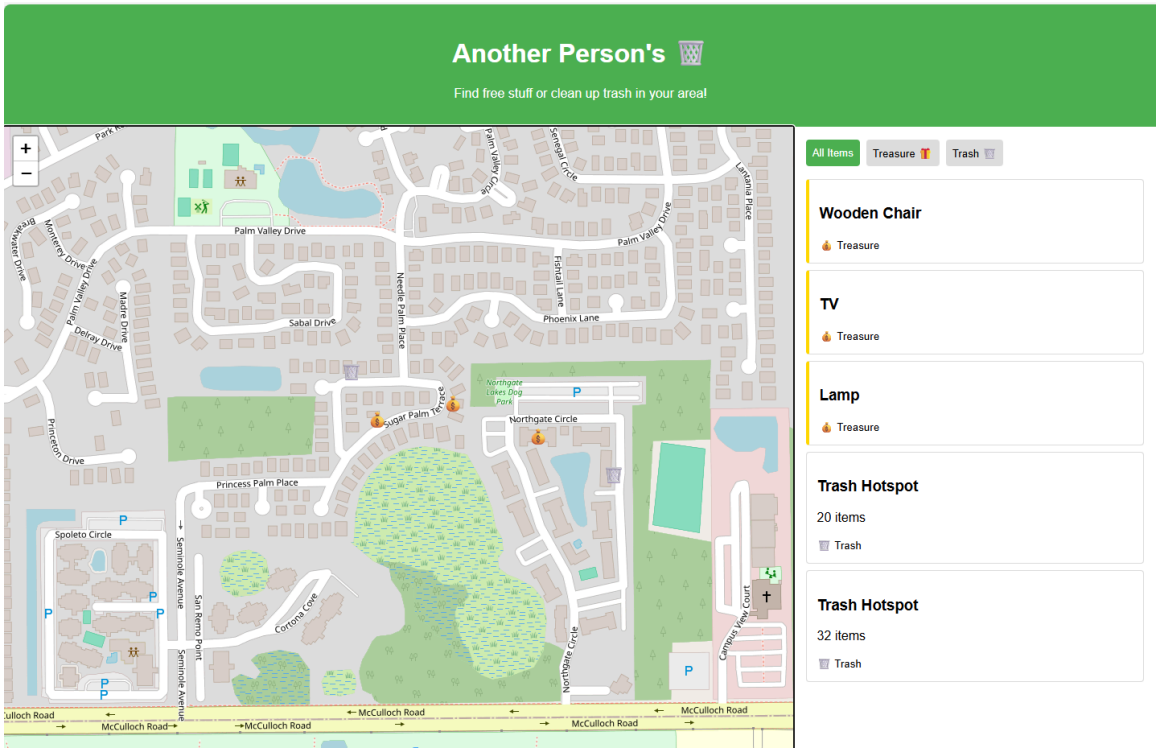


Figure 7.5 — Website Map View

Figure 7.5 shows the Map View on the website, this page shows the recent activity and their locations. On the top of the information, you can see options to sort this data between the different types of options. This way of interfacing between options is helpful to allow the people who interact with this website an enjoyable and functional time.

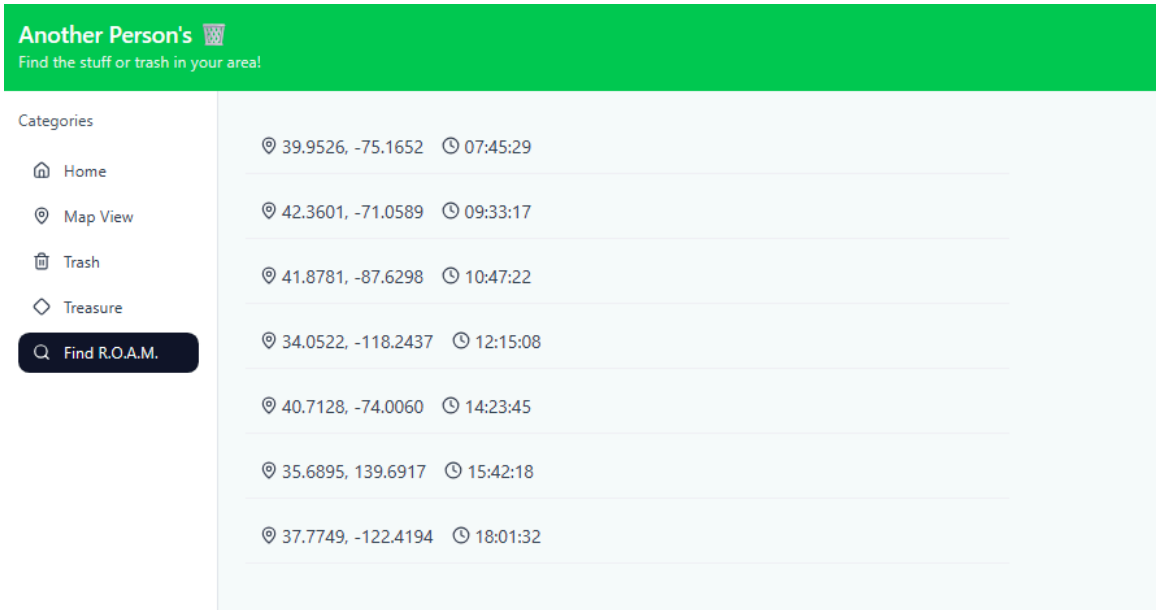
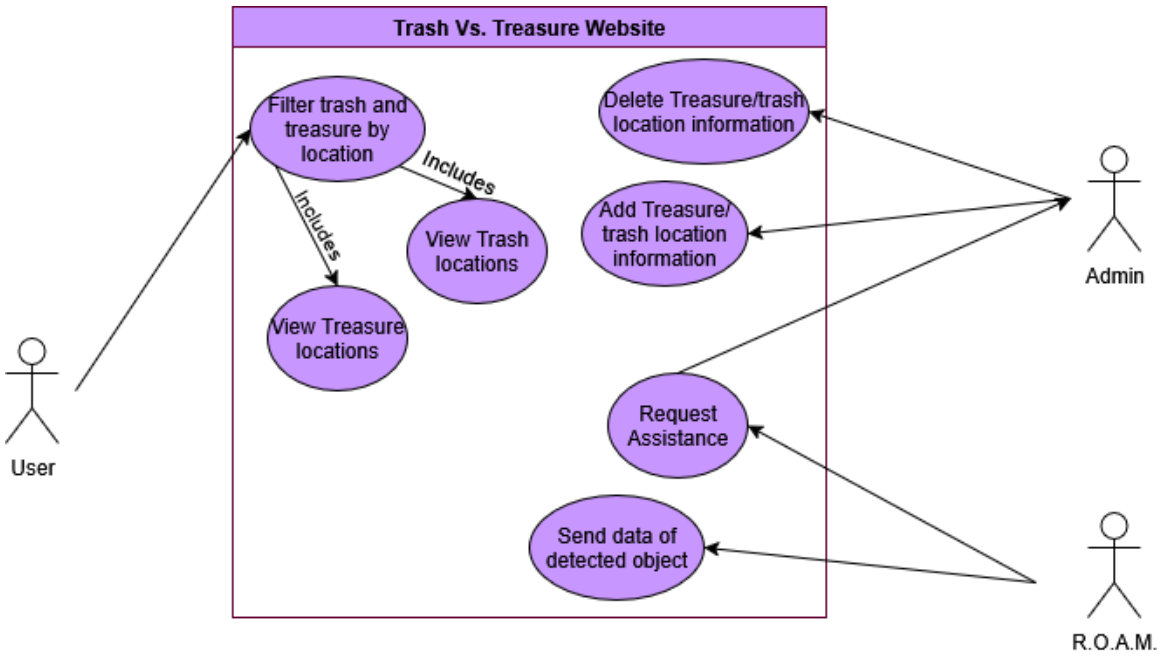


Figure 7.6 — Website Find R.O.A.M.

This last figure shows the last option on the sidebar which is the find R.O.A.M. option. Inside this page you will be able to see the location of this robot and for what time that was active. This will make finding R.O.A.M. significantly easier and more user-friendly.

7.2.2 User Interaction Diagrams



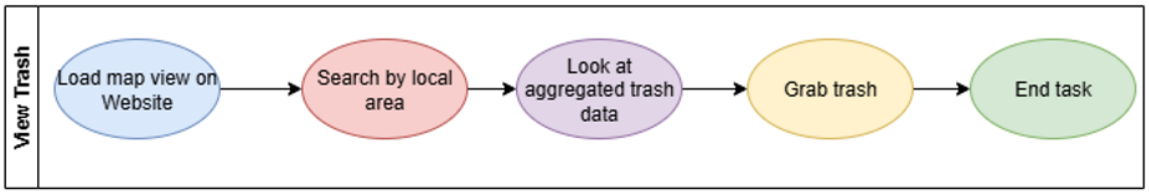


Figure 7.7 — Case Flow 1

Figure 7.7 describes one possible process model for this website that a user can experience, titled "View Trash". If a user wants to view trash that is in a particular area they can directly access aggregate trash information, search via their locale, organize the data to what they want to view specifically, grab the trash from that location, and then go on about their day. The organization of this chart allows users to see the simple way that the website can be used and navigated.

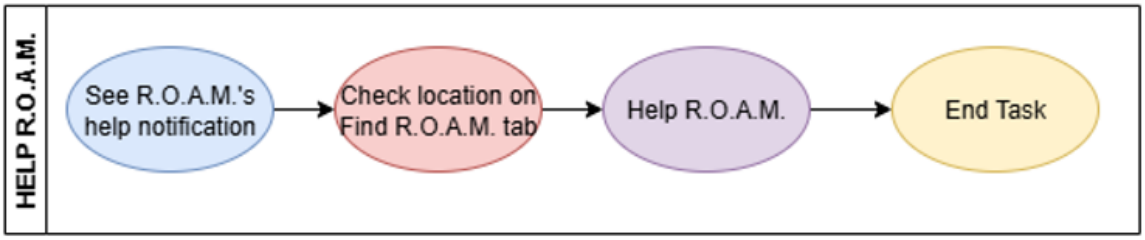


Figure 7.8 — Case Flow 2

The "Help R.O.A.M." process from the process model and notation diagram was laid out conceptually previously, but this is how a user would move through the actions. First, you may get a notification that the robot is in distress and in need of help, from there you are able to access the tab on our website which gives the user the coordinates of R.O.A.M. This allows the user to go and find the robot to help it, whatever situation it ended up in. Then that task is completed.

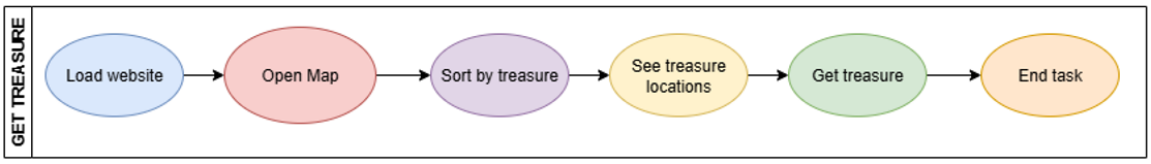


Figure 7.9 — Case Flow 3

This case flow in figure 7.8 outlines what a user may do to find treasure on this website. Based on the flow, you start by loading our website, opening the map view, sorting it by treasure, finding the location for this treasure, and then getting it. It is very straightforward and the process for trash and treasure are relatively similar. There are different ways to meet this goal through our website, but all had consideration put in to make the process easy and fun.

As shown in Figure 7.10, the website flowchart outlines a data-handling routine for R.O.A.M. 's web service, with the emphasis on how coordinate and image data are stored,

transmitted, and updated across the system. It begins by checking whether a previous HTTP POST request failed. And if it did, the system locally stores the collected data and resets the failure flag so that it can retry later. If no previous failure is detected, the system proceeds to send an HTTP POST request with the current data.

From there, the flow splits depending on the POST request's validity. If the request fails, the failure is logged and the system retains the data for future retry attempts. If successful, the incoming coordinates and image information are recorded in the SQLite database. Immediately after, a GET request is sent to the frontend, which prompts a browser-level update showing the robot's location and corresponding image data.

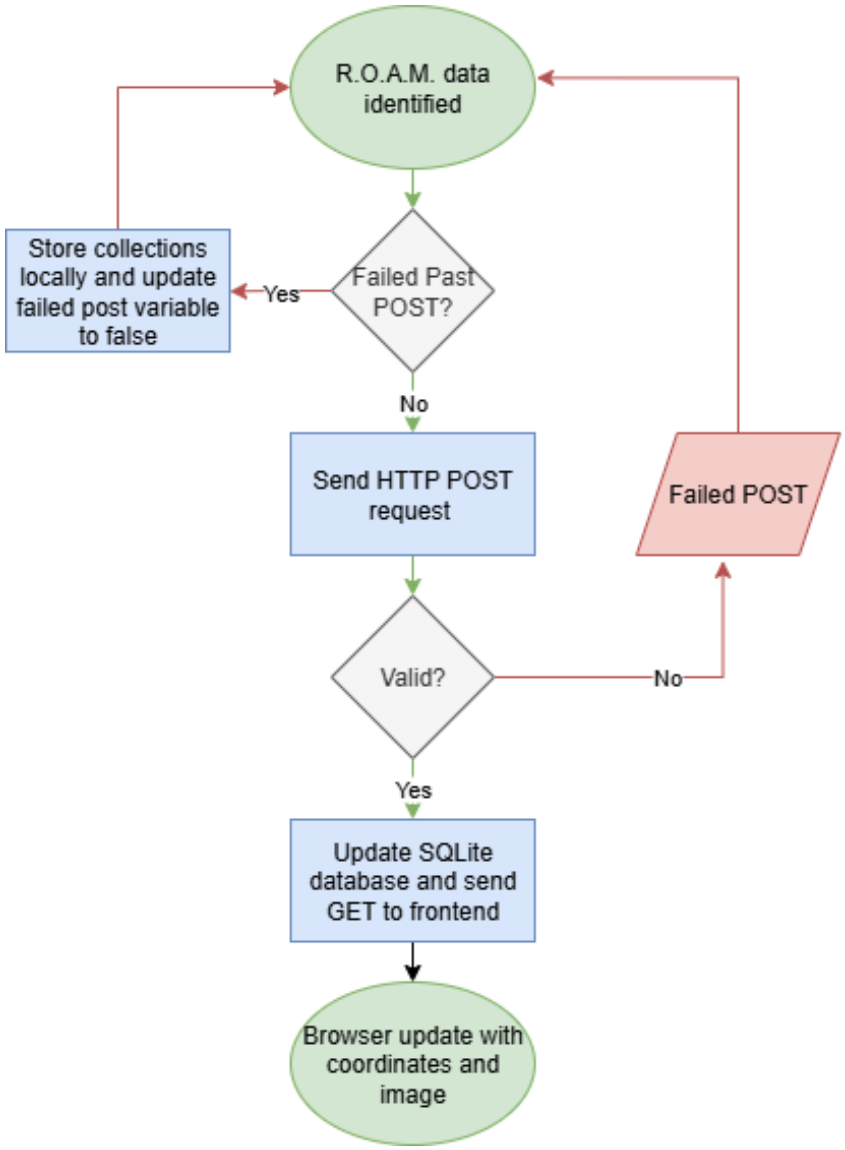


Figure 7.10 — Website Flowchart

7.3 SQLite Schema

The server that ingests ESP32 telemetry and classification events persists all incoming data in a local SQLite database. SQLite was chosen for its minimal configuration requirements, transactional integrity, and small footprint—qualities that align with R.O.A.M.'s need for a self-contained, reliable datastore on a lightweight cloud VM. The schema comprises three tables—readings, classifications, and failed_requests—each designed to balance query performance with fault tolerance. This section walks through the table structures, indexing strategy, the write-path logic with error handling, and the retry mechanism that guarantees eventual consistency.

7.3.1 Table Definitions

All valid sensor measurements flow into the readings table. Each row holds an automatically generated identifier, a UTC timestamp in ISO-8601 format, the ESP32's device ID, and one or more floating-point columns for metrics such as temperature or battery voltage. The classifications table records discrete object-detection events: every entry includes its own primary key, the moment the detection occurred, the robot's latitude and longitude, a text label ("trash" or "treasure"), and a file path pointing to the associated image stored on disk. To ensure that no data is ever dropped, malformed or otherwise unprocessable payloads are instead written to failed_requests. That table retains the original JSON string alongside an insertion timestamp, enabling later inspection or replay.

7.3.2 Indexing and Optimization

Although SQLite scales well for modest workloads, targeted indexing dramatically improves responsiveness for both live dashboards and historical queries. An index on readings(timestamp) makes it possible to rapidly assemble time-series graphs, while a composite index on classifications(latitude, longitude) accelerates map viewport queries—fetching only those detections that fall within the user's current view. The database runs in Write-Ahead Logging (WAL) mode with a synchronous setting of NORMAL, striking a balance between durability and write throughput. Occasional VACUUM and ANALYZE commands compact the database file and refresh internal statistics, ensuring query plans remain efficient as data grows[25].

7.3.3 Data Ingestion and Error Handling

When the Flask application receives a POST request at its /data endpoint, it immediately begins a short transaction. The handler parses the JSON payload to extract fields such as timestamp, device_id, sensor values, or classification details. It then attempts to insert these into the appropriate table within a single database connection context. If the insert succeeds, the transaction commits, and the server emits an internal trigger—often implemented as a lightweight GET request to a websocket endpoint or by pushing a notification to the front end—to update the map and data panels in the browser. If any DatabaseError arises during this insert (for example, a missing required field or a type mismatch), the exception is caught, the malformed JSON is persisted in failed_requests,

and a warning is logged. This approach isolates bad data without interrupting the overall ingestion pipeline and preserves the original payload for debugging or reprocessing.

7.3.4 Request-Handling State Machine and Retry Logic

Underneath the POST handler lies a simple state machine with four conceptual states: Idle, Ingest, Success, and Failure. In the Idle state, the server awaits a new HTTP request. Upon arrival, it transitions to Ingest, where it parses and writes data. A successful write moves the machine into Success, triggering the front-end update and returning a 200 OK to the ESP32 client. If the write fails, the machine enters Failure, logging the raw payload for later retry. Parallel to this, a background worker periodically scans failed_requests, re-parses each JSON entry, and re-attempts the original insert logic. Entries that finally succeed are removed from the failure table, closing the loop and guaranteeing that every piece of telemetry will eventually appear in the main schematic.

Chapter 8 System Fabrication

8.1 PCB Layout

This chapter will review all the different PCB designs that this project requires to have a functional connection between all parts. This will involve a main MCU PCB, a motor driver PCB, and three regulator PCBs. These will all be on separate boards to decrease noise caused by the motors, allow for efficient troubleshooting, and easy replacement if needed. We wanted to decrease any possibilities of having incidents where the regulator burned out on the main PCB, causing us to have to replace the entire PCB and not just one portion. This also allows us to have easier troubleshooting since many of the functions are offloaded from the main MCU PCB. In this chapter, you will see each design broken down into their own sections and talked about thoroughly.

8.1.1 MCU PCB

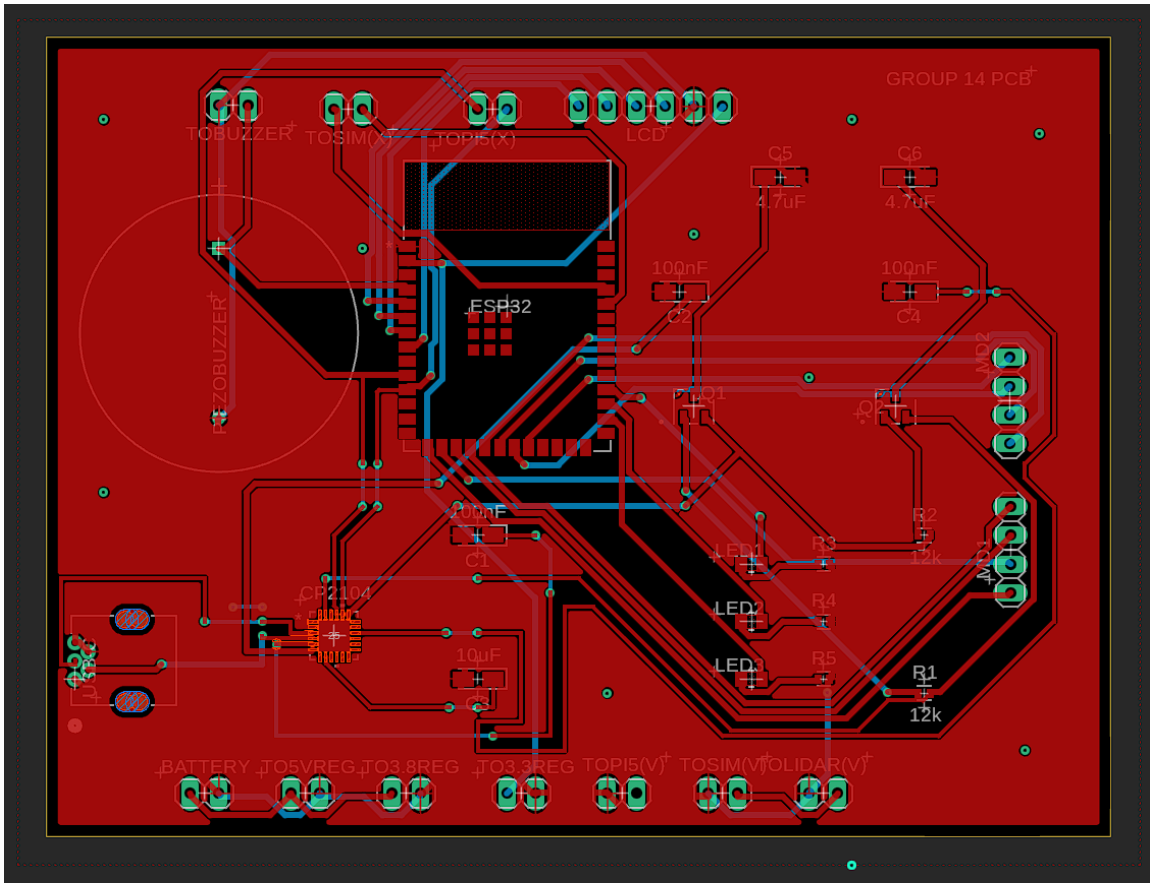


Figure 8.1 — Motor Driver PCB Design

The PCB above is the figure above is the PCB design for our system, this PCB is the heart of the system. Figure 6.1 presents a clearer version of this PCB as it was the schematic used to design this PCB.b This includes many components, the most notable and important being our ESP32, UART to USB converter chip, USB-C port, and the header pins. These all serve many incredibly important functions. The ESP32, which you

can see at the top center of this photo, it has all of our wiring going to many of the header ports. Many of those are the regulators that will be dispersing power through our overall project. Those are located on the bottom. This serves as the port between these two connections, we have the battery coming in to the board so that it does not directly connect to these devices, and we can split the power. This battery pin goes to the 5V regulator chip, the 3.8V regulator chip, and the 3.3V regulator chip. From there we have the regulators located externally, so replacement is more convenient if necessary. To the right of those power supply, we have the voltage connections that will be leaving the regulators and going to the voltage ports of our many devices. The three on the bottom right are for the power going to the Pi, Sim module, and LiDAR. To the right center of the PCB, is the logic for the motor drivers that is coming out of the ESP32 ports. This is where the logic will be calculated, and then the ESP32 will send a PWM signal based on that logic to these ports, which are connected on another PCB to the motor drivers. The adjacent connected side of this will be seen further below in the additional PCB designs we have. Looking at the top of this PCB there are our logic connections, this is for the LCD, Pi 5, SIM module. These will be the logic connections from our ESP32 that will send the data and receive data from these ports. A few of these are dev boards that will have their connections to those boards off of the main PCB. On the bottom left of this schematic we have our UART to USB converter, this is important for being able to transfer information between the devices, and it connects to the ESP32 RX and TX pins to configure this information. It also has the USB-C port that allows us to plug stuff into the PCB to configure it. We overall figured that having the rest of our PCBs separate to our main would allow us to be able to adjust or repurchase any PCBs separately that may have any issues or break in some way. This is why our configuration looks like it does, with our motor driver, and regulators being separate to the overall configuration.

8.1.2 Motor Driver

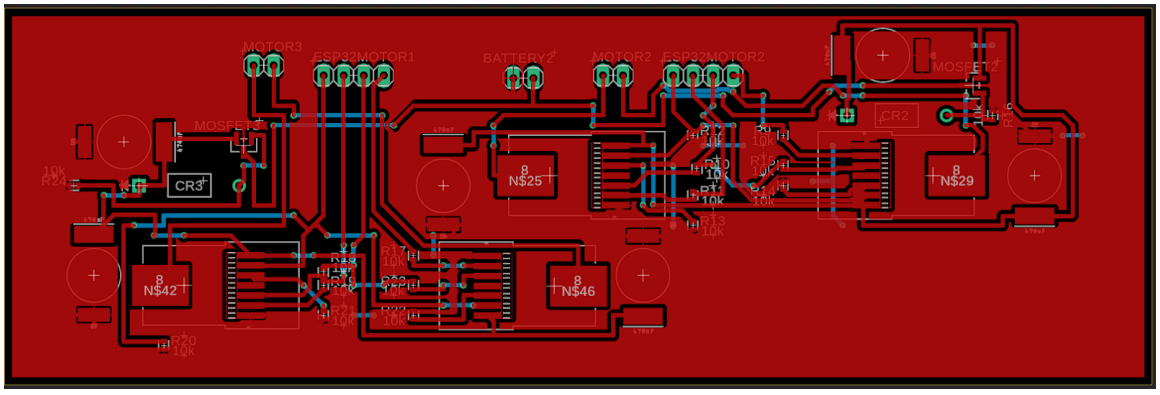


Figure 8.2 — Motor Driver PCB Design

This design was created to supply 12V motors with proper and controlled voltage directed by our ESP23. The motors that will be supplied power are the goBILDA 5202 Series Yellow Jacket motors which are brushed DC 12V motors. At 12V they have a no-load speed of 312 RPM and under no load they can draw .25A, with the load we have under stall conditions they are run closer to 9.2A. Our load will be moderate, meaning the motors cannot be weak. As for these, the motors are able to apply about 24kgcm, which

is suitable for our project. All of that went into consideration when figuring out what type of motor drivers that we needed to incorporate to this system. For that we found a few different options but settled on the BTS7960B since it offers the most integrable design and most versatile usage. This design is capable of handling up to 43A of continuous current. It also offers overcurrent, thermal protection, and protection for undervoltage. Each motor driver uses the BTS760B chip which gives it its H bridge configuration. This also means that there are a total of 8 pins that will be connected to our ESP32 to send signals to the motor drivers. This will mean that we have a significant amount of coding that goes into making the motor drivers send the correct PWM to the motors. There are a few header pins shown at the top of the PCB. On the left is going to motor one, then next to it is the logic pins going to the ESP32. These will work together by having the logic come in and down to the BTS79960B to turn certain mosfets on and off, then that signal will go to the motor to turn it at a certain speed and angle. Next to those on the right is the pin that is the battery coming into the motor driver system to give it power. Then on the right are the other headers going to the other motor, then the other inputs going to the other set of GPIO pins on the ESP32. These all work together to produce information for the motors to allow its movement.

As for the design of this motor driver, we started by organizing the components as best we could to resemble a similar organization to our schematic. Since the schematic had a lot of empty space, we tried to group some of the components more together to allow for a smaller board. This board, while still having empty space, has significantly less than it did. In this design we chose 20mil wide paths to route the PCB so we are able to achieve any maximum current that passes through without decreasing any efficiency. This design was modeled after the schematic and with the intent to be the most efficient we could create it to be, much time was spent figuring out the best way to configure the wiring to be clean so we can understand the traces visually. Having the header ports at the top was also important to be able to have these connections to the other PCBs as well as the other components.

8.1.3 Regulators

8.1.3.1 5.2V Regulator

This design shows the regulator PCB that we are using to step our voltage down to 5.2V so that we can supply the correct voltage to the rest of the circuit. The regulator being used is this LM257 which is a relatively popular kind. This one specifically is the adjustable type where you are able to use certain resistors to create the exact output that you want. This design was based off of the schematic version, with the goal to have the least amount of overlapping and the least amount of traces on the bottom of the PCB. The goal was not only for it to be visually pleasing but functional. We used 20 mil width tracing so it can allow 2A to the occasional 3A. While we do not expect to be continuously using 3A, there is a possibility the Pi will require it occasionally, this trace can endure that volume of current. We plan on needing 2A more continuously which is why we choose 20 mil. This design also has ports to accept the voltage of the battery coming from the MCU PCB, and then the voltage going out, back to the MCU PCB to connect to the header that powers the Pi. This chip was specifically made for the Pi as

5.2V was the suggested voltage requirement for this device. It is important to note this regulator has many stabilizing capacitors and uses them to reduce ripple and help get a stabilized output voltage.

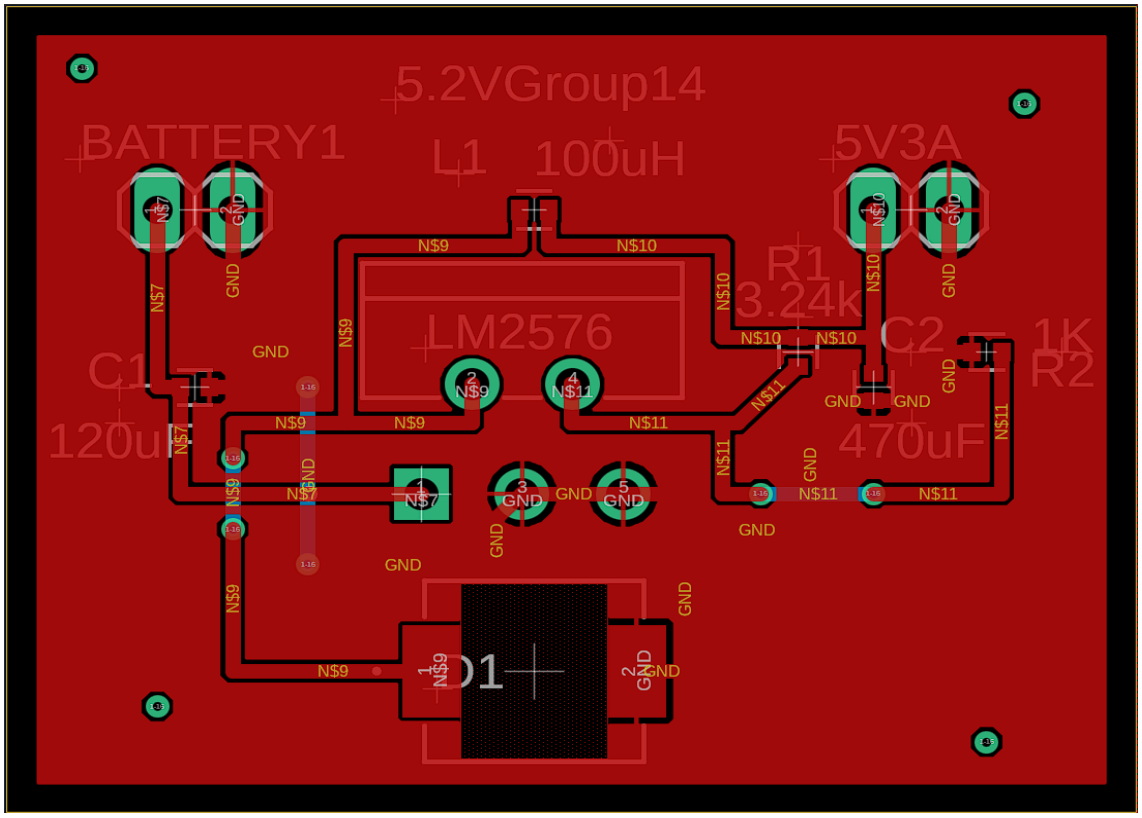


Figure 8.3 — 5.2V Regulator PCB Design

8.1.3.2 3.8V Regulator

The figure below displays the 3.8V regulator that is used for both the SIM module and the LiDAR sensor. This circuit uses the LM2576 adjustable chip to output an obscure voltage that is other than the common ones they offer. This means that there are two resistors that are available that will be used to configure the output voltage. The organization of this regulator is very similar to the one above where we have the battery coming in, or the connection on the MCU board we made that is going to the 3.8V header. Then we have the header that is going back to the MCU board to connect this power to the supply that was made for those devices. On the MCU is where the split will happen to send this voltage to the right equipment. Using 20 mil trace for this board will be important because it will let us get a solid 2A, and it can go up to 3A if needed, which it most likely should not need.

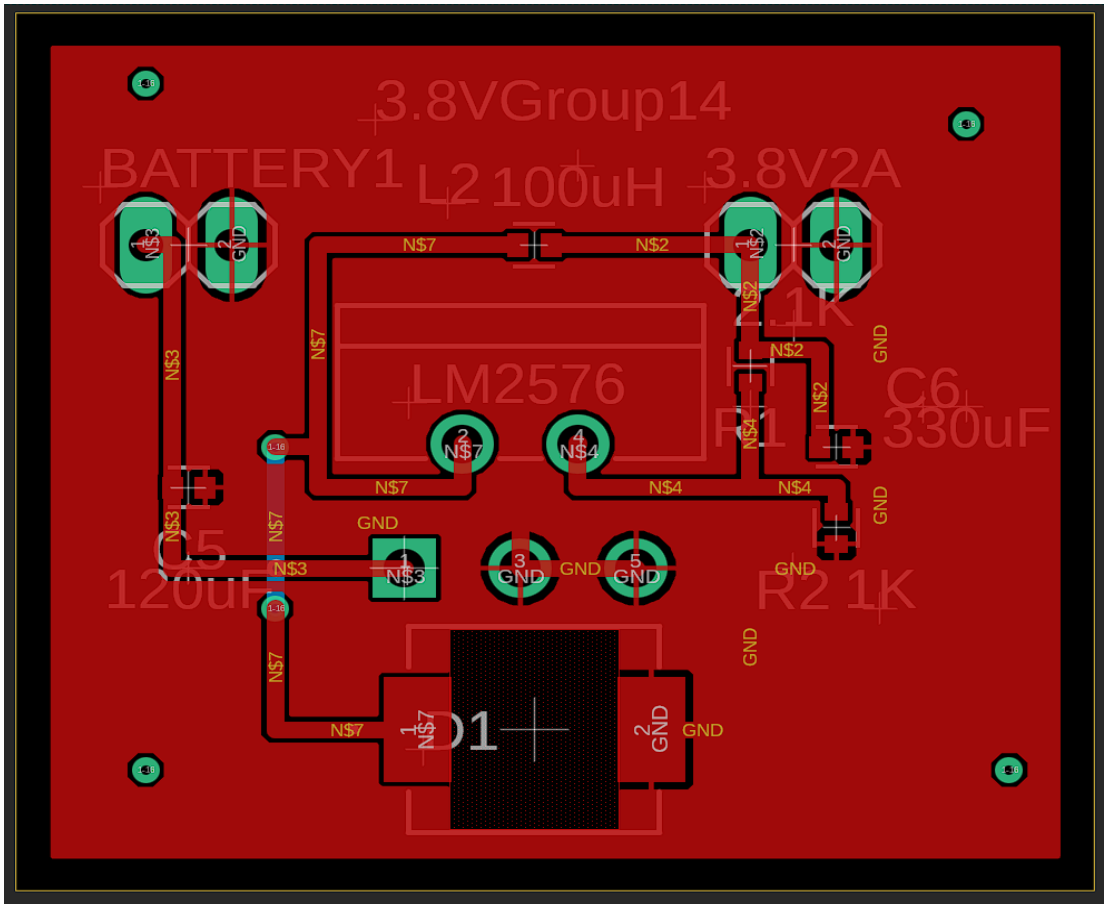


Figure 8.4 — 3.8V Regulator PCB Design

8.1.3.3 3.3V Regulator

The PCB design below describes a similar system to the two regulators before this where we have the battery coming in to supply this voltage that will be stepped down. It goes down to 3.3V which was designed to supply proper voltage to the ESP32. This means that the LM2576 does not have to be used with the adjustable one, the regulator chip is available with the 3.3V version as this is a very commonly used one. This means that we can easily configure this circuit without the use of the feedback resistors. This will receive an input from the MCU board that will be the battery voltage, then it will supply the proper 3.3V back to the MCU board where the voltage will be split into any of the needed devices. The only one that truly needs this is the ESP32.

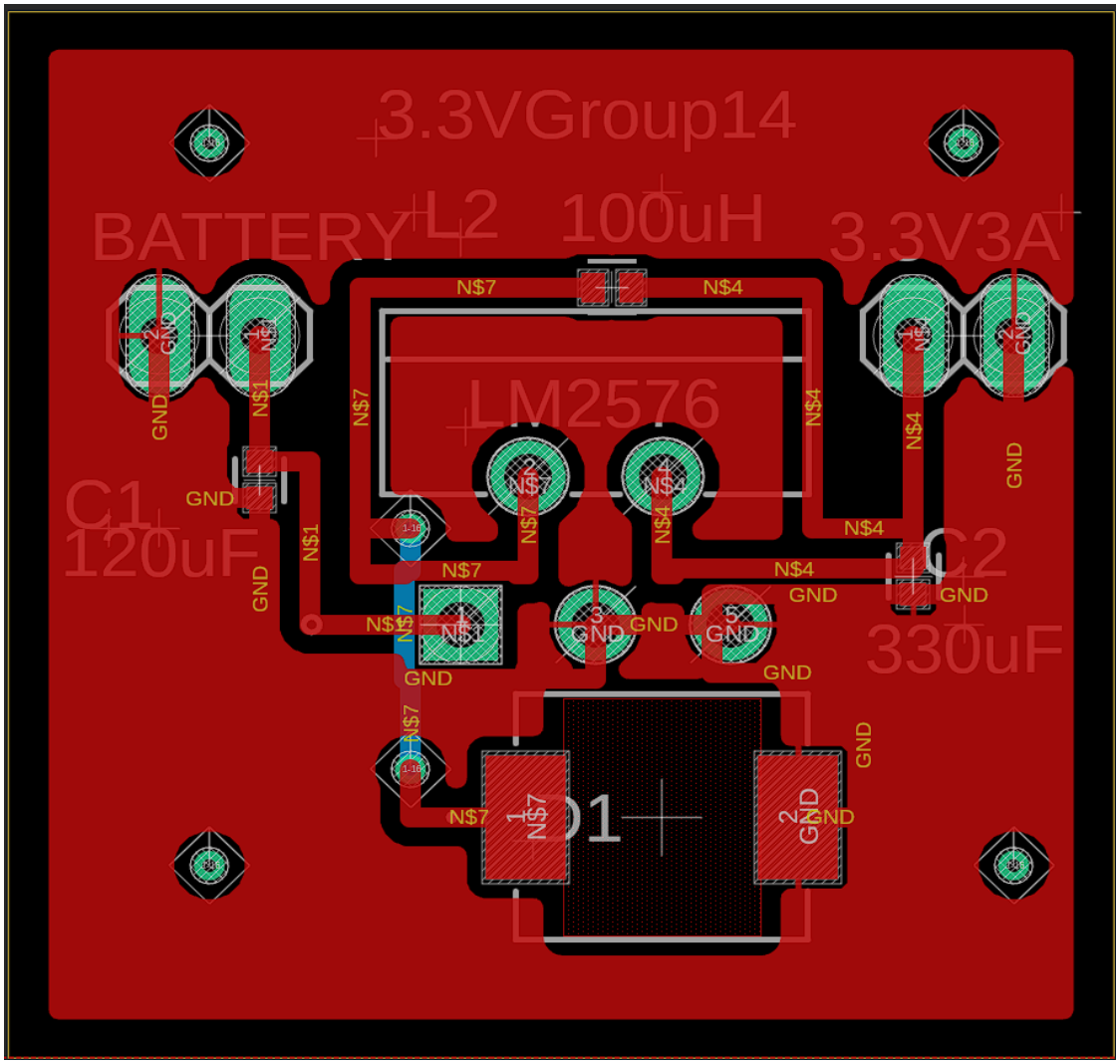


Figure 8.5 — 3.3V Regulator PCB Design

8.1.4 Conclusion

Overall, the design was made with the intention of being the most user-friendly and applicable circuits. The separate PCB designs allow us to simplify noise and troubleshoot these devices for replacements much easier. There are several components that interact with each other on the main MCU PCB being the ESP32, the UART converter, and the USB-C port. They all require power connections and logic connections, which exist on this MCU board. They will have header pins to off load onto other PCBs or development boards. This will allow us to be more customizable with our design. Needing mostly 2A, we choose 20 mil traces for these designs to efficiently transfer the power and hopefully avoid any thermal problems. The motor drivers being offloaded to another PCB additionally allowed for more protection against noise. More so, finding regulator circuits that could be customizable for specific voltage allowed us to have a very simplistic design to convert power. This system was created with efficiency, and convenience in mind as we hope to continue to develop this project and keep finding ways to improve our design.

Chapter 9 Prototype and System Testing

9.1 Hardware Testing

9.1.1 ESP32

The ESP32 being an integral part of this system has been used in all applications of our hardware testing. This device was selected as it gave a range of use for its pins, to develop this project we had to ensure that it functioned with all portions of our project meaning we worked on incorporating it to the SIM module, HMI interface, website, and more. Testing will be outlined further below as we talk about each instance of its use and how all parts will be integrated together through the ESP32. Additionally, this device will be continued to be used as we move into Senior Design 2, in the end we will need to integrate all components together and have it be able to withstand that amount of data, physically with its logic pins, and power wise.

9.1.2 SIM Module

Throughout prototyping, a lot of testing went into both the hardware components and the software components. The hardware that was associated with the SIM7600G-H module was the Esp32 module, sim card, gps, and antenna. The first thing tested was making sure everything was powered.The power for the sim module came from a computer via a micro usb port. The board provided the power for the antenna and the gps module, and the esp32 was also plugged into a computer. The good thing about the modules is that there are led's that show the status, so not a lot of code was needed for that part. Once this was confirmed, I started to write code for getting the GPS coordinates. This was done by using commands like AT+CGPSINFO, after the network was verified by AT+CREG and AT+CGATT, as shown in appendix d "GPS code". Since the SIM module was connected to the computer, this allowed us to send AT commands manually through the serial terminal, and see real time responses.

The GPS module performed really well while testing it. In areas where the sky was visible, the module was able to lock onto satellites and return accurate coordinates relatively quickly. I validated the data by plugging in the longitude and latitude coordinates to google earth and comparing them to where we got the data from. The module was consistent, accurate and stable over the tests I did. The GNSS antenna was proven to be effective.

I also evaluated the SIM7600G-H ability to establish a connection to cellular networks. Using a mint mobile sim card, I configured the module with the proper APN setting and verified registration with the network using AT commands. Once connected I tested the modules HTTP capabilities by sending curl requests to the server. The way I did this was by connecting my computer to the cell module and sending post requests to the server with git bash. This completely worked, and by using Ookla speed test, I found that the modem was getting speeds up to 45Mbps upload and download.

Although it performed well connected to the computer, issues started to rise when I tried to send POST's through the esp32 using AT commands.The ESP32 was intended to serve as the autonomous controller responsible for issuing AT commands to the SIM7600G-H and relaying GPS data to the server. However, the ESP32 failed to complete a successful HTTPS POST to the server. This suggests that there may be a conflict related to TLS handling, memory limitations, or certificate verification when operating the module through the ESP32's UART interface. I tried for days debugging and trying to get to the root of the issue with no luck. I even tried to contact the manufactures of the chip, sim module, and even mint mobile themselves. Performance testing confirmed that when the module is connected via USB, it can gather GPS data and transmit it to the server in under a second. Server responses were near-instantaneous, and map updates occurred in real time. This suggests that the performance bottleneck lies not within the SIM7600G-H itself but rather within the microcontroller integration. I have not yet isolated the root cause of the failure, but further debugging and reconfiguration of the ESP32-SIM7600 UART communication path is underway. My next step, which will be in SD2, is going to be talking to professors who deal with microcontrollers and know a lot more about protocols than me.

For the second phase of development (SD2), my plan is also to maintain the SIM7600G-H module as our core communication and GPS unit. My goal is to establish a consistent, programmatic data pipeline from the ESP32 to the server using the SIM module as the bridge. If this continues to be unsuccessful, I may pivot to an alternative strategy. Since the SIM7600G-H supports SMS messaging and I have verified that text messages can be sent and received through it, I may leverage a workaround involving a Google Voice number. In this configuration, the ESP32 could send SMS messages containing GPS coordinates to the Google Voice account, and a separate script could extract the messages and forward them to the server via curl requests. This path is less direct but would still allow us to collect and visualize remote data reliably.

In summary, the SIM7600G-H has shown solid performance in both GPS acquisition and data transmission when tested independently. The key challenge ahead is enabling seamless communication between the ESP32 and the module, which is critical for the system's full autonomy in the next development stage.

9.1.3 Raspberry Pi 5

9.1.3.1 Raspberry pi Camera module 3 Integration

The process of testing and integrating the raspberry pi camera module 3 with the raspberry pi started with making sure the raspberry pi had the tools to check and report back the out of the camera module. To do this I downloaded libcamera which is a bundle of command-line tools used to access a camera from the command prompt. This however was the improper tools to install for this version of raspberry pi's OS called bullseye, consequently whenever I tried accessing the camera through the libcamera commands I received a command not found. The correct set of tools is called rpicam. Now with rpicam I was able to see if the raspberry pi could detect the camera module. Initially, the camera was not detected and after going through raspberry pi's config files to determine

camera autodetect was on, I thought the physical connection might be damaged as ribbon cables can be fragile.

After replacing the ribbon cable that connects the camera to the Pi there was still no camera detected. So, if this was an issue it wasn't the only one. I would later find out the ribbon cables can be knocked out of the socket with light to moderate force making the camera undetectable. There will need to be extra securing and stabilizing of the ribbon cable within roam to ensure it doesn't become disconnected during deployment. It was then that I learned more about how the raspberry pi operates. The raspberry pi's device tree doesn't detect CSI connected devices even if the camera's imaging sensor is the default sensor, the device tree still needs to be expanded to expect it. This only required a change to the raspberry pi's config file to add the camera's imaging sensor to the device tree. Finally, the camera worked, using the rpicam-hello command I was able to see the camera being detected and get a preview of the camera working.

9.1.3.2 Raspberry Pi to ESP32 Integration

Ensuring that the raspberry pi is able to communicate with the ESP32 is an important first step for the development of R.O.A.M. This will be done first using UART. Since UART is a simpler embedded communication interface we can verify the connection between the devices and test the core functionality between integrated components more easily.

For testing the UART abilities of the raspberry pi including sending and receiving messages can be done in a few ways. The way that requires the least components and all equipment was on hand for is a loopback test. This is done by connecting a wire between the raspberry pi's TXD and RXD which are the sending and receiving IOs for UART. These are GPIO 14 and 15 on the raspberry pi. This allows for the sending and receiving of UART messages solely on the raspberry pi testing both functions at once.

The loopback test required the creation of a python script which would send a UART message then compare received UART messages to the sent message to see if there were no errors. I later discovered a command-line based serial communication program that could be easily installed and used to test the functionality of UART called Minicom. This was then used in conjunction with the python script to verify correctness. After confirming the wires connection and receiving nothing of what was sent in either python script or Minicom, I know the problem was elsewhere. This is when I reviewed the raspberry pi's configuration menu and files for UART options. In the Raspberry pi's main configuration menu there was an option for enabling UART under Interface options which I then enabled, this however didn't solve the issue. After rewriting the python script to provide more information for testing, I then discovered the UART flag within config.txt was not set which means UART over hardware wasn't enabled and the raspberry pi was defaulting to UART over bluetooth. So, after adding to the config.txt file "enable_uart=1" I ran the test code and got positive results. The results of this test can be seen in appendix F.

With the loopback test resulting in a positive and confirming the raspberry pi's ability to send and receive UART it was time to test integrating the raspberry pi and the ESP32.

This can be easily tested by setting up the ESP32 with test code that will simply parse and display received UART. To enable the two machines to communicate we first needed to connect the Raspberry pi's Tx to the ESP32's Rx GPIO ports. The ESP32 was only ever able to display the unknown character symbol and only did so once upon uploading and running the program. This is the only result we got after many testing and debugging attempts. Clearly there is a fundamental concept about UART receiving and parsing that we are missing. With the intended integration of SPI at the end of SD2, we will need to quickly figure out the error with receiving UART to verify the functions of other integrated systems before we move onto SPI.

9.1.4 LiDAR

The integration of the YDLidar X4 Pro into the R.O.A.M. platform begins with a clear assessment of both the hardware and software requirements. Physically, the X4 Pro demands a stable 5 V supply at up to 500 mA peak current, yet our existing power bus provides only 3.8 V. To bridge this gap, a high-efficiency step-up (boost) converter must be selected, one that can deliver sufficient current without introducing electrical noise into our analog sensors. Early evaluations suggest a compact, switch-mode regulator with integrated LC filtering to attenuate switching spikes will meet our needs while fitting within the tight enclosure constraints of the robot chassis.

Mechanically, securing the X4 Pro to the robot's frame requires attention to vibration damping and line-of-sight considerations. We plan to fabricate a laser-cut acrylic mount lined with silicone rubber gaskets to isolate the rotational scanner from high-frequency vibrations. The USB connector must be strain-relieved to prevent dislodging during operation, and shielded cabling will be routed away from high-current traces to minimize electromagnetic interference. A small custom PCB may also be designed to break out the boost converter's inputs and outputs, along with indicator LEDs for power and fault status, to facilitate on-robot troubleshooting.

9.1.5 HMI Design

To allow R.O.A.M. to interact with the world around it, we have included an LCD screen which will read out what the object identification code sees. This will be in combination with additional LED lights that will also light up a similar color to the LCD screen. The goal of this will be to have a way for humans who interact with R.O.A.M. to understand what its purpose is. We also have a function that will signal the website that roam needs help, this physically will be shown to the environment around roam by a red blinking LED, a screen on the LCD that will say "Help! Call XXX-XXX-XXXX" which will be filled in with a phone number at a later date. This help mode will also include a buzzer noise to draw attention to the robot. The help mode will stay activated until the issue is resolved and someone manually turns it off. To test this hardware we used the ESP32 to connect all of the necessary pins on the LCD screen which included connecting the 3.3V, ground, and all of the other logic pins, making sure those were properly defined on the code. We then connected the LED lights with a resistor from anode to the ESP32, the cathode going to ground. Additionally, we connected the positive end of the buzzer to a pin on the ESP32 and the negative to ground. Making sure all the grounds were tied

together, we were able to create a code that had the right pins included and that is listed in appendix D under “Integration of HMI”. This code will be demonstrated in the demo video, and for now it takes just inputs off of the serial monitor to demonstrate the functionality we hop to achieve with the additional code.

We spent a lot of time attempting to integrate this system with the Raspberry Pi which requires just the RX pin. In the end these two systems will be integrated as the main function is to be able to have a visual display of what the object detection code does. On the ESP32, the RX pin is the only one needed as the ESP32 will not be communicating back to the Pi, just receiving information. On the Pi the TX pin will need to be used since this is the way we are exporting the information. The only issue we came across was being able to receive information properly, we were getting symbols and corrupted information through UART, which was not what we expected. This was mentioned within the Raspberry Pi section of this chapter, as the issue was believed to stem from either the code we developed or that device’s transmission. Because of this, more time will be dedicated in senior design 2 to developing this connection, since we want this system to indicate what the object detection and LiDAR reads. Below is a photo of the breadboard testing that has this system connected together and working.

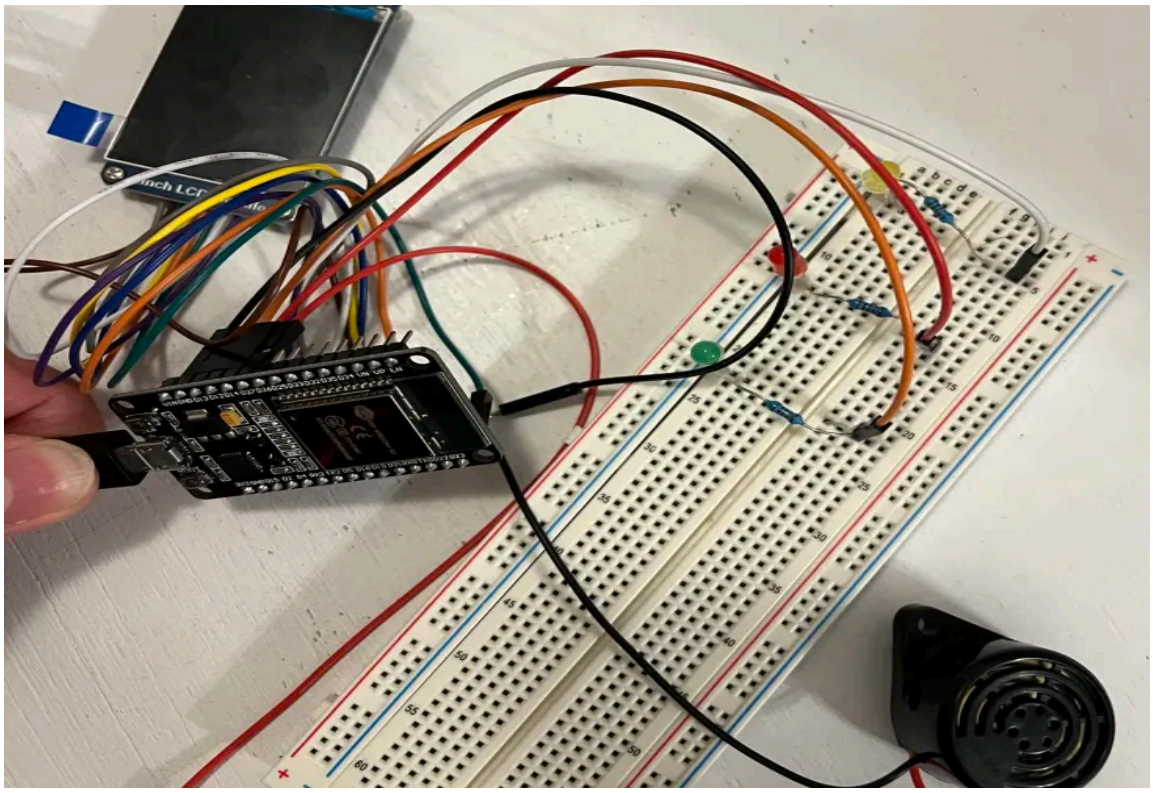


Figure 9.1 — HMI Breadboard Testing

9.1.7 Motors

To test the motors, our group has gone through the process of using the development board for the BTS7960 which uses the same circuit and chip that we plan to use for our motor driver PCB. We wanted to be able to test how efficient the motors were, how efficient our choice in motor driver chip was, and how we can send logic to these. Below is a photo you can see of the configuration, the motors we received had the ability to use logic encoders which we are not incorporating, meaning we just have the power cords connected to the development board. This board uses a 12V temporary power supply which is not the final power supply, just one that was on hand to test the motors as we continued to develop our system. The motor driver has forwarded and reverse driving capability which is what we wanted to test, we wanted to find out how the ESP32 would physically send the data to the motors. In the code we were able to adjust the PWM signal by shortening the frequency that was sent, which made the motors turn faster. We plan on developing this idea and continuing to develop a code that will be able to read an input from the ESP32 that is received from the Pi. The LiDAR will send information to the Pi about its surroundings, which will be interpreted by the Pi and then it will send that data to the ESP32, telling the motor driver what to do with the motors. In senior design two we plan on training the LiDAR to receive information and train the Pi to interpret it so that we can control the motors instead of manually sending information.

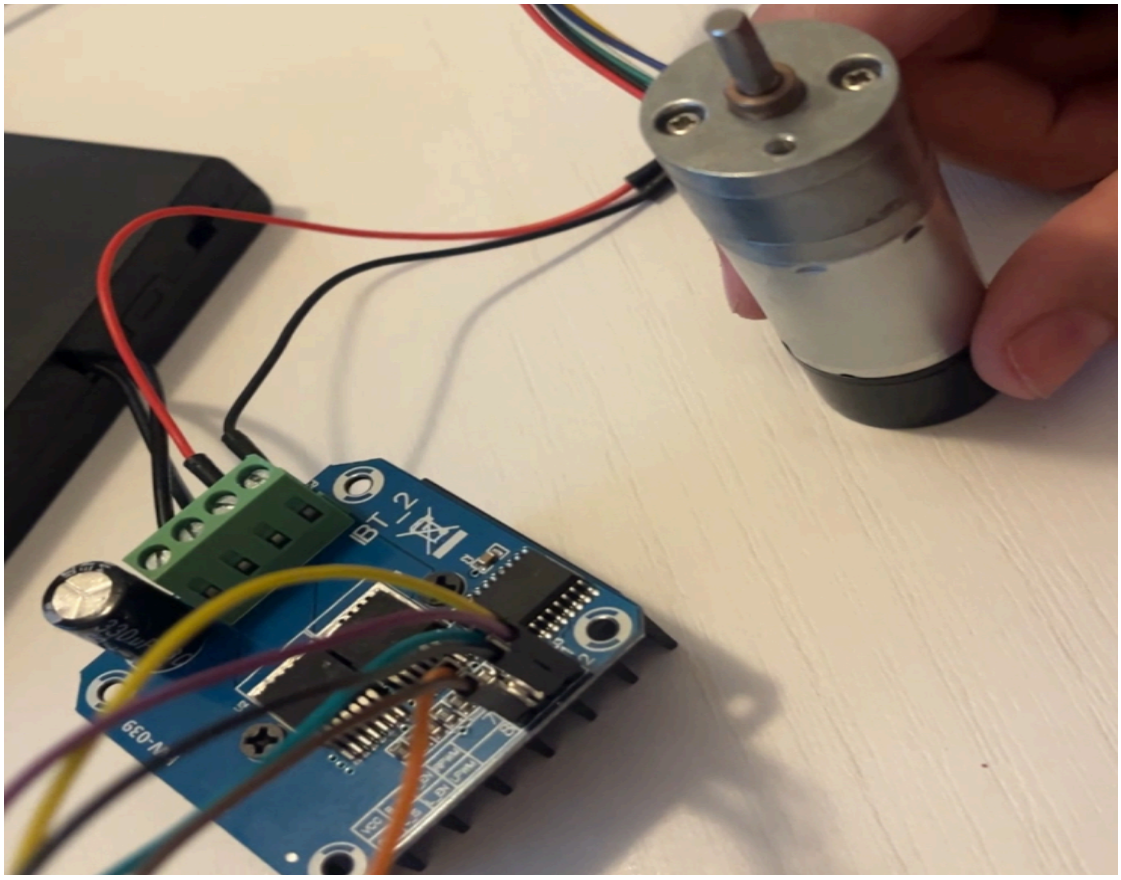


Figure 9.2 — Motor Driver to Motor

9.2 Software Testing

9.2.1 Object Classification Model

When beginning to develop and test the object classification the most important component that I have control of is the database. A good quality database significantly increases the accuracy of the object classification model and makes a more reliable system. After training the model, it immediately attempts the test images provided with the dataset. With these images the trained model performed very well with an mAP@50 of 88.1%, a precision of 94.3%, and a recall of 84.4% on roboflow This means with the given test images the model detected at least 50% of the object 88.1% of the time, had false positives 5.7% of the time, and missed objects 15.6% of the time. This performance is good and seemed like we had a well constructed database. However, Upon the first real world test of the trained model, this trained model was able to detect trash at atleast a 60% confidence level within 30 inches and was able to detect some basic furniture items it was trained from 42 inches away or more when given the proper angle. This first test was not done on the raspberry pi but on a laptop using an external webcam. This was only to check functionality and accuracy of the trained model without needing the integrated raspberry pi and camera system. Additionally, the accuracy of the object classification model was assessed in a non-standardized environment which R.O.A.M. would not likely find itself, because of this it's hard to determine the effectiveness of the trained model in real world settings.

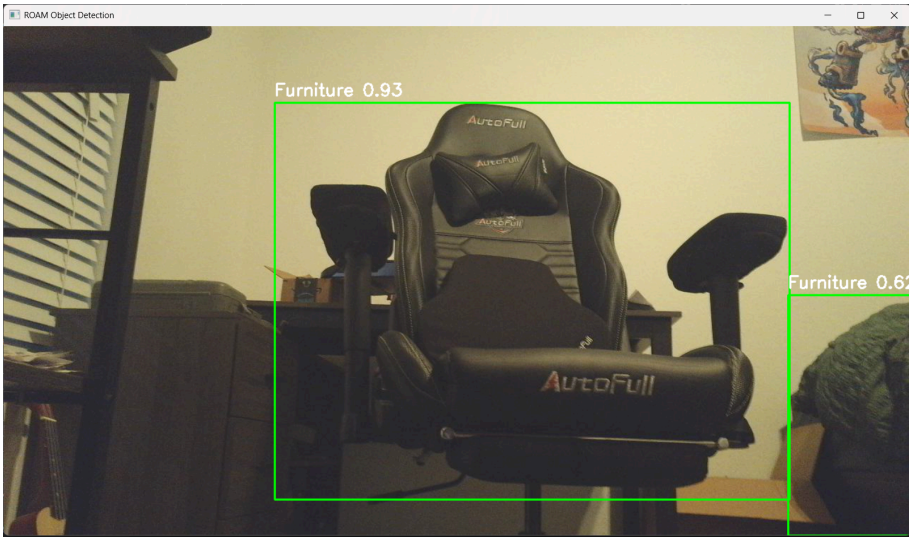


Figure 9.3 — Object Classification model: First Test

The second test was done from the raspberry pi 5 with the raspberry pi camera model 3 integrated. This test started with error, first because of a recent change in the raspberry pi camera stack OpenCV is no longer able to directly capture video from the camera. Instead I had to use Picamera 2 to capture the video, this was a small change since Picamera 2 was only used to capture the video and OpenCV's functions like cv2,cvtcolor which is very important for changing the coloring of images from BGR to RGB. Next, I discovered I had a conflict between two libraries Numpy and SimpleJpeg. The conflict was easy to resolve as they just had mismatched versions and needed to be uninstalled then reinstalled to compatible versions. Finally, we were able to see the trained model perform very similarly to how it did in test 1. Performing okay with larger furniture objects like chairs and couches that it was well trained on, but performing poorly when it comes to identifying trash.

Going forward in SD2 we should create a standardized testing environment or at the least have a list of conditions that must be met to test R.O.A.M. including but not limited to: Outdoors, a set number of trash objects in a set position, at least 1 furniture object. The performance of the trained model during tests leaves much to be desired meaning a much more robust and accurate database is required. The plan for SD2 is to entirely recreate the database with better images that are labeled better. There also needs to be a big improvement in the test photos for the model to better predict its effectiveness before testing on the raspberry pi. Once this improvement is made, then we can implement SORT to track objects as they are being detected to minimize the amount of repeat identifications. The program also ran at a poor framerate, so if possible upgrading the raspberry pi 5 with a HAT or upgrading the edge device entirely should be considered.

9.2.2 LiDAR Integration

On the software side, the X4 Pro communicates via a TTL-to-USB bridge chip and is supported by the YDLidar SDK, which provides sample code in both C++ and Python. Initial attempts to compile the SDK on the Windows 11 experimental setup have exposed missing dependencies in CMakeLists.txt, and a crash on launch (Error c0000005 at address 0x00000001) potentially stemming from an unhandled pointer dereference in the Windows-oriented demo binary. To resolve this, we will strip out the Windows-specific sections, reorganize the build to use the native libusb stack, and define custom udev rules so that the ttyUSB0 interface is accessible without root privileges when testing on the Windows OS. Once the build passes without errors, test applications will be run to verify successful enumeration of the LiDAR device over USB.

Following a successful build, attention turns to integrating the data stream into the perception pipeline. The SDK's callback interface can hand us raw angle–distance pairs at up to 10 Hz, but real-time mapping demands buffering and threading strategies to avoid dropped frames. We will develop a lightweight Python wrapper around the C++ library, exposing scan publications as ROS 2 LaserScan topics, which seamlessly feed

into our SLAM and obstacle-avoidance nodes. A series of calibration tests in a controlled environment, using reflective targets at known distances, will quantify the device's angular resolution and range accuracy.

Performance metrics such as scan frequency, point cloud density, and CPU utilization on the Pi 5 will be logged over extended runs to assess thermal throttling and frame stability. If the Raspberry Pi 5's native USB controller proves to be a bottleneck, we may offload pre-processing of scan data to a dedicated microcontroller or upgrade to a USB 3.0 HAT adapter. Ultimately, the goal is to achieve a consistent 8–10 Hz sweep with minimal jitter, enabling the robot to update its occupancy grid in real time when entering the field.

Moving forward, this integration experiment serves as the foundation for advanced spatial awareness on R.O.A.M. With reliable LiDAR data, the robot can construct robust environmental maps and perform dynamic obstacle avoidance. The next phase includes outdoor validation over varied terrains and lighting conditions, followed by stress-testing under sustained operation. By the end of the next system development stage, we aim to have an end-to-end LiDAR integration that not only powers the perception stack but also meets our benchmarks for power efficiency, mechanical resilience, and software reliability.

9.2.3 ESP32 and SIM 7600 Web Interface

The API-endpoints and HTTP(S)-requests compatibility was made crucial in the communication experimentation between the website and ESP32; reinforced by the ability to adapt to the SIM7600's AT command sequence in tandem. The web interface between ESP32 and SIM7600 is essentially constructed upon an asynchronous, command-response model over a physical UART link. The SIM7600, as a 4G LTE modem, supports a rich AT command set that allows for network registration, GPRS attachment, PDP activation, and the initiating of HTTP(S) requests. On the software side, the expectation is that the ESP32 can drive the SIM7600 through platform-appropriate drivers or libraries (like the documentation's included AT command wrappers), issuing sequences such as module restart, APN setting, and data transmission with appropriate timing and response parsing. Under ideal conditions the SIM7600's responses include server pings, successful HTTP status returns, and payload echo from the endpoint. However, when this workflow is ported to ESP32, results may diverge, manifesting as timeouts, null responses, or incomplete data transfer, which would all lead us back to careful troubleshooting of both the hardware and firmware integration.

9.2.3.1 IoT Certificates for the ESP32

Adding secure, certificate-based communication is no longer optional for modern IoT deployments, and SSL/TLS certificates are fundamental for protecting server APIs and maintaining data privacy, integrity, and device authenticity over cellular channels. The experimentation with SIM7600 revealed that reliable HTTPS transactions, especially when the module is commanded from an ESP32 host, almost always require that the correct SSL/TLS certificate chain be installed both on the SIM7600 (for server and possibly client verification) and in some cases, on the ESP32 itself depending on how SSL termination and data handling are split between the two components. Uploading the

proper PEM-formatted certificate to the SIM7600 via AT+CCERTDOWN is necessary, but not always sufficient, if ESP32's firmware doesn't correctly associate or reference the certificate in the SSL context.

On a Windows environment, certificates are typically managed via the OS's pre-established cryptographic libraries and trusted store, enabling tools like curl to leverage the installed root and intermediate certificates automatically. This configuration is less straightforward in microcontroller environments. For the ESP32, certificates must be explicitly embedded into firmware in addition to its built-in signature, as a .crt file, and loaded at runtime for TLS handshakes, using mechanisms built into libraries like 'WiFiClientSecure' or 'ESP_SSLClient'. For the SIM7600, certificates must be downloaded to the module using the AT+CCERTDOWN command, referenced in subsequent configuration via AT+CSSLCFG, specifying the certificate for the authentication mode, and finally listed with AT+CCERTLIST with a verbose output for troubleshooting. Certificate management commands on the SIM7600 allow for flexible use cases: server verification alone, mutual authentication, or even client-only configurations for specific IoT deployments.

However, establishing a reliable certificate flow is not trivial. There are strict requirements regarding certificate length, format, and validity, for example, only PEM or DER, typically under 10 KB per file, encoded correctly without superfluous whitespace or unsupported extensions. A frequent source of troubleshooting is the occurrence of errors best summarized as "CA missed" or "Handshake error", which typically would indicate that the server's SSL certificate chain is not fully represented in the SIM7600's context, or that the ESP32's firmware has failed to parse or present the correct CA to the module. Recent transitions in industry, such as Azure deprecating the Baltimore CyberTrust Root in favor of DigiCert Global Root G2, have caused connectivity to break unless the new CA is present, an issue especially noticeable as cloud providers update their trust lists. Another challenge is storage, the ESP32 is resource-constrained and must sometimes offload certificate management to external flash in the presence of images in the future integrations, while the SIM7600 module has limited file slots and maximum certificate counts. Rotation and renewal of certificate chains add further complexity, and these issues are exacerbated when scaling and should be addressed if commercial deployment is ever considered.

In parallel, it is important to consider the role of the underlying cellular network. This system employs Mint Mobile SIMs, which operate on T-Mobile's infrastructure. Although data services were confirmed to work on PC hosts and mobile phones, IoT devices such as the ESP32-SIM7600 pairing may be subject to undocumented restrictions if not formally certified for operation on that network. T-Mobile, like many carriers, reserves the right to limit access for devices that are unverified or unwhitelisted at the IMEI level, which can result in silent PDP rejections, inconsistent packet delivery, or filtering of traffic at the backend. These effects might not be immediately visible through conventional debugging and can introduce an additional axis of failure when deploying cellular modules in production environments. The absence of formal certification for the ESP32-SIM7600 hardware stack could very well be contributing to the connectivity

issues observed during real-world testing, and merits deeper investigation before considering the device for its field experimentation.

9.2.3.2 AT Commands and API Endpoints

To ensure that HTTP(S) communication between the ESP32-SIM7600 stack and the server backend remained reliable, significant attention was devoted to harmonizing the AT command sequences used on the device with the expectations of the Flask API served behind Nginx. The SIM7600 module operates over a serial interface via AT commands—structured queries like AT+HTTPINIT, AT+HTTPPARA="URL",<endpoint>, and AT+HTTPACTION=1—that initiate and manage web requests. These commands were issued by the ESP32 using carefully timed UART transmissions, governed by firmware routines that monitor and parse the modem's responses. On the server side, Flask's routing and response handling were adapted to ensure that requests from the SIM7600 could be interpreted correctly despite the module's limited HTTP header customization and nonstandard formatting. For instance, the module cannot specify custom User-Agent strings or negotiate complex TLS configurations, so Flask routes were constructed to respond predictably even to sparse requests, with liberal use of request.args.get() and content-type flexibility.

Nginx played a crucial role as a reverse proxy, translating incoming HTTPS traffic to the Flask application while maintaining certificate integrity for TLS handshakes initiated by the SIM7600. Special attention was given to timeout settings, maximum request body size, and 502/504 fallback conditions, all of which were tuned to accommodate the modem's slower round-trip latency and occasional retransmission behavior. Compatibility was verified using manual inspections of server logs, matching known SIM7600 AT commands with corresponding Flask log entries, and evaluating whether status codes and payloads echoed correctly from the API layer to the device. This process revealed the necessity of uniform response encoding (typically raw UTF-8 JSON) and predictable status headers (HTTP/1.1 200 OK) to prevent partial parsing or framing errors on the ESP32 side.

To capture real-world performance of our cellular IoT stack, we introduced a dedicated logging endpoint (/data) within the Flask application and leveraged Nginx as a reverse proxy to terminate TLS and forward decrypted POST requests. Every time the ESP32-SIM7600 issues an AT-initiated HTTPACTION, Nginx writes the incoming request to its access log before proxying to Flask. The Flask route then echoes the JSON payload—latitude, longitude, and timestamp—into the application log, creating a full-stack record of each telemetry transaction.

Below is an excerpt from the live server logs demonstrating seamless capture of SIM7600-originated payloads over HTTPS:

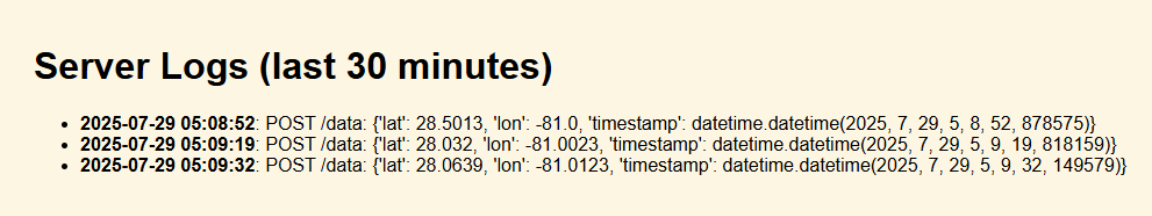


Figure 9.4 Roambot.dev Server Logs

Finally, robustness testing involved simulating network conditions and HTTP retries, confirming that the ESP32 could re-issue failed commands and parse Flask responses even under intermittent connectivity. This informed firmware routines for backoff timing and error handling, creating a bridge between device constraints and modern cloud-side expectations. The result is a tightly coupled request-response model in which every layer—from serial command syntax to web server routing—has been explicitly tuned to accommodate the idiosyncrasies of cellular IoT communication.

9.2.3.3 Next Steps

The critical question that arises is whether these integration pains are attributable mainly to physical-layer instabilities or to deeper software issues like incomplete AT command implementation or faulty TLS/SSL certificate management. Empirical results, as well as community discussions, indicate both factors are often at play. On the protocol side, persistent failures around HTTPS endpoints and MQTT TLS connections hint at certificate handling issues, which must be addressed at both the SIM7600's internal context and the ESP32 host's own software libraries. This interconnectedness urges a holistic view of a robust ESP32 and SIM7600 deployment not just complying with the datasheet but using a looping diagnostic workflow, where observed failures (e.g., HTTP 500, AT command errors, SSL handshake failures) drive both hardware and software reflection.

Chapter 10 Administrative Content

10.1 Budget Table

Table 10.1 — Budget Table

Category	Subcategory	Cost
Software	Autonomous driving	$50
	Object Classification	$50
	Website	$100
Hardware	Motors + Motor Control	$40
	PCB Design	$150
	Processing	$400
	Chassis	$120
	Power	$65
Overhead	Shared Resources	$20
	Misc Supplies	$50
	Wear and Tear	$10
	Contingency Reserve	$80
Total		$1135

10.2 Bill of Materials

Table 10.2 — Bill of Materials

Component	Sub-system	Quantity	Unit cost	Total
Raspberry Pi	Autonomous driving	1	$300	$300
LIDAR	Autonomous driving	1	$100	$100
Esp32 chip	Autonomous driving	1	$6	$6
PCB	All	5	$75	$250
Regulators	All	10	$10	$100
Camera	Trash Detection	1	$25	$25
Chassis car kit	All	1	$120	$120
Antenna	Website	1	$50	$50
Sim Card	Website	1	$20	$20
Cell Plan	Website	1	$25/month	$25/month
Domain	Website	1	$5	$5
Miscellaneous	All		$64	$64
Ethernet Cable	Autonomous Driving + Trash	1	$8	$8

	Detection			
Total				$1138

10.3 Work Distribution

Given our motivations and backgrounds, we all have chosen our topics to work on and split the workload with. As Claire is the only electrical engineer, we wanted to make sure she had enough with creating a PCB as well as enough power distribution work. As for Ethan, he has shown interest in the embedded software task, he has had a bit of experience in it and wants to do work to strengthen that. Daniel has shown interest in learning more about the machine learning tasks, and the object classification interested him. Lastly, Nathan has shown that he wants to learn more about autonomous driving and website development, which is why he was tasked with this. Across four members, we have further split into a 'buddy system' with Claire and Ethan in one pair and Daniel and Nathan in another. This was done to focus each person's efforts on one field at a time, and was brought up because of some similarities and differences in our tasks. When getting advice about senior design we were always told since it was such a long project, to find something that was enjoyable, something you were interested in. When you want to do the task at hand it will not feel like work, so we all found something we all would find enjoyable.

Table 10.3 — Work Distribution

Person	**Role / Work Distribution**
Ethan (main)	Embedded software for cellular communication.
Claire	
Claire (main)	Power supply, PCB design, and autonomous driving hardware.
Ethan	
Daniel (main)	Object classification CV software.
Nathan	
Nathan (main)	Autonomous driving software and website.

Daniel	

10.4 Milestones

Table 10.4 — Senior Design 1 Milestones

Senior Design I				
Start Date	Planned End Date	Req. End Date	Task	Status
5/12/2025	5/18/2025	5/20/2025	Team Recruitment	Completed
5/12/2025	5/20/2025	5/23/2025	Project Idea	Completed
5/25/2025	5/28/2025	5/30/2025	D&C Report	Completed
5/25/2025	7/01/2025	7/25/2025	CV Algorithm Design	Completed
6/03/2025	6/05/2025	6/06/2025	Update D&C on group website	Completed
6/02/2025	6/24/2025	6/28/2025	Prototype PCB design	Completed
6/21/2025	7/04/2025	7/07/2025	Midterm Report	Completed
5/25/2025	7/11/2025	7/25/2025	Accomplish Basic Object Classification	Completed
5/25/2025	7/11/2025	7/25/2025	Accomplish Basic Website	Completed
6/28/2025	7/21/2025	7/25/2025	Build Prototype	Completed
5/25/2025	7/28/2025	7/29/2025	SD1 Final Report (120 Page) & small demo video	Completed

Table 10.5 — Senior Design 2 Milestones

Senior Design II				
Start Date	Planned End Date	Req. End Date	Task	Description
		TBD	Review Report	Incomplete
		TBD	Finalize PCB	Incomplete
		TBD	Fix Prototype issues	Incomplete
		TBD	Assemble Final Product	Incomplete
		12/25	Live Demo	Incomplete

Chapter 11 Conclusion

This paper covers an extensive amount of information that will be the basis for the scope, research, development, testing, and prototyping for our robot. R.O.A.M. is an autonomous sidewalk tracking device that is designed to identify, classify, and report salvageable things that litter places like residential neighborhoods. The combination of object detection, autonomous navigation, real-time data reporting, GPS tracking, and an interactive website all will help us reach our goal of reducing waste. We wanted to address the fact that overconsumption and the unnecessary discarding of reusable goods has become an issue in many communities. To combat this, we hoped to use a community focused solution to promote reusing goods and relying on your neighbors to clean up the streets. Core functions in this involve object detection, website integration, GPS locating, and autonomous driving. To do this we used a combination of heavily researched electronics which include brushed DC motors, a LCD screen, a buzzer, LEDs, a GPS module, SIM module, Raspberry Pi, LiDAR, and an ESP32. These will all be responsible for many integrated parts of this project. The ESP32 and Pi will coordinate to transmit and analyze data using UART and other PWM logic to send information to other equipment. We will have the cellular module and GPS to send items and data to our website. On the website front, we will have an intuitive interface for people to explore our real time collection data and see the map based locations. This website will be the way for people to locate the items they may want to collect, it also doubles as a way to keep track of if R.O.A.M. gets stuck and needs assistance. We spent a lot of time researching the information we will use to build R.O.A.M. in the next semester. This will involve us continuing to test and integrate the systems we have in place so they can all communicate with each other and be integrated on one PCB. The PCB we have designed has significant peripherals which include our motor drivers, regulators, UART to USB converter, LED lights, and a buzzer. These were developed over the process of figuring out what we wanted to have on R.O.A.M. as it traveled to make it functional. In the testing phase we were able to organize certain portions of the core function into practice, and in the next semester there will be significant development to into continued integration and testing.

Appendix A

References

[1] ABLIC Inc. (2024) *Introduction - what is an LDO? what is a linear regulator?*, *ABLIC Inc.* Available at: https://www.ablic.com/en/semicon/products/power-management-ic/voltage-regulator-ldo/intro/ (Accessed: 05 July 2025).

[2] B. C. Soydas, "The impact of programming language choice on execution time when performing virtual simulation with a driver model: A comparison of C++ and Python performance using the Open Simulation Interface (OSI) in esmini," M.S. thesis, Dept. of Automotive Engineering, Chalmers Univ. of Tech., Gothenburg, Sweden, 2024.

[3] *Client challenge*. (n.d.). https://www.monolithicpower.com/en/learning/mpscholar/power-electronics/dc-dc-converters/buck-converters#:~:text=Buck%20converters%20employ%20a%20simple,the%20rest%20of%20the%20circuit.

[4] Derpanis, Konstantinos G. *Overview of the RANSAC Algorithm*. 2010.

[5] Energy, B. (2023, November 16). How many times is the lithium battery life cycle? *Benzo Energy / China Best Polymer Lithium-ion Battery Manufacturer,Lithium Ion Battery,Lipo Battery Pack,LiFePO4 Battery Pack, 18650 Batteries, Rc Battery Pack*. https://www.benzoenergy.com/blog/post/lithium-battery-life-cycle.html

[6] Environment America (2021, September 29). *Trash in America.* Environment America Research & Policy Center. https://environmentamerica.org/center/resources/trash-in-america-2/

[7] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000. [Online]. Available: http://opencv.org/

[8] *Home - Starship Technologies: Autonomous robot delivery - The future of delivery - today!* (2025, July 1). Starship Technologies: Autonomous Robot Delivery - the Future of Delivery - Today! https://www.starship.xyz/

[9] *How do brushed DC motors work? The need for regular maintenance explained | ASPINA*. (n.d.). ASPINA | Corporate Brand of Shinano Kenshi. https://www.aspina-group.com/en/learning-zone/columns/what-is/013/

[10] IEEE Robotics and Automation Society, *IEEE Standard Ontologies for Robotics and Automation*, IEEE Std 1872-2015, Apr. 2015. doi: 10.1109/IEEESTD.2015.7084073

[11] J. B. Sigongan *et al.*, "GULP: Solar-Powered Smart Garbage Segregation Bins with SMS Notification and Machine Learning Image Processing," *arXiv:2304.13040 [cs]*, Apr. 2023, doi: https://doi.org/10.25147/ijcsr.2017.001.1.142.

[12] Pranjal Malewar, "Clearbot, a solar-powered, marine trash-collecting robot," *Inceptive Mind*, Jan. 10, 2021. https://www.inceptivemind.com/clearbot-solar-powered-marine-trash-collecting-robot/17108/?utm_source

[13] R. Pereira, M. Couto, F. Ribeiro, R. Rua, J. Cunha, J. P. Fernandes, and J. Saraiva, "Energy efficiency across programming languages: how do energy, time, and memory relate?," in *Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering (SLE 2017)*, Vancouver, BC, Canada, Oct. 2017, pp. 256–267. doi: 10.1145/3136014.3136031

[14] G. Jocher, L. Stoken, A. Chaurasia, and J. Qiu, "YOLOv8: State-of-the-art real-time object detection and segmentation," GitHub repository, 2023. [Online]. Available: https://github.com/ultralytics/ultralytics

[15] M. Tan, R. Pang, and Q. V. Le, "EfficientDet: Scalable and efficient object detection," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Seattle, WA, USA, Jun. 2020, pp. 10781–10790.

[16] W. Liu et al., "SSD: Single shot multibox detector," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, Amsterdam, The Netherlands, Oct. 2016, pp. 21–37.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 25, Lake Tahoe, NV, USA, Dec. 2012, pp. 1097–1105.

[18] Raspberry Pi Ltd., "Raspberry Pi 5 product brief," Raspberry Pi Documentation, Cambridge, U.K., 2023. [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-5/

[19] Raspberry Pi Ltd., "Raspberry Pi Camera Module 3 datasheet," Raspberry Pi Documentation, Cambridge, U.K., 2023. [Online]. Available: https://www.raspberrypi.com/documentation/accessories/camera.html

[20] Intel Corporation, "Intel RealSense Depth Camera D435: Product datasheet and specifications," Santa Clara, CA, USA, 2022. [Online]. Available: https://www.intelrealsense.com/depth-camera-d435/

[21] OmniVision Technologies, Inc., "OV9782: 1-megapixel global shutter image sensor," Datasheet, 2021. [Online]. Available: https://www.ovt.com/sensors/OV9782

[22] OpenAI. (2025, July 4). *How to use the SIM7600G-H to send data over LTE to a server* [ChatGPT conversation]. ChatGPT. https://chatgpt.com/share/68685bd6-eb44-800c-9087-b6f8735e9b1d

[23] Shenzhen EAI Technology Co., Ltd., “YDLIDAR X4 Pro Datasheet,” Datasheet, 2022. [Online]. Available: https://static.generation-robots.com/media/YDLIDARX4PRODatasheet.pdf

[24] U.S. Food and Drug Administration, “Performance standards for light-emitting products; Part 1040.10 Laser products,” Title 21 Code of Federal Regulations, 2023. [Online]. Available: https://www.ecfr.gov/current/title-21/chapter-I/subchapter-J/part-1040#p-1040.10

[25] SQLite.org, “Write-Ahead Logging,” [Online]. Available: https://www.sqlite.org/wal.html. . [Accessed: Jul. 2025].

Appendix D - Code

Integration of HMI

```
#include <SPI.h>

#include <Adafruit_GFX.h>

#include <Adafruit_ST7789.h>



// TFT Pins

#define TFT_CS     5

#define TFT_DC     2

#define TFT_RST    4

#define TFT_BL     15  // optional, for backlight control



Adafruit_ST7789 tft = Adafruit_ST7789(TFT_CS, TFT_DC, TFT_RST);



// LED Pins

#define YELLOW_LED 13

#define GREEN_LED  12

#define RED_LED    14



// Buzzer Pin

#define BUZZER_PIN 27



// Track red alert state

bool redAlertActive = false;
```

```
void setup() {
 Serial.begin(115200);

 // LED & Buzzer setup
 pinMode(YELLOW_LED, OUTPUT);
 pinMode(GREEN_LED, OUTPUT);
 pinMode(RED_LED, OUTPUT);
 pinMode(BUZZER_PIN, OUTPUT);
 pinMode(TFT_BL, OUTPUT);
 digitalWrite(TFT_BL, HIGH); // Turn on backlight

 // TFT init
 tft.init(240, 320);  // ST7789 is 240x240
 tft.setRotation(2);  // adjust as needed
 showIdleScreen();
}

void loop() {
 if (Serial.available()) {
   char input = Serial.read();

   Serial.print("Received: ");
   Serial.println(input);

   switch (input) {
```

```
case '1':
  redAlertActive = true;
  digitalWrite(RED_LED, HIGH);
  showRedScreen();
  tone(BUZZER_PIN, 1000); // Alarm tone
  break;

case '0':
  if (redAlertActive) {
    redAlertActive = false;
    digitalWrite(RED_LED, LOW);
    noTone(BUZZER_PIN);
    showIdleScreen();
  }
  break;

case '2':
  if (!redAlertActive) {
    digitalWrite(YELLOW_LED, HIGH);
    showYellowScreen();
    delay(3000); // 3 seconds
    digitalWrite(YELLOW_LED, LOW);
    showIdleScreen();
  }
  break;
```

```
    case '3':
      if (!redAlertActive) {
        digitalWrite(GREEN_LED, HIGH);
        showGreenScreen();
        delay(3000); // 3 seconds
        digitalWrite(GREEN_LED, LOW);
        showIdleScreen();
      }
      break;

    default:
      if (!redAlertActive) {
        showIdleScreen();
      }
      break;
  }
 }
}

// === Display Functions ===

void showIdleScreen() {
 tft.fillScreen(ST77XX_WHITE);
 tft.setTextColor(ST77XX_BLACK);
```

```
 tft.setTextSize(3);
 tft.setCursor(50, 90);
 tft.print("R.O.A.M.");
 tft.setTextSize(2);
 tft.setCursor(80, 140);
 tft.print("Group 14");
}

void showYellowScreen() {
 tft.fillScreen(ST77XX_YELLOW);
 tft.setTextColor(ST77XX_BLACK);
 tft.setTextSize(3);
 tft.setCursor(60, 110);
 tft.print("TRASH");
}

void showGreenScreen() {
 tft.fillScreen(ST77XX_GREEN);
 tft.setTextColor(ST77XX_BLACK);
 tft.setTextSize(3);
 tft.setCursor(45, 110);
 tft.print("TREASURE");
}

void showRedScreen() {
```

```
tft.fillScreen(ST77XX_RED);

tft.setTextColor(ST77XX_WHITE);

tft.setTextSize(3);

tft.setCursor(45, 90);

tft.print("HELP!");

tft.setTextSize(2);

tft.setCursor(10, 150);

tft.print("Call XXX-XXX-XXXX");

}
```

GPS CODE

```
#define MODEM_RX 16  // Connect to SIM7600 TX
#define MODEM_TX 17  // Connect to SIM7600 RX

void sendCommand(const char* cmd, int delayMs = 1000) {
  Serial1.println(cmd);
  Serial.print(">> "); Serial.println(cmd);
  unsigned long start = millis();
  while (millis() - start < delayMs) {
    while (Serial1.available()) {
      Serial.write(Serial1.read());
    }
  }
}

float convertToDecimal(String nmea, String direction) {
  if (nmea.length() < 4) return 0.0;
  int dotIndex = nmea.indexOf('.');
  int degLength = dotIndex - 2;
  float deg = nmea.substring(0, degLength).toFloat();
  float min = nmea.substring(degLength).toFloat();
  float dec = deg + (min / 60.0);
  return (direction == "S" || direction == "W") ? -dec : dec;
}

void setup() {
  Serial.begin(115200);
  Serial1.begin(115200, SERIAL_8N1, MODEM_RX, MODEM_TX);
  delay(2000);

  sendCommand("AT");
  sendCommand("ATE0");
  //sendCommand("AT+CGPS=1");
  delay(2000);
}

void loop() {
  Serial1.println("AT+CGPSINFO");
  delay(2000);
  String response = "";
  while (Serial1.available()) {
    char c = Serial1.read();
    response += c;
  }

  Serial.println(response);

  int idx = response.indexOf("+CGPSINFO:");
  if (idx != -1) {
    String data = response.substring(idx + 10);
    data.trim();
    if (!data.startsWith(",")) {
      int parts[6];
      String tokens[8];
      int tokenCount = 0;
```

```
      int last = 0;
      for (int i = 0; i < data.length(); i++) {
        if (data[i] == ',') {
          tokens[tokenCount++] = data.substring(last, i);
          last = i + 1;
        }
        if (tokenCount >= 7) break;
      }
      tokens[tokenCount++] = data.substring(last);

      float lat = convertToDecimal(tokens[0], tokens[1]);
      float lon = convertToDecimal(tokens[2], tokens[3]);

      Serial.print("Latitude (decimal): "); Serial.println(lat, 6);
      Serial.print("Longitude (decimal): "); Serial.println(lon, 6);
    } else {
      Serial.println("Waiting for GPS fix...");
    }
  }

  delay(5000);
}
```

Appendix E - ChatGPT Prompt and Response

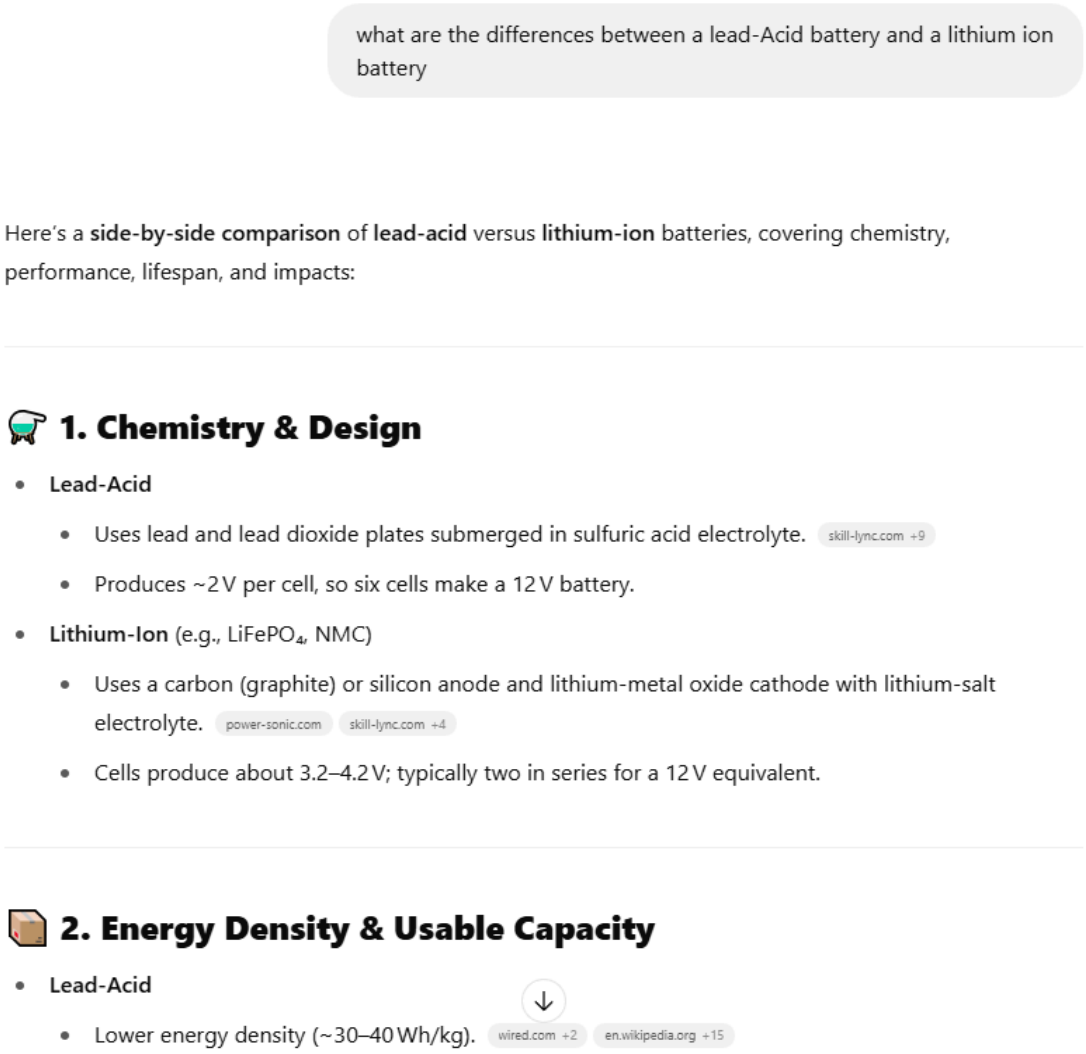


Figure E1 — Chapter 5.1 ChatGPT Q&A Part 1.1

- Generally use only ~50% of capacity to avoid damage, limiting real usable energy.

- **Lithium-Ion**
  - Much higher energy density (~150–275 Wh/kg). reddit.com +1 en.wikipedia.org +2
  - Can safely discharge to ~80–100%, giving a full usable capacity.

---

3. Efficiency & Discharge Behavior

- **Lead-Acid**
  - Typical round-trip efficiency is ~80–85%. energysage.com +15 wired.com
  - Capacity drops significantly under high discharge rates—a 0.8 C discharge might only yield 60% of rated capacity. power-sonic.com +2
- **Lithium-Ion**
  - High efficiency (~95%). energysage.com +10
  - Capacity remains consistent even at high discharge rates.

---

4. Cycle Life & Longevity

- **Lead-Acid**
  - ~300–500 full cycles. Deep discharge accelerates degradation. en.wikipedia.org +15
  - Lifespan roughly 3–12 years depending on use. energysage.com
- **Lithium-Ion**
  - ~2,000–7,000 cycles depending on chemistry (e.g., $LiFePO_4$ ~3,000–7,000). power-sonic.com

Figure E2 — Chapter 5.1 ChatGPT Q&A Part 1.2

💵 5. Cost & Total Value

- **Lead-Acid**
  - Much cheaper up-front (~$500–1,000 for small systems). energysage.com
  - Overall value suffers due to shorter life and less usable energy.
- **Lithium-Ion**
  - Higher upfront (~$5,000–15,000 installed). energysage.com +3
  - But lower $/cycle and better usable energy, making total cost of ownership often lower.

🔋 6. Weight & Size

- **Lead-Acid**
  - Heavy and bulky (energy density ~35–45 Wh/kg). en.wikipedia.org
- **Lithium-Ion**
  - Lighter and more compact (3–5× energy density a

W en.wikipedia.org

Lithium-ion battery

July 1, 2025 — A lithium-ion or Li-ion battery is a type of rechargeable battery that uses the reversible intercalatio...

🔧 7. Maintenance & Use

- **Lead-Acid**
  - Require periodic maintenance (water topping, equalizing charge).
  - Self-discharge up to 20%/month and risk sulfation when stored partially charged. reddit.com +7

✅ Bottom Line

- Choose **lead-acid** when upfront cost matters most, weight isn't an issue, and usage is infrequent.
- Choose **lithium-ion** when you need longer life, higher efficiency, more energy use, and compact size despite higher initial investment.

If you'd like, I can help you calculate cost over time, select specific battery models, or highlight how each chemistry would perform in your project context.

Figure E3 — Chapter 5.1 ChatGPT Q&A Part 1.3

What is the differences between a Lead-Acid Battery and a Lithium-Ion Battery?

Great question! Lead-acid and lithium-ion batteries are both rechargeable, but they differ significantly in performance, design, and applications. Here's a breakdown to help you compare them:

⚡ Key Differences Between Lead-Acid and Lithium-Ion Batteries

Feature	Lead-Acid Battery	Lithium-Ion Battery
Energy Density	Low (30–50 Wh/kg)	High (150–250+ Wh/kg)
Weight	Heavy	Lightweight
Lifespan	~300–500 cycles	2,000–10,000+ cycles
Charging Time	6–15 hours	1–3 hours
Depth of Discharge (DoD)	~50% recommended	Up to 85–100%
Efficiency	80–85%	95–98%
Maintenance	Requires regular checks (e.g., water levels)	Maintenance-free
Cost (Initial)	Lower upfront cost	Higher upfront cost
Cost (Lifetime)	Higher due to replacements	Lower due to longevity
Environmental Impact	Highly recyclable	Less recyclable, but improving

Figure E4 — Chapter 5.1 CoPilot Q&A Part 1.1

Safety	Stable but can leak acid	Sensitive to overheating, needs battery management system

🔋 Use Cases

- **Lead-Acid**: Best for backup power (UPS), automotive starters, and budget-sensitive applications.
- **Lithium-Ion**: Ideal for electric vehicles, portable electronics, solar energy storage, and high-performance systems.

🧠 Quick Analogy

Think of **lead-acid** like a sturdy old pickup truck—reliable, affordable, but not very efficient. **Lithium-ion** is more like a sleek electric car—faster, lighter, and more efficient, but with a higher price tag.

Want help choosing the right one for a specific project or device?

Figure E5 — Chapter 5.1 CoPilot Q&A Part 1.2

what is the difference between a 6V lithium ion battery and a 12V lithium ion battery

Here's a detailed comparison between **6 V** and **12 V lithium-ion batteries**, highlighting their differences and what they mean for your project:

🪫 1. **Voltage & Cell Configuration**

- **6 V battery** typically uses **2 lithium-ion cells in series**, each with a nominal voltage of ~3.2–3.7 V → total ~6.4–7.4 V nominal and ~8.4 V fully charged tycorun.com +14.
- **12 V battery** uses **4 cells in series**, totaling ~12.8–14.8 V nominal and up to ~16.8 V fully charged .

🔋 2. **Energy Storage & Capacity**

- Voltage alone doesn't define battery capability—**amp-hours (Ah)** also matter:
  - A **6 V 200 Ah** battery stores 1,200 Wh (6 V × 200 Ah).
  - A **12 V 100 Ah** battery also stores 1,200 Wh (12 V × 100 Ah).
- Even with identical Wh, packaging and internal resistance can differ reddit.com +2 redodopower.com.

Figure E6 — Chapter 5.1 ChatGPT Q&A Part 2.1

🧩 3. **Physical Size & Weight**

- **6 V batteries** are generally **more compact and lighter per Wh**, although older lead-acid 6 V ones were heavier due to thicker plates redodopower.com +1.
- **12 V lithium batteries** offer more energy in a single package and are easier to install without configuring cells manually litime.com +13.

🔧 4. **Ease of Use & Installation**

- A single **12 V battery** simplifies wiring and avoids series balancing issues.
- Using two **6 V batteries in series** works, but requires matching and balancing to ensure consistent performance .

💵 5. **Cost & Lifecycle**

- **12 V lithium packs** typically have higher upfront cost but:
  - Require less maintenance,
  - Are more available,
  - Benefit from scale economies ipowerqueen.com.
- Conversely, high-capacity **6 V packs** may cost more per unit but provide longer lifespans, especially in deep-cycle use .

🧪 6. **Use Cases & Suitability**

- **6 V** often serves in **deep-cycle applications** (like golf carts and RV setups) due to robust cell design and high Ah ratings forestriverforums.com +15.
- **12 V** packs are more versatile and ideal for:
  - Standard electronics,
  - Solar systems,
  - Automotive applications,
  - Anywhere 12 V is already the norm .

Figure E7 — Chapter 5.1 ChatGPT Q&A Part 2.2

what is the difference between a 6V lithium ion battery and a 12V lithium ion battery

Absolutely! Here's a clear side-by-side comparison to help you understand the differences between a **6V** and a **12V lithium-ion battery**:

🔋 6V vs 12V Lithium-Ion Battery Comparison

Feature	6V Lithium-Ion Battery	12V Lithium-Ion Battery
Voltage Output	6 volts	12 volts
Typical Use Cases	Toys, lighting, small electronics, compact solar setups	RVs, marine systems, power tools, solar storage, backup power
Power Capacity	Lower wattage output	Higher wattage output
Configuration	Often used in series to build higher voltages (e.g., 2 × 6V = 12V)	Standalone or used in parallel for more capacity
Size & Weight	Smaller and lighter	Larger and heavier
Charging Requirements	Needs a 6V charger	Needs a 12V charger
Flexibility	Good for modular setups	More plug-and-play for 12V systems
Cost	Generally cheaper per unit	Higher cost per unit, but fewer needed

Figure E8 — Chapter 5.1 CoPilot Q&A Part 2.1

what option would you suggestion btween lead acid and 6v lithium and 12v lithium

Here's a comprehensive breakdown and recommendation between your three battery options: **lead-acid**, **6 V lithium**, and **12 V lithium**, with guidance for your specific project needs:

⚙️ Option 1: Lead-Acid Battery

✅ **Pros:**

- Low upfront cost
- Reliable and widely available
- Good for high current (starting load) applications redodopower.com +8 redodopower.com +5 en.wikipedia.org +1

❌ **Cons:**

- Heavy, bulky (~35–40 Wh/kg) en.wikipedia.org +4
- Limited usable capacity (~50% DOD)
- Shorter cycle life (~300–500 cycles) en.wikipedia.org +14
- Maintenance needed (e.g., watering flooded types); emits gases
- Low efficiency (~80–85%) ecofishingshop.com

Figure E9 — 5.1 ChatGPT Q&A Part 3.1

⚡ Option 2: 6 V Lithium Battery (e.g., $LiFePO_4$)

✅ **Pros:**

- Higher energy density, longer cycle life (3k–10k cycles) en.wikipedia.org +15
- Deeper discharge capability (~80–100% usable)
- Lighter and compact
- Maintenance-free; no off-gassing reddit.com +15 rvlife.com +4 montanaowners.com

❌ **Cons:**

- More complex installation (often requires two 6 V in series for 12 V systems, plus cell balancing) endless-sphere.com
- Heavier than a single 12 V lithium due to two batteries instead of one mygrandrv.com +3
- Higher upfront cost
- Availability of 6 V lithium may be limited

🔋 Option 3: 12 V Lithium Battery ($LiFePO_4$)

✅ **Pros:**

- Plug-and-play with standard 12 V systems
- Lighter and smaller (~20–30 lb) redodopower.com +15 redodopower.com +3
- Long lifespan (3k–10k cycles, 10+ years)
- Very high efficiency (~95%) and fast charging (up to 4× faster than lead-acid) ecofishingshop.com +6

- Minimal maintenance; safe for indoor use (no venting gases) montanaowners.com +1

❌ **Cons:**

- Highest initial cost
- Requires proper BMS to protect cells
- Needs compatible charger (not typical lead-acid)

Figure E10 — Chapter 5.1 ChatGPT Q&A Part 3.2

what would you suggest between lead-acid and 6v lithium-ion and 12v lithium-ion

Great question—choosing between **lead-acid**, **6V lithium-ion**, and **12V lithium-ion** batteries depends on your specific needs, budget, and application. Here's a side-by-side comparison to help you decide:

🔋 Battery Type Comparison

Feature	Lead-Acid	6V Lithium-Ion	12V Lithium-Ion
Energy Density	Low	High	High
Weight	Heavy	Light	Light
Lifespan	300–500 cycles	2,000–5,000+ cycles	2,000–5,000+ cycles
Maintenance	Regular (e.g., water checks)	Maintenance-free	Maintenance-free
Charging Time	Slow (6–15 hrs)	Fast (1–3 hrs)	Fast (1–3 hrs)
Cost (Initial)	Low	Medium	High
Cost (Lifetime)	High (due to replacements)	Lower	Lower
Flexibility	Good for budget or legacy systems	Modular (can build custom voltages)	Plug-and-play for 12V systems

Figure E11 — Chapter 5.1 CoPilot Q&A Part 3.1

Use Cases	Backup power, vehicles, budget setups	Custom solar, compact devices	RVs, marine, solar, off-grid homes

🧠 Recommendation by Scenario

- **Budget-Conscious or Stationary Use**: Go with **lead-acid** if cost is your top concern and weight/space isn't an issue.
- **Modular or Custom Systems**: Choose **6V lithium-ion** if you want to build a flexible setup (e.g., 2 × 6V = 12V) with higher amp-hour capacity.
- **Simplified, High-Performance Systems**: Opt for **12V lithium-ion** if you want a ready-to-use, efficient, and long-lasting solution for RVs, solar, or mobile power.

🧩 Quick Analogy

Think of it like this:

- **Lead-acid** is the old-school pickup truck—rugged, cheap, but needs more care.
- **6V lithium-ion** is a box of Lego bricks—versatile and efficient if you like to build.
- **12V lithium-ion** is a Tesla—sleek, powerful, and ready to roll.

If you tell me what you're powering or what your priorities are (cost, weight, runtime, etc.), I can give you a more tailored suggestion!

Edit in a page

Figure E12 — Chapter 5.1 CoPilot Q&A Part 3.2

Provide a Flask POST route at /upload that accepts a multipart image (image), form fields label, latitude, longitude, uses parameterized sqlite3 inserts, and returns HTTP 400/500 on errors.

Here is the Flask POST route as per your request:

python Copy Edit

```python
from flask import Flask, request, jsonify
import sqlite3, os

app = Flask(__name__)

# Define the upload folder
UPLOAD_FOLDER = 'uploads'
os.makedirs(UPLOAD_FOLDER, exist_ok=True)

@app.route('/upload', methods=['POST'])
def upload():
    # Check if 'image' is in the request files
    if 'image' not in request.files:
        return jsonify(error='No image provided'), 400

    image = request.files['image']

    # Check if the image has a valid filename
    if image.filename == '':
        return jsonify(error='Empty filename'), 400

    # Retrieve form fields
    label = request.form.get('label')
    lat = request.form.get('latitude')
    lng = request.form.get('longitude')

    # Ensure all required form fields are provided
    if not all([label, lat, lng]):
        return jsonify(error='Missing metadata'), 400

    # Save the image to the upload folder
    filename = os.path.join(UPLOAD_FOLDER, image.filename)
    image.save(filename)

    try:
        # Connect to SQLite database
        conn = sqlite3.connect('data.db')
        cur = conn.cursor()

        # Parameterized query to insert data into the 'detections' table
        cur.execute(
            "INSERT INTO detections (label, latitude, longitude, filename) VALUES (?, ?, ?, ?)",
            (label, float(lat), float(lng), filename)
        )
```

Figure E13 — Chapter 5.3 ChatGPT Q&A

```
        # Connect to SQLite database
        conn = sqlite3.connect('data.db')
        cur = conn.cursor()

        # Parameterized query to insert data into the 'detections' table
        cur.execute(
            "INSERT INTO detections (label, latitude, longitude, filename) VALUES (?, ?, ?, ?)",
            (label, float(lat), float(lng), filename)
        )

        # Commit changes to the database
        conn.commit()

    except Exception as e:
        # Return an error message in case of database failure
        return jsonify(error='Database write failed', message=str(e)), 500

    finally:
        # Close the database connection
        conn.close()

    # Return a success message
    return jsonify(status='success'), 200

if __name__ == '__main__':
    app.run(debug=True)
```

Figure E14 — Chapter 5.3 ChatGPT Q&A

Question: How do I place custom trash and treasure icons on a Leaflet map and enable marker clustering for up to 300 points?

Here's the response following your specified format:

Q2: Prototyping Interactive Map Icons

Question: How do I place custom trash and treasure icons on a Leaflet map and enable marker clustering for up to 300 points?

Answer:

javascript Copy Edit

```javascript
// Define custom icons
const trashIcon = L.icon({
  iconUrl: '/icons/trash.png',
  iconSize: [32, 32],
  iconAnchor: [16, 32]
});
const treasureIcon = L.icon({
  iconUrl: '/icons/treasure.png',
  iconSize: [32, 32],
  iconAnchor: [16, 32]
});

// Create marker cluster group
const markers = L.markerClusterGroup({ maxClusterRadius: 60 });

// Add points
data.forEach(item => {
  const icon = item.type === 'treasure' ? treasureIcon : trashIcon;
  const marker = L.marker([item.lat, item.lng], { icon });
  markers.addLayer(marker);
});

// Add to map
map.addLayer(markers);
```

Figure E15 — Chapter 5.3 ChatGPT Q&A

Question: My fetch calls to /upload are failing with CORS errors. Explain preflight requests and two ways to fix this, then show the simplest Flask-CORS patch.

When a browser makes a cross-origin POST with custom headers or content types, it first sends an OPTIONS "preflight" request to check if the server allows the real request.

Two ways to address CORS:

1. **Manually set response headers** (Access-Control-Allow-Origin, Allow-Methods, etc.) in each route.
2. **Use the flask_cors extension** to automate header injection.

The simplest fix with Flask-CORS is:

python Copy Edit

```python
from flask import Flask
from flask_cors import CORS

app = Flask(__name__)
CORS(app)  # Allow all origins by default

@app.route('/upload', methods=['POST'])
def upload():
    # your upload logic here
    ...
```

Figure E16 — Chapter 5.3 ChatGPT Q&A

Question: Write pure CSS Grid rules to split a container into 75% map and 25% sidebar with a 16px gap, and stack to one column below 600px.

css

Copy Edit

```css
.container {
  display: grid;
  grid-template-columns: 3fr 1fr;
  gap: 16px;
}

@media (max-width: 600px) {
  .container {
    grid-template-columns: 1fr;
  }
}

#map { height: 100vh; width: 100%; }
#sidebar { background: #fafafa; padding: 1rem; }
```

Figure E17 — Chapter 5.3 ChatGPT Q&A

Question: What best practices should we follow to ensure academic integrity when using ChatGPT's code suggestions?

- **Always review and test** every suggestion under real conditions.
- **Annotate AI-generated snippets** with comments like `# Generated by ChatGPT`.
- **Verify** that no deprecated APIs or insecure patterns slipped in.
- **Credit ChatGPT** in your documentation or footnotes.
- **Use AI output as a starting point**, not a final submission—modify and learn from it.
- **Cross-check complex logic** with official docs or peer review.

Figure E18 — Chapter 5.3 Copilot Q&A

Question: Provide a Flask POST route at /upload that accepts a multipart image (image), form fields label, latitude, longitude, uses parameterized sqlite3 inserts, and returns HTTP 400/500 on errors.

Question: Provide a Flask POST route at `/upload` that accepts a multipart image (`image`), form fields `label`, `latitude`, `longitude`, uses parameterized sqlite3 inserts, and returns HTTP 400/500 on errors. Answer:

Python Copy

```python
from flask import Flask, request, jsonify
import sqlite3, os

app = Flask(__name__)
app.config['MAX_CONTENT_LENGTH'] = 5 * 1024 * 1024  # 5 MB limit

DB_PATH    = 'data.db'
UPLOAD_DIR = 'uploads'
os.makedirs(UPLOAD_DIR, exist_ok=True)

def init_db():
    conn = sqlite3.connect(DB_PATH)
    conn.execute('''
      CREATE TABLE IF NOT EXISTS detections (
        id INTEGER PRIMARY KEY,
        label TEXT NOT NULL,
        latitude REAL NOT NULL,
        longitude REAL NOT NULL,
        filename TEXT NOT NULL
      );
    ''')
    conn.commit()
    conn.close()

init_db()

@app.route('/upload', methods=['POST'])
def upload():
    # file presence and type checks
    if 'image' not in request.files:
```

```python
def upload():
    # file presence and type checks
    if 'image' not in request.files:
        return jsonify(error='Image missing'), 400
    img = request.files['image']
    if img.mimetype not in ('image/jpeg', 'image/png'):
        return jsonify(error='Invalid file type'), 400

    # form fields validation
    label = request.form.get('label')
    lat   = request.form.get('latitude')
    lng   = request.form.get('longitude')
    if not all((label, lat, lng)):
        return jsonify(error='Missing metadata'), 400

    # save file
    filename = os.path.join(UPLOAD_DIR, img.filename)
    try:
        img.save(filename)
    except Exception:
        return jsonify(error='File save error'), 500

    # insert into SQLite safely
    try:
        conn = sqlite3.connect(DB_PATH)
        cur  = conn.cursor()
        cur.execute(
            'INSERT INTO detections (label, latitude, longitude, filename) \
            (label, float(lat), float(lng), filename)
        )
        conn.commit()
    except Exception:
        return jsonify(error='Database error'), 500
    finally:
        conn.close()

    return jsonify(status='success'), 200
```

Figure E19 — Chapter 5.3 Copilot Q&A

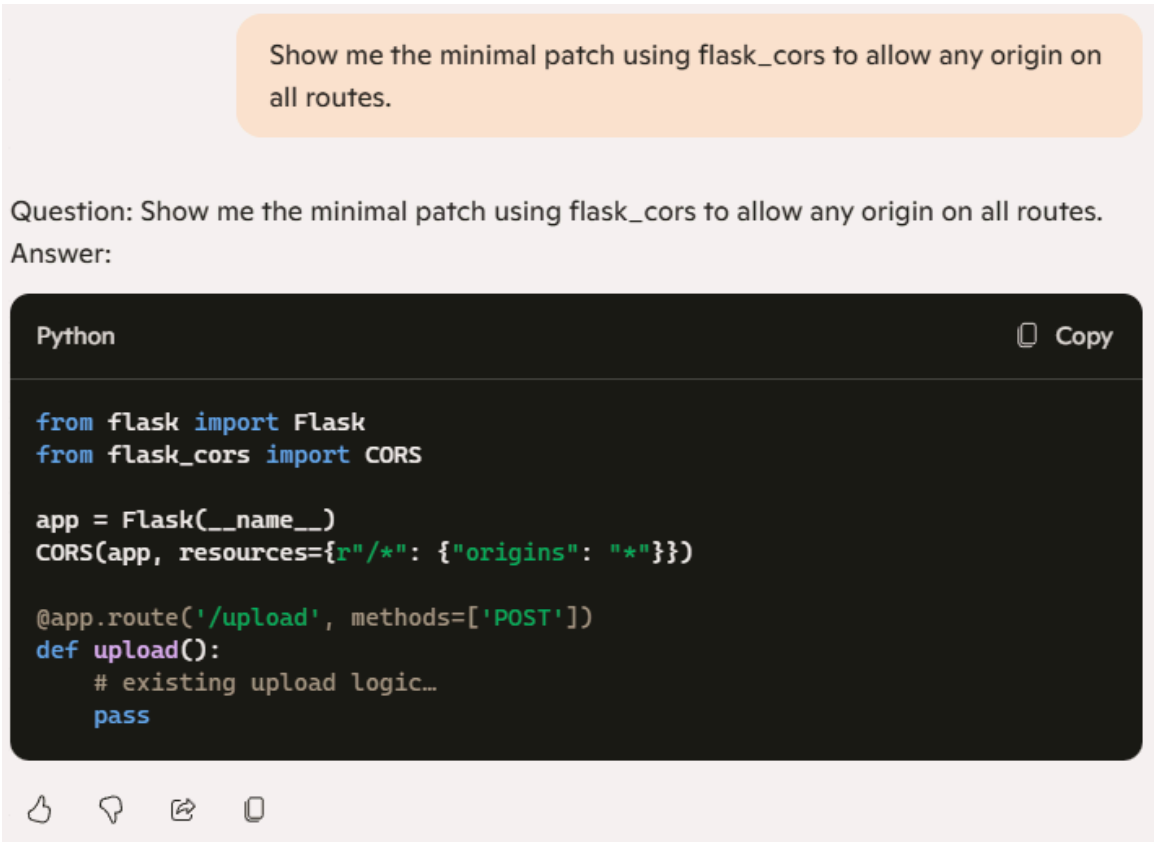


Figure E20 — Chapter 5.3 Copilot Q&A

Generate pure CSS Grid rules to split a container into a 75% map panel and 25% sidebar with a 16 px gap, and stack into one column below 600 px.

Question: Generate pure CSS Grid rules to split a container into a 75% map panel and 25% sidebar with a 16 px gap, and stack into one column below 600 px. Answer:

Css Copy

```css
.container {
  display: grid;
  grid-template-columns: 3fr 1fr;
  gap: 16px;
}

@media (max-width: 600px) {
  .container {
    grid-template-columns: 1fr;
  }
}

#map {
  width: 100%;
  height: 100vh;
}

#sidebar {
  padding: 1rem;
  background: #f8f8f8;
}
```

Figure E21 — Chapter 5.3 Copilot Q&A

Figure E22 — Chapter 5.3 Copilot Q&A

Q5: Write a reusable JavaScript function `createMarker(lat, lng, type)` for Leaflet that uses custom PNG icons for `"trash"` and `"treasure"`, sets `iconAnchor`, wraps in a module, and plugs into a marker cluster. Answer:

Javascript Copy

```javascript
// icons.js
export function createMarker(lat, lng, type) {
  const icons = {
    trash: L.icon({
      iconUrl: '/img/trash.png',
      iconSize: [32, 32],
      iconAnchor: [16, 32]
    }),
    treasure: L.icon({
      iconUrl: '/img/treasure.png',
      iconSize: [32, 32],
      iconAnchor: [16, 32]
    })
  };
  return L.marker([lat, lng], { icon: icons[type] || icons.trash });
}

// main.js
import { createMarker } from './icons.js';

const clusterGroup = L.markerClusterGroup({ maxClusterRadius: 50 });
dataPoints.forEach(pt => {
  const marker = createMarker(pt.lat, pt.lng, pt.type);
  clusterGroup.addLayer(marker);
});
map.addLayer(clusterGroup);
```

E23 Chapter 5.2 ChatGPT

Write arduino code that tests to see if the SIM7600G-H has the ability to send data

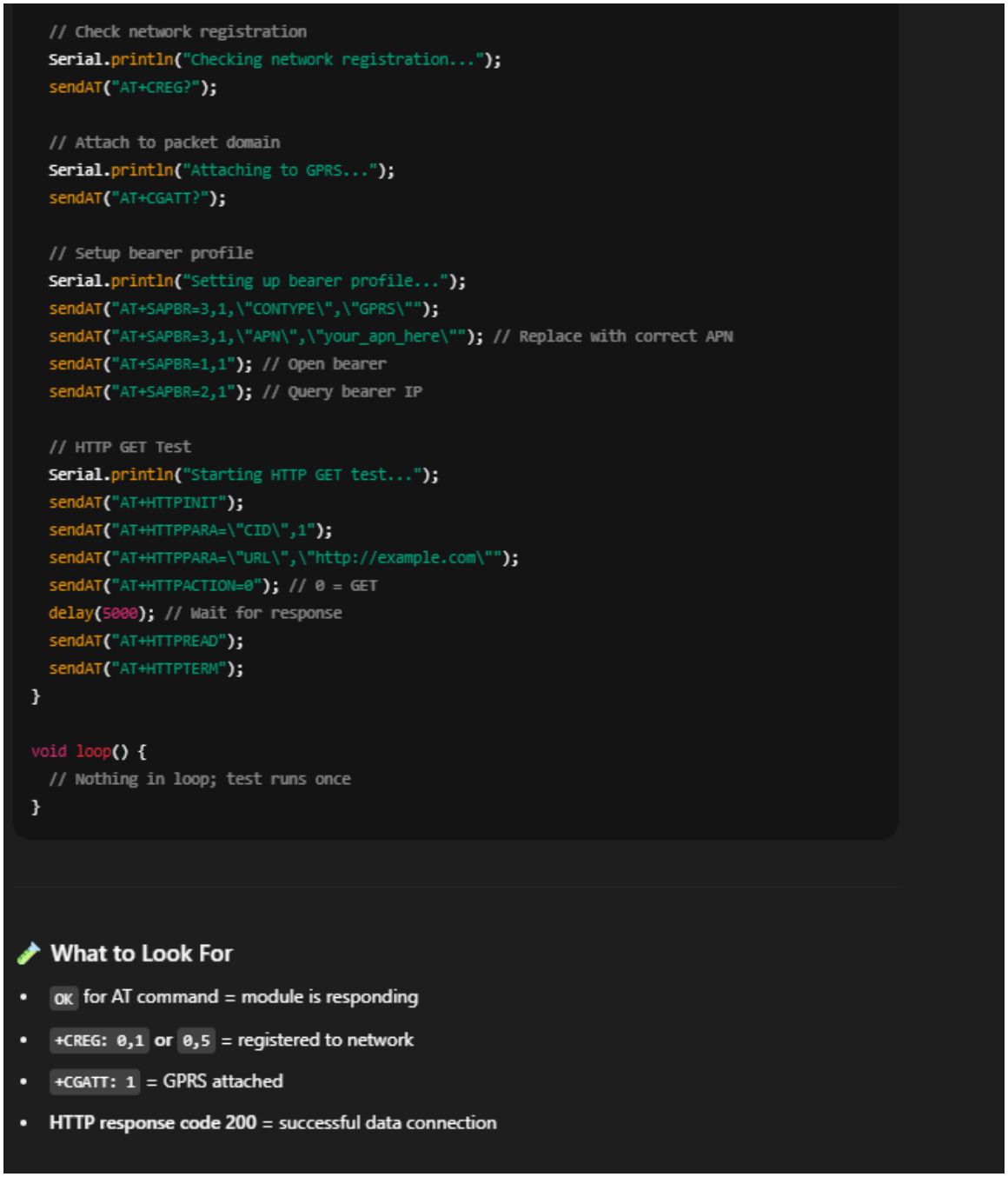
Here is an **Arduino sketch** that **tests whether your SIM7600G-H module is ready to send data over LTE** by performing these actions:

1. Initializes serial connections
2. Sends AT commands to check module presence
3. Checks network registration (`AT+CREG?`)
4. Attaches to GPRS (`AT+CGATT?`)
5. Opens a bearer (`AT+SAPBR`)
6. Performs a basic HTTP GET to test data connectivity

✅ Arduino Code: SIM7600G-H LTE Data Test

cpp Copy Edit

```cpp
#define TINY_GSM_MODEM_SIM7600
#include <SoftwareSerial.h>

// Use hardware Serial2 if available, otherwise use SoftwareSerial
// Make sure your TX/RX pins match your wiring
#define MODEM_RX 16
#define MODEM_TX 17
SoftwareSerial simSerial(MODEM_RX, MODEM_TX); // RX, TX

void sendAT(String cmd, int timeout = 2000) {
  simSerial.println(cmd);
  long int time = millis();
  while ((time + timeout) > millis()) {
    while (simSerial.available()) {
      String response = simSerial.readStringUntil('\n');
      response.trim();
      if (response.length() > 0) {
        Serial.println("> " + response);
      }
    }
  }
}

void setup() {
  Serial.begin(115200);
  simSerial.begin(115200);  // SIM7600 default baud

  Serial.println("Initializing SIM7600G-H Test...");

  delay(1000);

  // Basic communication test
  Serial.println("Sending AT...");
  sendAT("AT");
```



```
  // Check network registration
  Serial.println("Checking network registration...");
  sendAT("AT+CREG?");

  // Attach to packet domain
  Serial.println("Attaching to GPRS...");
  sendAT("AT+CGATT?");

  // Setup bearer profile
  Serial.println("Setting up bearer profile...");
  sendAT("AT+SAPBR=3,1,\"CONTYPE\",\"GPRS\"");
  sendAT("AT+SAPBR=3,1,\"APN\",\"your_apn_here\""); // Replace with correct APN
  sendAT("AT+SAPBR=1,1"); // Open bearer
  sendAT("AT+SAPBR=2,1"); // Query bearer IP

  // HTTP GET Test
  Serial.println("Starting HTTP GET test...");
  sendAT("AT+HTTPINIT");
  sendAT("AT+HTTPPARA=\"CID\",1");
  sendAT("AT+HTTPPARA=\"URL\",\"http://example.com\"");
  sendAT("AT+HTTPACTION=0"); // 0 = GET
  delay(5000); // Wait for response
  sendAT("AT+HTTPREAD");
  sendAT("AT+HTTPTERM");
}

void loop() {
  // Nothing in loop; test runs once
}
```

What to Look For

- `OK` for AT command = module is responding
- `+CREG: 0,1` or `0,5` = registered to network
- `+CGATT: 1` = GPRS attached
- **HTTP response code 200** = successful data connection

E24 Chapter 5.2 Copilot

Write arduino code that tests to see if the SIM7600G-H has the ability to send data

To test whether the SIM7600G-H module can send data using an Arduino, you'll want to run a simple sketch that:

1. Initializes communication between the Arduino and the SIM7600G-H.
2. Sends AT commands to check network registration and GPRS attachment.
3. Performs a basic HTTP request (e.g., GET or POST) to verify data transmission.

Here's a straightforward example using SoftwareSerial on an Arduino Uno:

Cpp Copy

```cpp
#include <SoftwareSerial.h>

SoftwareSerial sim7600(7, 8); // RX, TX

void setup() {
  Serial.begin(9600);          // Serial monitor
  sim7600.begin(115200);       // SIM7600 default baud rate

  Serial.println("Starting SIM7600G-H test...");

  sendAT("AT");                // Basic attention
  sendAT("AT+CPIN?");          // Check SIM status
  sendAT("AT+CREG?");          // Check network registration
  sendAT("AT+CGATT?");         // Check GPRS attachment
  sendAT("AT+CSQ");            // Check signal quality

  // Optional: Try sending a basic HTTP GET request
  // Make sure the PDP context is set first
  sendAT("AT+CGDCONT=1,\"IP\",\"your_apn_here\"");
  sendAT("AT+HTTPINIT");
  sendAT("AT+HTTPPARA=\"CID\",1");
  sendAT("AT+HTTPPARA=\"URL\",\"http://example.com\"");
  sendAT("AT+HTTPACTION=0");   // 0=GET
}

void loop() {
  if (sim7600.available()) {
    Serial.write(sim7600.read());
  }
}

void sendAT(String command) {
  sim7600.println(command);
  delay(1000);  // Give time for response
}
```

E25 Chapter 5.2 ChatGPT

"What is the SIM7600G-H and what can it do?"

The **SIM7600G-H** is a compact, all-in-one **LTE Cat 4** cellular modem from SIMCom. Think of it as a "cell-phone engine" you can bolt onto a microcontroller or single-board computer to give your project high-speed data, location, and even voice/SMS capability.

Key capabilities

Area	What it offers
Cellular data	LTE Cat 4 (150 Mb down / 50 Mb up) with automatic fallback to 3G (HSPA+) and 2G (EDGE/GPRS) for global roaming.
Positioning	On-board multi-constellation GNSS (GPS, GLONASS, Galileo, BeiDou) with raw NMEA output—handy for asset tracking.
Protocols	AT command set for TCP/UDP sockets, HTTP/HTTPS, FTP, MQTT, SSL/TLS, SMTP, and simple ping—no external TCP/IP stack required.
Voice & SMS	Circuit-switched voice (PCM/analog), VoLTE on some carriers, plus full SMS and USSD support.
I/O interfaces	Four UARTs (one high-speed, three 115 kBd), USB 2.0 device, I^2C, SPI, GPIOs, ADC, PWM, SD card, audio I^2S/PCM lines.
Power	Runs from a 3.4–4.3 V rail; bursts up to 2 A during LTE TX, so a stout supply or Li-ion cell is essential.
Form factor	30 × 30 mm LCC package shared with other SIM7-series modules—drop-in upgrade path.

Typical things you can do with it

1. **Remote telemetry:** Push sensor data from an ESP32 or Arduino over LTE to cloud dashboards or private servers.
2. **Edge trackers:** Combine GNSS fixes with cellular back-haul for vehicle, drone, or asset tracking.
3. **IoT gateways:** Use MQTT/SSL functions to connect low-power field devices to AWS IoT, Azure, etc., without writing a full TCP/IP stack.
4. **SMS control:** Send/receive text commands for out-of-band control or fail-safe messaging.
5. **Voice calling:** Add speaker/microphone and place traditional or VoLTE calls for intercom or alarm panels.

E26 Chapter 5.2 ChatGPT

Appendix F - Other

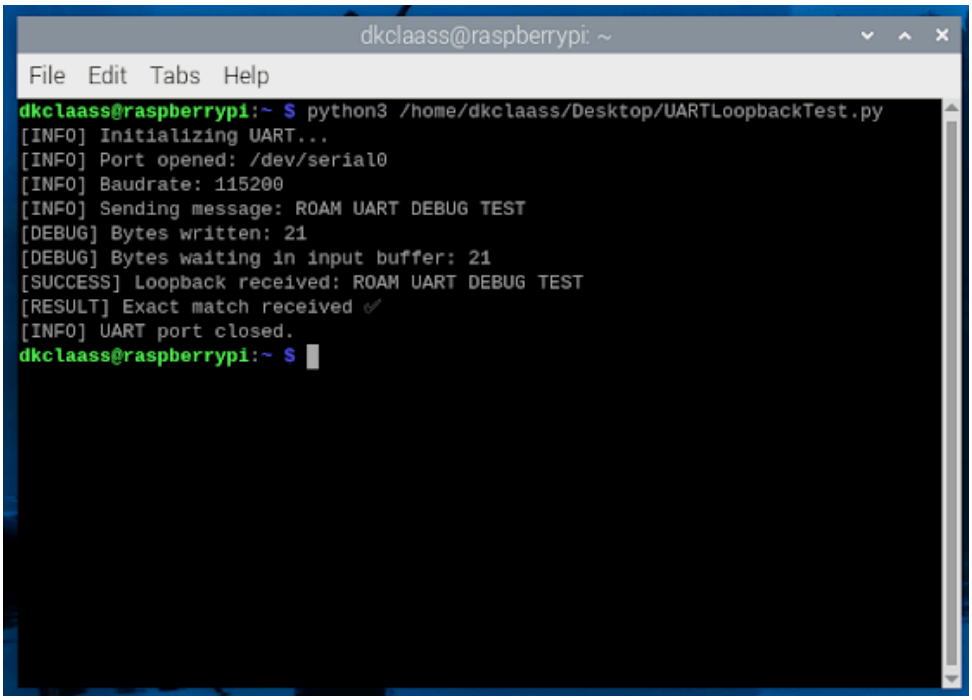


Figure F1 — Chapter 9.1.3.2 Raspberry Pi UART Loopback Test results