R.O.A.M. - Robotic Observer of Abandoned Materials

Ethan Robotham, Claire Persinger, Daniel Claassen, Nathan Little

Dept. of Electrical Engineering and Computer Science, University of Central Florida, Orlando, Florida, 32816-2450

***Abstract*— R.O.A.M. The robotic observer of abandoned materials is an autonomous ground robot designed to detect, classify and document waste and discarded objects along urban sidewalks. Using an onboard Raspberry Pi 5 and YOLOv11-based computer vision, R.O.A.M. identifies and differentiates general trash from reclaimable items such as furniture. The system pairs the PI with an ESP32 microcontroller, which manages motor control for real-time navigation. A 2D LiDAR sensor provides environmental awareness, allowing R.O.A.M. to follow sidewalks and detect pedestrians or obstacles. Detected items are captured, annotated, and stored with corresponding GPS coordinates and uploaded to our web platform.**

***Index Terms* — Autonomous robots, IoT systems, Flask server, SQLite, SIM7600G-H, AT commands, web telemetry.**

I. Introduction

Urban sidewalks accumulate a mix of ordinary trash and reusable items that are often missed by manual cleanup or community reuse efforts. To make this process more scalable, we present R.O.A.M. (Robotic Observer of Abandoned Materials), an autonomous ground robot that detects, labels, and geotags discarded objects in real time while navigating along sidewalks. The goal is simple: identify what's on the ground, decide whether it's trash or potentially reclaimable "treasure," and publish its approximate location to a public-facing website so it can be acted on. R.O.A.M. combines on-board AI and LiDAR with a lightweight microcontroller integration layer. A Raspberry Pi 5 runs the perception and autonomy stack, including YOLOv11-based object detection and a 2D LiDAR pipeline that keeps the robot centered on the sidewalk and aware of obstacles. An ESP32-S3 acts as the system's integrator and real-time controller. The Pi sends compact JSON messages over UART, either a signed steering value s in $[-256,255]$, or a classification label C with 'T' for treasure and 'G' for garbage. The ESP32 scales steering magnitudes to 8-bit PWM (0–255) for the motor drivers, updates on-board indicators (LCD text and LEDs), and, on classification events, queries a SIM7600G-H module for GPS coordinates via AT commands and forwards the labeled location to the project website as a map pin. This split design keeps compute-heavy tasks on the Raspberry Pi and assigns time-critical I/O and telemetry to the ESP32, which makes the system easier to reason about and debug. The message protocol is intentionally minimal so that decisions made by the autonomy stack translate directly into motion and reporting with low latency. To avoid redundant posts, object events are consolidated on the Pi side before they are forwarded, and only confirmed classifications are sent downstream. In practice, this yields a clear end-to-end flow: perception produces a label, the microcontroller attaches GPS, and the website reflects the result almost immediately. This paper provides an overview of the full system, so readers can understand the system as a whole. We summarize the overall system design and function, then outline the roles of the main components, including the Raspberry Pi 5, ESP32-S3, SIM7600G-H, 2D LiDAR, motor drivers, and user indicators. We also describe the web pipeline that displays geotagged events and the microcontroller integration that turns messages into motion and map pins. For this application, our focus is the architecture and integration choices that make R.O.A.M. practical for GPS-inferred waste recovery on sidewalks.

II. System Design

A. *Components*

Table 1. Power Distribution Needs

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
RASPBERRY 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

The functionality of R.O.A.M. relies on the chosen components of this project to be suitable for the task at

hand and properly powered. The chart above shows the needed distribution of the project and how each of our components generally are powered. Because some of these components had ranges of acceptable voltages slightly outside the ones listed, we were able to overlap some of the designs for regulators so that we could simplify the systems. With those power needs, we also needed to take into consideration what types of powerful components we could use that exist in those power needs.

We spent significant time discussing the microcontroller that we would choose to be the main integrator for this system. This device had to be able to be created on a PCB, support multiple peripheral interfaces, and be reliable. We went with the ESP32-S3 as it provides built-in wifi, dual-core processing, sufficient GPIO pins, and because of its small footprint and simplistic design we were able to have PCB integration. The ESP32's functionality will be to send information to our website, connect with our SIM7600 module to track our GPS location, obtain information from the Raspberry Pi about object classification and LiDAR direction, and allow for peripherals such as an LCD screen, LED lights, and a buzzer. All of those systems will intersect on that main MCU PCB board.

Beyond the ESP32, we have other certain components that will be responsible for collecting data that will be used by our main MCU. One of these items is the Raspberry Pi 5. The Raspberry Pi that we are using has two main purposes, one being to process the information coming from the LiDAR, and to run the machine learning for our object detection. These two systems are very demanding, this is why we went with the Pi that has 16GB. Previously there was the idea of using a Jetson or even two Raspberry Pis, but in the end getting one with enough power would streamline our system without losing any efficiency. The goal is to have the LiDAR sensor interpret the environment and send data to the Pi, this is where there will be some calculation of what to send to the ESP32. Once that is received, the ESP32 will send information to our motor drivers to adjust the course of our robot. The motor drivers are designed to replicate a H-bridge motor driver using the BTS7960 chip.

Another system that will be used by the same Pi is our object classification system. This will take in information from our Pi camera, interpret it on the Pi, and then send information to the ESP32 about what kind of object is perceived. Then it will be in the categories of trash, and treasure. From there, the ESP32 will send that information to the website, as well as display it via the LCD screen and LEDs.

The last main peripheral we have is our SIM7600G-H module, which is responsible for sending information about our coordinates to the ESP32 and hence our website. All of these interconnected parts will be powered by a Lithium-ion battery with a regulator to bring the current and voltage to what is safe for those components. We have a 5.2V regulator which will power the Raspberry Pi 5, and a 3.4V regulator which will power the ESP32 as well as the Sim7600 module. That power supply will also send power to our motors and motor drivers. The power supply being safe and accurate is critical to this design, without it we will not be able to meet the objectives we have set out for ourselves.

B. Printed Circuit Board

As previously stated, the outcome of our project was dependent on functional PCB boards that not only allow for a certain power output to allow our systems to run smoothly, but they also depend on functional systems that use the PCB connections to send data. One of the PCB designs we created was the power distribution chips, these being our regulators which were previously mentioned. This design uses two different types of regulator chips, the LM2576, and LM2679. For the lower voltage we only needed to see 2A which is why we were able to choose the less robust 2576. But for the Pi, it is a power hungry system that requires a greater amount of allotted amps, being 3A, which is why the LM2679 was more appropriate. These both take in power from a 12V Lithium-ion battery and steps it down those respective amounts so that our components can be properly powered. The ESP32 for example is a sensitive MCU as it can be fried by anything more than 3.5V. Due to this, we made sure to test our regulators on a load to confirm that it would suitably power our components and not allow for more than what it was designed for.

Another system we had built on the PCB board was our motor drivers. This design was modeled after the BTS79600 motor driver, which allows for a current of up to 43A. This was important since our motors are pulling a load, it needs to be able to withstand any issues that can come from our motors such as back EMF and stall current. We in this case need to not only protect our motor drivers, but our motors. This design does that. In our iteration of this design, we chose to use only the forward functionality of the design, meaning that we did not have to use as many GPIO pins on our ESP32. Since the ESP32 has limited usable inputs, we wanted to make sure we did not unnecessarily take up any pins. In this case we just had our enables tied high so they were just always on. This design was tested using a 100 ohm resistor as our motor so that we could confirm any issues before connecting them to our motors and accidentally blowing them. After careful testing and testing our system with the comparable development board, we were successful in its integration.

The last PCB that we designed involved all of our systems being integrated. This was our main MCU board which houses the main ESP32, and all adjacent connections. This board was responsible for taking in the main 12V batteries voltage, and dispersing them to the necessary regulators, as well as taking some of that back in and sending it to the ESP32. This board also has the connections to program the UART to USB converter. In this case, when we want to program the

ESP32, we keep the UART to USB connected to it. Once that is programmed it stays flashed, then we connect the Rx and Tx pins to those on the Pi, this allows a communication between the two. There is also another set of pins programmed for Rx and Tx information, which will be connected to the SIM7600. Additionally, this PCB has all the peripherals attached, the LEDs, the LCD screen, the buzzer, and the logic for the motor drivers come in here. This board is the hub for as far as connections, and it is imperative that we are able to functionally use it and communicate with all necessary components.

With all the testing done on these boards, we have four functional PCB boards to run our project off of that are able to properly disperse information and voltage to the rest of our systems. Having this functional brings us one step closer to meeting our goals and objectives.

III. Object Classification

A. *Purpose*

The object classification subsystem in R.O.A.M. is made to allow the identification, categorization, and location of materials to be logged when encountered. This is used to fulfill R.O.A.M's main goal of promoting efficient cleanup and recovery of usable items in urban areas.

B. *Concept*

The object classification system starts with the Raspberry Pi camera module 3 connected to the Raspberry Pi 5. The camera module is constantly taking a video, which is then delivered to the Raspberry Pi. On the Pi, each frame of the video is processed using YOLOv11, a deep-learning model that is trained to detect furniture items and garbage discarded on urban sidewalks. YOLOv11 detects and labels the frames with the object or objects detected and outputs the classification of the frame and confidence rating. These are sent in a JSON Package to the ESP3 over UART2.

To avoid congestion between the Raspberry Pi and ESP32, we need to track objects across multiple frames. This prevents the same object from being sent every time it's detected within a frame because if we sent a JSON package each time an object was detected on any given frame the system would be overloaded then stall and crash. To remedy this, we pass each frame to a SORT algorithm. The SORT algorithm assigns a unique ID to each object and tracks it across frames. When a new object is confirmed, the system formats the relevant information into a JSON package containing the classification. This package is then transmitted through UART to the ESP32.

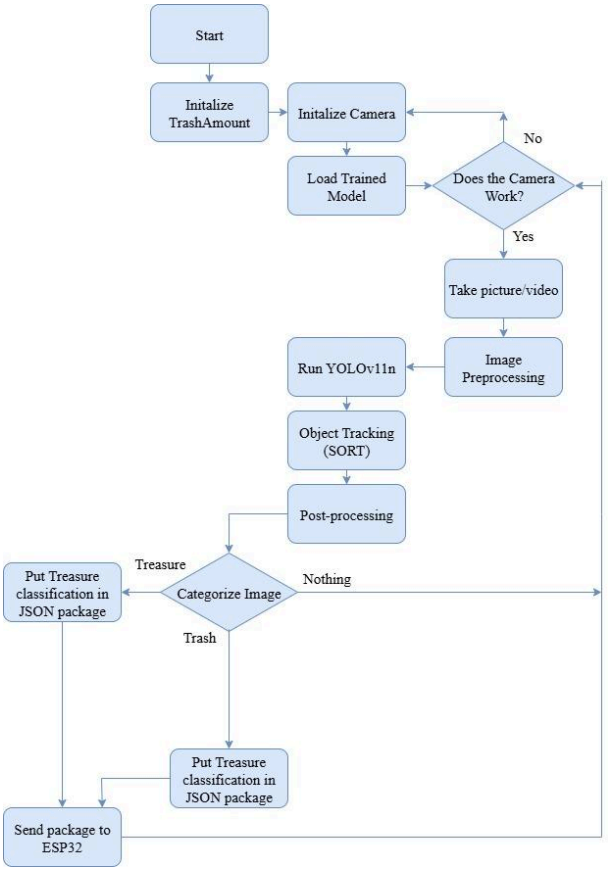


Fig 1. Flowchart of the Object Classification Pipeline.

C. *Components*

The Raspberry Pi Camera Module 3 was selected as our main camera because of its MP and HDR support, which many other options lacked. Its MP is about 12, This is important because the dataset used for object classification is collected using a 10 MP camera. It's important to keep the data the model is trained on and the data we are inputting into the system as consistent as possible. HDR support is required for R.O.A.M. because its environment is mainly outdoor, involving a wide range of bright and dark lighting that can change quickly. HDR helps balance these lighting and allows R.O.A.M. to see in a larger variety of environments.

The Raspberry Pi 5 is the main processing unit for the complicated analytics R.O.A.M. requires. The Pi handles computations regarding YOLOv11, SORT, and LiDAR. The 16 gigabyte RAM version was selected to provide enough resources for all computational requirements. The Raspberry Pi has to be loaded with libraries to function, including picamera2 and OpenCV.

YOLOv11 is a convolutional neural network used for object detection. Processing each frame from the camera and detecting objects. YOLOv11's single-pass detection method is particularly effective for real-time applications, as it can process each frame quickly enough for the robot's movement. It is the latest generation of the YOLO family and has impressive statistics when compared to previous generations

helping in accuracy and consistency. YOLOv11 is trained on a dataset of 474 images that are manipulated and rotated to add variety to the dataset.

The SORT algorithm was implemented to maintain consistent tracking of detected objects across multiple frames. It was chosen for its efficiency and low computational cost, which makes it ideal for the Raspberry Pi's processing environment. SORT assigns unique IDs to each object detected by YOLOv11 and tracks their movement, preventing repeated detections of the same item as R.O.A.M. moves past. This helps the system send clean and organized data without being redundant. Its lightweight design complements YOLOv11 by adding spatial consistency to the classification results while keeping processing demands low.

D. Testing

Testing for the object classification subsystem focused on confirming that the camera, detection model, tracking system, and data transmission all worked together accurately and in real time. Initial tests were performed indoors using printed and physical examples of trash and furniture to verify that the YOLOv11 model correctly identified each class. The purpose of these tests are to confirm R.O.A.M. 's ability to accurately detect objects and detect objects from a distance of 0.5m to 3m These tests allowed adjustments to the confidence threshold until detections consistently matched the truth. A confidence level of 0.65 was decided on from these tests.

Table 2. Object Classification Detection Distance Trial

	Couch	Fridge	Table	Chair	Couch	Shoe Rack	Mini fridge	Desk	Cabinet	Chair
0.5m	Y	Y	Y	Y	X	Y	Y	X	Y	Y
1.5m	Y	Y	X	Y	Y	Y	Y	X	Y	Y
3m	X	Y	x	Y	Y	X	X	X	Y	X
Score	2/3	3/3	1/3	3/3	2/3	2/3	2/3	0/3	3/3	2/3

This table shows the results of testing R.O.A.M.'s object classification distance trial. This trial is meant to test R.O.A.M's ability to detect objects within a specific range and reveal any weaknesses in the object detection dataset that need to be revised. R.O.A.M. was capable of detecting most objects at atleast one of the 3 ranges which is the desired result but shows certain weaknesses within groups of similar looking items like tables and desks. The test had an overall accuracy of 66% which does not meet the target accuracy of 70%.

Table 3.Object Classification Detection Accuracy Trial

	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Trial 6	Trial 7	Trial 8	Trial 9	Trial 10
Objects seen	Empty Can	Candy Bag	Couch	Cabinet	Fridge	Chair	Candy Bag	Misc Garbage	Shoe Rack	Table
Accuracy	Y	X	Y	Y	Y	Y	X	X	Y	Y

This test was aimed at evaluating R.O.A.M. 's ability to detect the objects it's meant to detect at any range. The goal for this test is 100% accuracy as it just tests whether it is possible to detect certain items. The failure of this test could reveal weaknesses in the dataset, however there are other factors to consider. The model is trained on objects that are nested in grass so the indoor environment used for testing may influence the results of garbage detection and not furniture items.

IV. Autonomous Driving

A. Purpose

Autonomous navigation of the sidewalk is the core navigation method of the robotic system used here. The goal of the subsystem is to enable self-navigation on a sidewalk and stop or avoid obstacles to create a safe environment on the sidewalk without human supervision. The formula to accomplish this goal is using control algorithms to perceive surroundings, plan paths, and make real-time decisions.

B. Concept

The autonomous driving subsystem operates based on sensor fusion and feedback control. Utilizing a 2D LiDAR sensor provides continuous low-power mapping of the environment by scanning the surroundings to detect the geometry of the path. A forward-tilted 2D LiDAR sensor captures data in a 360° sweep, producing point-cloud frames of approximately 1000 points per rotation. During testing it was found that only about 500-700 of these points were valid due to filtering of excessively close return distances. Furthermore, the front field of view (FOV) of ±45° from the forward-center axis of 0° restricts points further to about 25% of what was originally available.

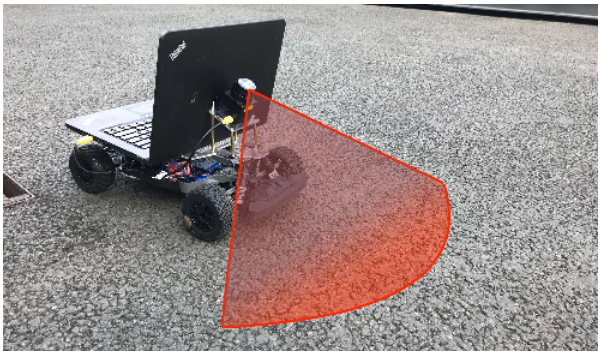


Fig 2. Example FOV of Tilted LiDAR Sensor from a previous autonomous rover project.[1]

As seen in the red region of Fig. 1 where the laser meets the ground is where the distance is measured when the sensor is tilted, this also shows the need for reducing the FOV. For surface identification, assuming a flat sidewalk and sloped or jagged boundaries, the LiDAR's downward pitch allows the system to infer elevation discrepancies between the smooth and jagged regions in view. This change in distance from the center sidewalk elevation is used as the basis of determining the boundaries of the navigable space.

The control loop divides the FOV into left and right halves of ±45° with distances calculated using

$$L_{avg} = (1 / N_L) \sum_{i=45}^{0} d_i \quad (1)$$

$$R_{avg} = (1/N_R) \sum_{i=0}^{45} d_i . \quad (2)$$

These average distances are used in finding the lateral error to find the distance from the center. The lateral error is found by taking the difference of right and left average distances

$$\Delta = R_{avg} - L_{avg}. \quad (3)$$

Then the steering command (u) is determined differentially with

$$u = k_p \cdot \Delta + \alpha \cdot u_{prev}, \quad (4)$$

and the value of u will be positive or negative to dictate a left or right turn respectively. The smoothing factor (α) is there to smooth the output and helps to prevent jitter between each steering command. The robot can use this set of computations to remain centered on the sidewalk or between obstacles even when faced with uneven sidewalk terrain or partial occlusions.

C. Components

The autonomous driving subsystem consists of integrated sensing, processing, and actuation modules designed for real-time sidewalk navigation. The primary sensor is the YD LiDAR X4 Pro, a 2D scanning LiDAR that performs 360° rotations at 6–12 Hz and captures up to 1000 range points per revolution. Distance data is transmitted via UART to the Raspberry Pi 5, which functions as the central processor. The Raspberry Pi executes a Python-based control algorithm that filters noise, segments valid points, and computes the steering command using the differential control law described earlier.

The computed control signal is then sent to an ESP32 microcontroller through serial communication. The ESP32 translates these commands into motor control signals, using PWM to adjust the left and right wheel speeds for path correction. Addressable LEDs connected to the ESP32 serve as real-time debugging indicators in the testing phases, visually reflecting direction or obstacle proximity without affecting the control logic.

Power is supplied through separate regulated sources: 5 V for the Raspberry Pi and LiDAR, 7.4–12 V for the motor drivers, and 3.3 V for the ESP32. Communication between subsystems uses low-latency TTL-level serial links to maintain responsiveness. Together, these components form a closed-loop control architecture that enables stable sidewalk navigation, dynamic obstacle avoidance, and self-correction based on continuous LiDAR feedback.

V. Web Traffic and Hosting

A. Purpose

The web traffic and hosting subsystem provides an online interface for browsing Trash and Treasure data among other things. The purpose of it is to establish a reliable bidirectional communication channel between the physical robot and a remote web server for logging data and errors. This will allow users to observe GPS-based labels of items and enable engineers to track movement and verify communication health during testing. The subsystem forms the networking backbone of the system and ensures that all the remote data transfers from the robot's SIM7600G-H LTE modem to the hosted web server in a secure and efficient manner.

B. Concept

A client-server model with standard internet protocols, where the physical robotic unit functions as the client and the host is a cloud-hosted flask web server. The cellular module provides the robot with network access via 4G data and allows HTTP or HTTPS communication with the remote server domain. The robot periodically transmits HTTP POST requests, which the flask backend parses and stores in a database for retrieval and visualization. These payloads are encoded as JSON objects, formatted as {"lat": <value>, "lon": <value>, "status": <value>}, allowing for consistent and language-agnostic data interchange.

On the server side, the Flask application is hosted using Nginx and Gunicorn on an Ubuntu-base system, providing secure and scalability in request handling. The backend properly routes incoming data to a SQLite database, ensuring persistence of location and sensor information. A lightweight front-end interface integrates an interactive web map (based on the open-source Leaflet JavaScript library), which plots the robot's reported coordinates within 1 second or its sending.

C. Operation

When the robot's control system generates a new GPS reading, it serially transmits the data to the SIM7600G-H module using AT commands. The module formats and executes the following operations sequentially:

1. Establish network attachment using AT+CGATT=1.
2. Initialize HTTPS mode with AT+CHTTPSSTART.
3. Open a secure TCP socket with the host domain (e.g., roambot.dev) via AT+CHTTPSOPSE.
4. Send an HTTP POST request containing JSON payload data to the endpoint /data using AT+CHTTPSPOST.

Upon successful transmission, the server responds with a standard HTTP status code (e.g., 200 OK), confirming that the data was received and logged.

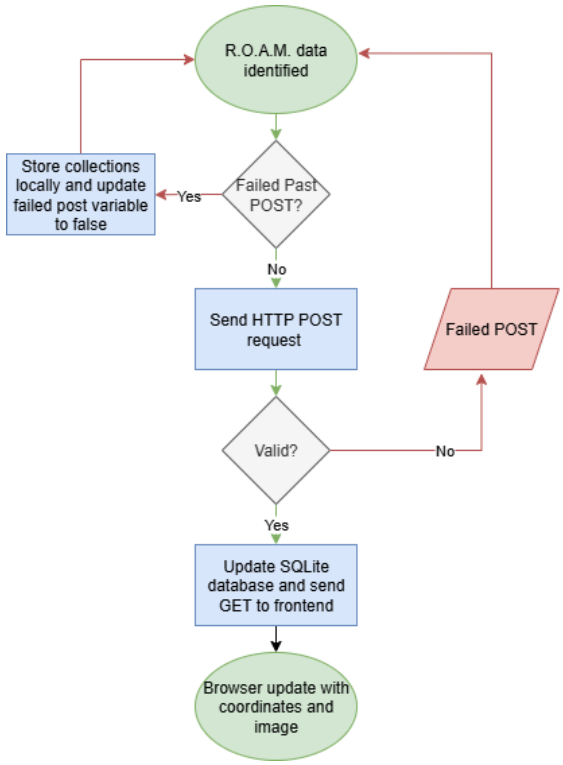


Fig 3. Web communication flowchart for R.O.A.M.

The robot firmware then closes the connection (AT+CHTTPSCLOSE) and may enter a short idle state before the next transmission cycle. This intermittent connection model reduces bandwidth usage and power consumption, making it well-suited for low-power mobile robots operating over cellular networks.

On the backend, the Flask application receives the POST data, processes it, and inserts it into the SQLite database through SQLAlchemy ORM. Once stored, this data becomes accessible to any connected client via the /latest or /history API endpoints. The web interface refreshes these endpoints at defined intervals (e.g., every 5 seconds) to update the Leaflet map, effectively displaying the robot's position in near real time.

D. Components

The web traffic and hosting subsystem combines cellular networking, data handling, and web visualization into a single system. A SIM7600G-H 4G module provides the robot's internet connection and communicates using standard AT commands. These commands manage network attachment and data sessions, allowing the robot to send telemetry through HTTP POST requests in a simple JSON format.

On the server side, a Flask application receives the incoming data and stores it in a SQLite database, chosen for its simplicity and low resource requirements. The data is then made available through an Nginx web host, which also serves an interactive Leaflet.js dashboard. This dashboard displays the robot's live position and historical routes in a clear, map-based view.

Together, these components create a smooth link between the robot's onboard systems and the web, making it easy to monitor performance and location in real time.

VI. Microcontroller System Integration

A. Purpose

The microcontroller subsystem is responsible for turning the Raspberry Pi's high-level decisions into concrete actions and user-visible feedback. In this project, the ESP32 acts as the central integration point that listens for compact JSON messages from the Raspberry Pi, converts steering requests into precise PWM signals for the motor drivers, and, when a classification event occurs, obtains GPS coordinates from the SIM7600G-H and forwards the result to the project website for visualization. By taking ownership of real-time I/O, signaling, and telemetry handoff, the ESP32 cleanly separates time-critical device control from the compute-heavy perception and autonomy that run on the Raspberry Pi.

B. Concept

At runtime, the Raspberry Pi sends double-quoted JSON over a 115200-bps UART link to the ESP32. Two small message families drive the entire behavior. Steering is represented as {"L": s} where s is a signed integer in [−256, 255]. The sign indicates which wheel is commanded, negative values address the left wheel and positive values address the right wheel, while the magnitude is mapped to the ESP32's 8-bit PWM output (0–255), yielding proportional motor control without additional fields. A value of zero has a special meaning which stops both wheels, and makes it easy for the autonomy stack to assert a safe halt condition with a single message. Classification is represented as {"C":"T"} for Treasure and {"C":"G"} for Garbage. When such a label arrives, the ESP32 updates on-board indicators for operator awareness, displaying "Treasure" with a green LED for T, or "Trash" with a yellow LED for G, and then queries the SIM7600G-H for the current GPS coordinates by issuing AT commands over its serial interface. After coordinates are available, the ESP32 packages the label and location and sends them to the

website (via AT commands) through the SIM7600, so that the event appears as a pin on the map. If the LiDAR stack determines the path is blocked by an obstacle that cannot be traversed, the ESP32 presents a red LED state so the blocked condition is immediately visible on the robot. In this arrangement, the Pi focuses on autonomy and vision, while the ESP32 provides deterministic handling of messages, motor outputs, indicator updates, and the cellular/GNSS exchange.

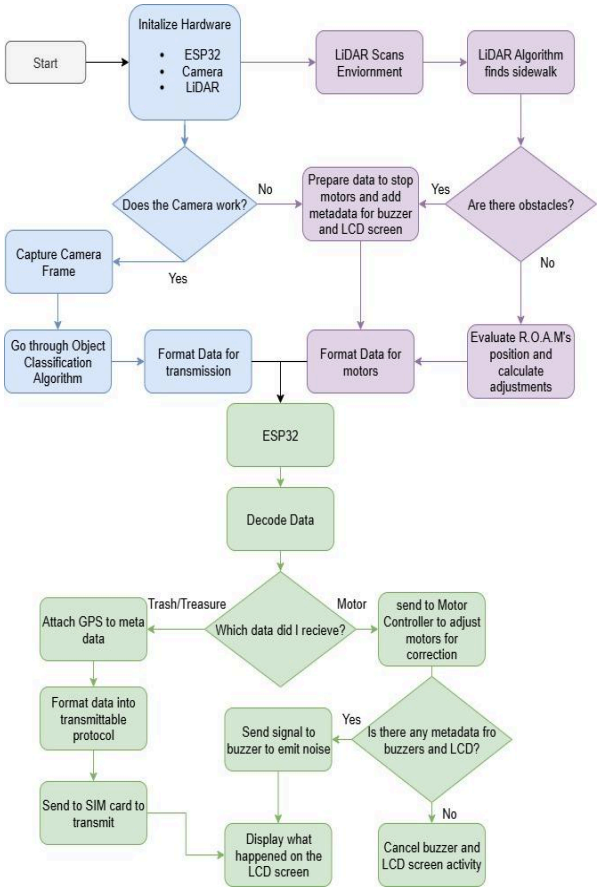


Fig 4. Testing for SIM7600

C. Components

The integration revolves around five cooperating elements. The Raspberry Pi executes the autonomous driving logic and object classification, producing concise messages that encode only what the ESP32 needs to act. The ESP32 serves as the message terminus and hardware controller, parsing JSON received over UART and translating those directives into PWM outputs, LED/LCD updates, and GPS reporting steps. The motor drivers receive the ESP32's PWM signals with an 8-bit duty resolution, which allows the single signed steering field to directly control left or right motion with proportional intensity. The SIM7600G-H provides both GNSS and the cellular path used to send event data to the project website; the ESP32 communicates with this module through standard AT commands on its serial pins, first requesting a GPS fix and then sending the label and coordinates once available. Finally, the user indicators on the robot—an LCD and a set of LEDs—offer immediate, readable feedback: "Treasure" with green for a positive classification, "Trash" with yellow for a negative classification, and red when the path is blocked. This division of responsibilities keeps the overall design simple to understand: high-level perception and planning happen on the Pi, and the ESP32 turns those decisions into motion, status, and location reporting.

D. Testing

We tested integration progressively: first each subsystem under bench/simulation load, then the full stack. For motion, we sent known steering values from the Raspberry Pi and confirmed the control rules—negatives drove the left wheel, positives drove the right, and zero stopped both. The signed steering mapped cleanly to 0–255 PWM, so motor effort tracked the command as expected. On the classification and telemetry side, we started by validating GPS. We ran a loop that posted GPS coordinates to the server once per second and checked accuracy by walking along a sidewalk with the antenna in hand.

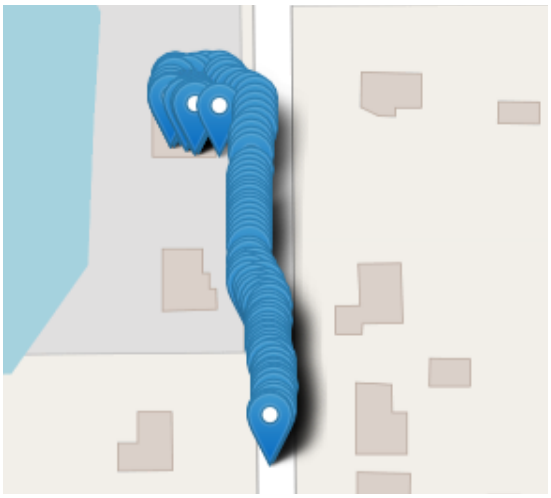


Fig 5. Testing for SIM7600

Next, we verified that the ESP32 could parse messages from the Pi. With that in place, we injected treasure and garbage labels, confirmed the LCD/LEDs reflected the labels, and watched the ESP32 query the SIM7600G-H via AT commands for coordinates. Once a fix was available, the ESP32 sent the label and location to the website and a matching map pin appeared, completing the perception-to-telemetry loop. We also exercised blocked-path cases from the LiDAR pipeline to ensure the red LED state was clearly presented on the robot.

After each piece passed in isolation, we ran end-to-end field tests. We monitored R.O.A.M. as it followed a sidewalk with pre-placed items and

confirmed that steering, classification, GPS posting, and map pins all worked together as intended.

VII. Conclusions

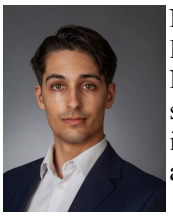
This project was developed as an autonomous, residential navigating, and object identifying robot. The goal was to achieve the objectives we have previously talked about while also working to promote sustainable systems for acquiring second hand items. Throughout the development of this project we have successfully integrated a power supply using PCB regulators that we designed, we have integrated LiDAR based navigation, machine learning classification models, a website interface, and GPS coordination. These parts working together are critical for the overall functionality of this project, and to meet our standards we needed to have the navigation keep R.O.A.M. within the bounds of our sidewalk for 25 meters, have an upload time of less than 10 seconds, and an object detection rate of 70% for items that are between 0.5m and 3m away. To meet these objectives, we had to spend time on each sub system to make them functional so that when the full integration took place we could eliminate certain problems. The final prototype for this project combined two microcontrollers, one that handled delegating tasks on the robot and our website, and another that was solely responsible for the LiDAR and machine learning. All items were connected through the ESP32 which served as the main functional brain for this robot. When combining the multiple sub systems there were certain challenges, such as limiting power consumption, communication issues between the ESP32 and Raspberry Pi, and overall hardware and PCB issues. All of these complications were able to be solved eventually, but the process of solving these complicated issues meant that our group spent significant time working on each individual piece to troubleshoot.

The Engineers

Daniel Claassen is currently a senior at the University of Central Florida and will graduate with his bachelors of science Computer Engineering.

Claire Persinger is currently a senior at the University of Central Florida and will graduate with her bachelors of science in Electrical Engineering on the Power Track. She currently works as a substation protection and control design engineer at Commonwealth Associates.

Ethan Robotham is a senior at the University of Central Florida and will graduate with honors, earning a bachelor of Science in Computer Engineering. He works as an embedded engineer at Lockheed Martin and plans to continue his education by pursuing a master's degree in electrical engineering.



Nathan Little is a 24 year-old Computer Engineering student on the VLSI track. Nathan hopes to pursue a career in the specialized area of human-machine interfaces, working for a company such as Paradromics, Synchron, or Neuralink.

Acknowledgment

The authors wish to acknowledge the assistance and support of Dr. ChungYong Chan, Dr. Arthur Weeks, Dr. Justin Phelps, Dr. Xin Xin and the UCF Texas Instruments Lab.

References

[1]
X. Wang and J. G. Walters, "LiDAR-based Terrain Recognition in Off-road Mobile Robot," Proceedings of the 33rd International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+ 2020), Sep. 2020, doi: https://doi.org/10.33012/2020.17742.

[2]
J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2016.

[3]
Ultralytics, "YOLOv11 Documentation," 2024. [Online]. Available: https://docs.ultralytics.com

[4]
A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple Online and Realtime Tracking," *Proc. IEEE Int. Conf. Image Processing (ICIP)*, 2016.

[5]
Raspberry Pi Ltd., "Camera Module 3 Product Page," 2023. [Online]. Available: https://www.raspberrypi.com/products/camera-module-3/

[6]
G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[7]
Raspberry Pi Ltd., "Picamera2 Documentation," 2023. [Online]. Available: https://github.com/raspberrypi/picamera2