ECE 445

SENIOR DESIGN LABORATORY

Final Report

Autonomous Wi-Fi Mapping Car

Autonomous Car to Map Wi-Fi Signal Strength in a Room

Team No. 2

JOSH POWERS
(jtp6@illinois.edu)

BEN MAYDAN
(bmaydan2@illinois.edu)

AVI WINICK
(awinick2@illinois.edu)

TA: Jason Jung
Professor: Arne Fliflet

December 10, 2025

Abstract

This report provides a detailed description of the design and implementation of our project, an **Autonomous Wi-Fi Mapping Car**, a car designed to navigate any indoor environment with Wi-Fi, and use an integrated antenna built into an ESP32 to generate a heat map of the Wi-Fi signal strength at many different locations. The system is built on a custom PCB, which is used to distribute power and connect signals between the onboard ESP32 and external components such as our LiDAR [14] and a Raspberry Pi which we used to offload most of the intensive computational algorithms. Verification testing proved our power distribution system worked as intended, our run time was sufficiently long, and the car was able to drive autonomously in an environment and show a gradient heat map of the Wi-Fi strength.

Contents

1 Introduction

1.1 Problem Statement

Wireless local area networks are fundamental to modern environments, yet their performance is frequently compromised by architectural interference and "dead zones." Identifying these coverage gaps typically relies on manual surveys. These are labor-intensive and time-consuming processes that often yield inconsistent, low-resolution data [3] [11]. To overcome the inaccuracies of this guesswork-based approach [1], this project introduces an autonomous mobile robot capable of generating high-resolution signal strength maps. By leveraging Simultaneous Localization and Mapping (SLAM) to navigate and correlate Received Signal Strength Indicator (RSSI) values with precise location data, the system creates detailed heat maps to effectively automate network diagnostics and infrastructure optimization [2].

1.2 Proposed Solution

To address the need for indoor network analysis, we propose an autonomous RC car designed to map Wi-Fi signal strength and generate a visual heat map. The system utilizes an omnidirectional car with a Light Detection and Ranging (LiDAR) sensor, an ESP32 microcontroller for low-level hardware control and path planning, and a Raspberry Pi for SLAM. All onboard electronics, including power distribution from a Li-Ion battery, are integrated via a custom PCB. The system operation is divided into two phases: an initial manual control phase to construct a 2D map, followed by an autonomous phase to gather signal data [3].

We propose a robust communication hierarchy to manage these phases. During the manual phase, a user controls the vehicle via a custom host computer Graphical User Interface (GUI) over Bluetooth, while the ESP32 relays parsed LiDAR data to the Raspberry Pi over UART to construct the map in real-time. Once the user engages autonomous mode, the ESP32 calculates the most efficient trajectory to cover the mapped area. As it navigates, the system continuously correlates Wi-Fi RSSI values with specific SLAM coordinates, eventually transmitting the dataset back to the host computer to render the final visualization.

The final product successfully achieved the majority of its high-level requirements, ensuring robust control and data visualization. First, the device allows for responsive manual control from a host computer via Bluetooth, executing all directional commands with a latency of less than 500 milliseconds from a distance of 5 meters. Regarding the autonomous mapping requirement, the car demonstrated the capability to drive a planned 5-meter route and generate a 2D map during the final demo; however, a visual map for this report is unavailable due to a post-demo wiring incident that damaged the Raspberry Pi’s GPIO receiver. We showed SLAM working in previous breadboard demos. Despite this hardware setback, the system fully delivered on its core purpose by successfully measuring Wi-Fi signal strength at distinct locations and generating a heat map that uses at least three different colors to accurately represent varying signal intensities.

1.3 Visual Aid

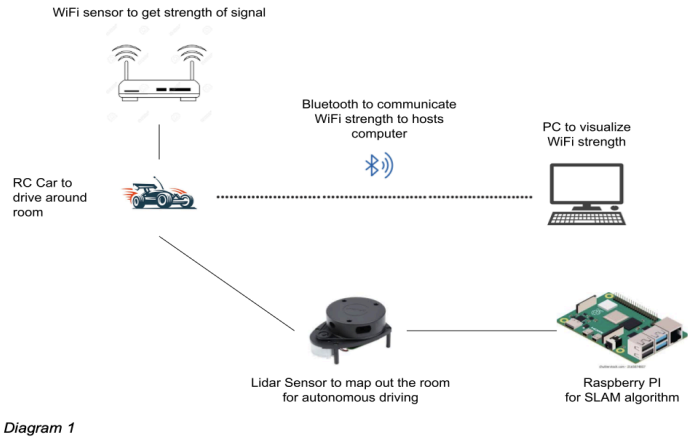

Image 1: Visual Aid for Project

Image 2 presents a high level overview of the autonomous Wi-Fi mapping car system architecture and component interaction. The RC car serves as the centralized hub, using LiDAR and a Raspberry Pi to navigate its environment. The car is also responsible for driving and receiving Wi-Fi signal strength data at many discrete locations. The Raspberry Pi receives the LiDAR data points and runs the Simultaneous Localizing and Mapping algorithm to build a map of the room and its obstacles. Once the map is completed, the car drives autonomously, collects the RSSI data at many different coordinates within the (X,Y) plane it's driving on, and generates a signal strength heat map that is visualized on the host computer.

1.4 Block Diagram

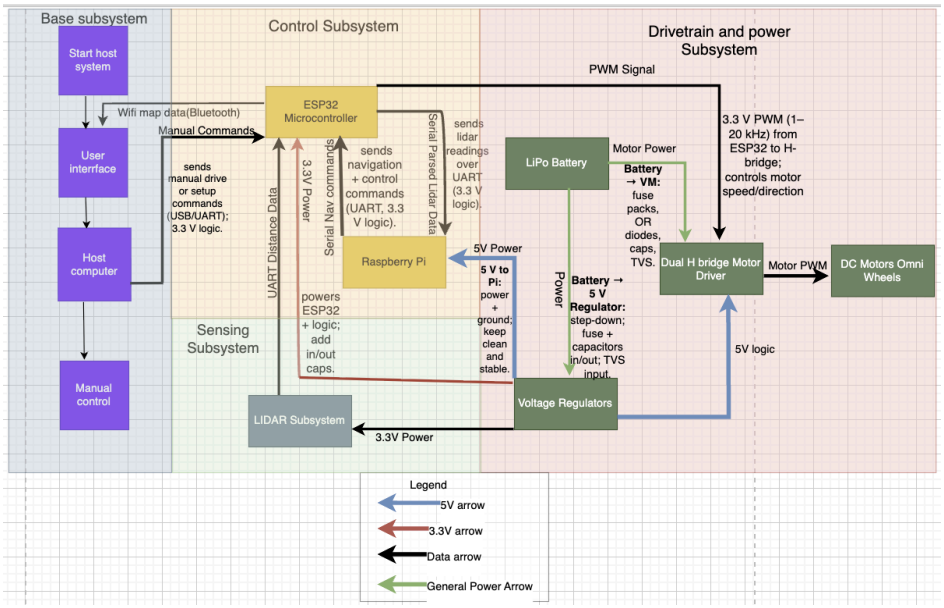

Figure 1: block diagram

Figure 1 shows the high level system design, broken down into four subsystems: Base, Control, Sensing, and Drivetrain & Power. The **Base Subsystem** holds the host computer and user interface, initializing the system and communicating commands during the initial driving phase. The **Control Subsystem** uses the ESP32 microcontroller as the main interface for all of the communication between components, routing data between the host, **Sensing Subsystem (LIDAR)**, and the Raspberry Pi. The Pi receives the LIDAR points and uses them to build a map. The **Drivetrain & Power Subsystem** is responsible for energy distribution to all the various components via the PCB, utilizing two 7.4 V Li-Ion batteries and voltage regulators to supply separate 5 V and 3.3 V rails, while delivering high current power to the dual H-bridge motor controllers for the omnidirectional wheels.

2 Design

2.1 Design Procedure

Our design consists of four subsystems: control, sensing, drivetrain, and power. Each subsystem has its own functions and requirements. At the center of the system is the control subsystem, consisting of a two-tiered processing model using an ESP32-S3 microcontroller for real-time hardware control and a Raspberry Pi 4 for high-level SLAM computation. The goal of this subsystem is to decouple the computationally heavy mapping algorithms from the timing-critical motor actuation, ensuring that the SLAM process does not cause delays in the drivetrain. The main design decisions for the control subsystem involved the communication architecture; while we initially attempted a simple serial stream, we found it unreliable for large data transfers like map files. Consequently, we pivoted to a robust, packet-based protocol using Consistent Overhead Byte Stuffing (COBS) to guarantee data integrity. Additionally, we implemented a custom chunking protocol to prevent buffer overflows on the ESP32 when receiving large map data blobs from the Pi.

The physical hardware design evolved through iterative testing, particularly regarding the Drivetrain and Sensing subsystems. We originally verified the wiring using socketed motor driver modules, then transitioned to integrating the TB6612FNG drivers directly onto the PCB to reduce size and improve reliability. For the sensing subsystem, we discovered during bench testing that the LiDAR motor drew a peak current of 500mA during spin-up, which was significantly higher than the datasheet specification. This necessitated the addition of a 470 µF bulk capacitor and specific layout isolation to keep motor ripple out of the digital return paths. We also implemented a USB-C interface configured as a device with specific termination resistors to ensure reliable programming and communication without complex external dongles.

For the power subsystem, we determined that the system needed to sustain operation for approximately one hour under load. Calculating the worst-case current draw revealed a peak demand of roughly 8.5A, primarily driven by the motors and LiDAR. Consequently, we selected two 7.4V (2S) 3300mAh Li-Ion batteries capable of high discharge rates. The most critical design decision was splitting the power distribution into two isolated paths: one for the "noisy" high-current motors and one for the "quiet" sensitive logic chips. This decision was driven by

early tests where the Raspberry Pi suffered brownouts and crashes when the motors stalled or the LiDAR spun up [10]. To implement this, we utilized a P-channel MOSFET for low resistance and efficient high-current handling and a Schottky diode for fast switching and reverse polarity protection. We also added significant bulk and ceramic decoupling capacitors across the board to further stabilize the rails against the transient spikes observed during testing.

2.2 Design Details

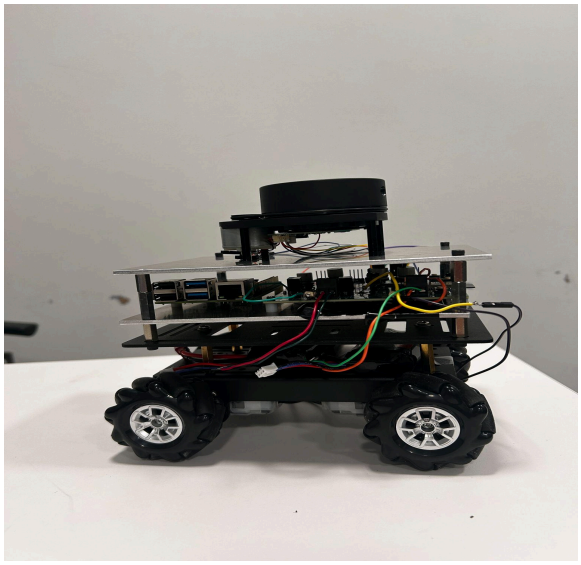

Image 2: Physical design (includes batteries, PCB, and lidar with third layer on top)

2.2.1 Base Subsystem

The Base Subsystem functions as the remote interface for the system, hosted on a computer to manage high-level state control and data visualization. We designed this subsystem to communicate via Bluetooth sending manual velocity commands directly to the ESP32. While it does not interact directly with the sensors or motors, it acts as the final hub for the data collection pipeline. Upon completion of the autonomous mapping phase, the subsystem receives the position and signal strength data, structured as $(x, y, RSSI)$ tuples, which it then parses to render the visual heat map of the Wi-Fi coverage.

2.2.2 Control Subsystem

This subsystem contains the ESP32 microcontroller and the Raspberry Pi. The ESP32's functions are to (1) maintain the Bluetooth link with the Base Subsystem, receive manual commands/start signals, and send Wi-Fi map data; (2) generate PWM drive signals to the dual H-bridge motor driver in the Drivetrain & Power Subsystem; (3) parse incoming LiDAR distance/angle data (UART) from the Sensing Subsystem and forward the data to the Raspberry Pi, and (4) indicate when the final Wi-Fi map data is ready for the Base Subsystem.

Hardware Design and Signal Integrity

To ensure reliable communication between the ESP32 and the Raspberry Pi, we placed 220Ω series resistors on the UART lines. These limit transient currents and reduce high-frequency noise without affecting signal integrity. The calculated RC time constant for these lines is negligible for our baud rate:

$$T = R \cdot C_{trace} \approx (220\Omega)(20pF) = 4.4 \text{ ns}$$

Additionally, the reset network for the ESP32 was designed with a specific RC delay to filter switch bounce and guarantee a valid low signal during reboot:

$$T_{reset} = (10k\Omega)(0.1\mu F) \approx 1 \text{ ms}$$

The Raspberry Pi runs navigation: It ingests serial LiDAR data from the ESP32, performs mapping and localization, and returns localization data to the ESP32 for execution. To manage this data flow, we implemented a custom packet protocol using defined Opcodes, such as PI_READY ($0x01$) and MAP_DATA ($0x02$). This packet structure was critical because the Raspberry Pi's map transmission speed initially overwhelmed the ESP32's hardware limitations, specifically its 128-byte UART receive buffer, necessitating a chunked-transfer approach and more complex packet encoding.

Power-wise, the ESP32 receives 3.3 V and the Raspberry Pi 5 V from the Drivetrain & Power Subsystem's regulators [10]. Thus, Control is the hub: Bluetooth to Base, UART to Sensing, PWM to Drivetrain, and regulated power in from Drivetrain & Power.

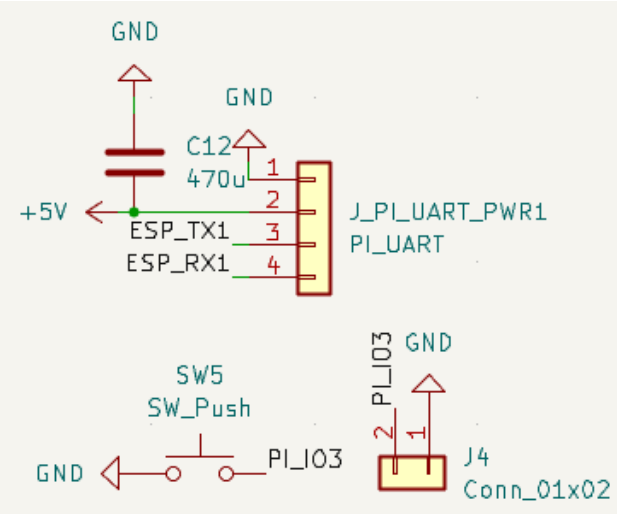

Figure 2: Raspberry Pi Header and Shutdown Button

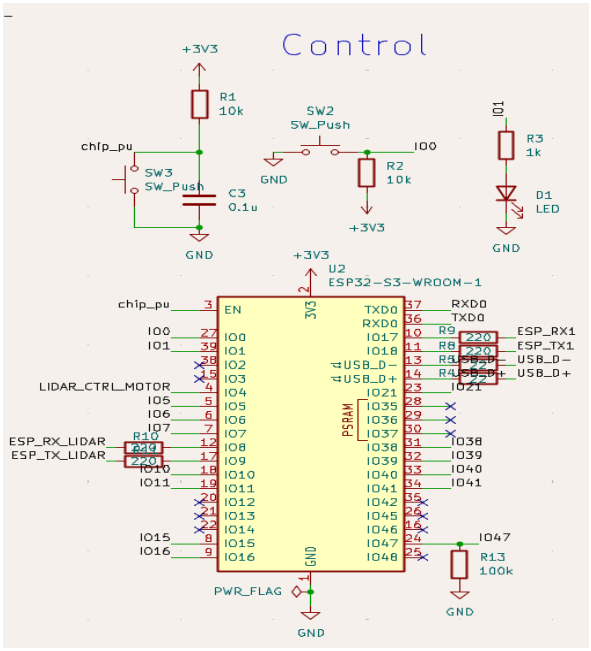

Figure 3: ESP32 and IO pin connections

2.2.3 Sensing Subsystem

This subsystem is the LIDAR sensor (RPLIDAR A1), powered at 5V from the Drivetrain & Power Subsystem's regulators. When the RC car is started, the Control Subsystem initializes the LIDAR, after which the sensor continuously streams UART distance/angle data to the ESP32 within the Control Subsystem.

Electrical Interface and Signal Integrity

The LIDAR interfaces via a seven-wire pinout that separates the logic power (VCC_5V) from the internal motor power ($5V\_MOTO$). To ensure data integrity on the UART transmission lines (TX/RX), we placed 220Ω series resistors near the driver. The RC time constant introduced by these resistors is negligible for the data link:

$$\tau = R \cdot C_{trace} \approx (220\Omega)(20\text{ pF}) = 4.4\text{ ns}$$

Power Stability and Transient Analysis

Although the LIDAR has no direct logical connection to the motor driver, its internal spin-motor creates significant electrical noise. During testing, we discovered the LIDAR motor draws a peak current of 500mA during spin-up, exceeding the documented 100mA specification. To prevent this load from corrupting the logic supply, we placed a $470\mu F$ bulk capacitor at the motor power header. We calculated the expected voltage droop (ΔV) during a spin-up transient to confirm the regulator's headroom:

$$\Delta V = \frac{I \cdot \Delta t}{C} = \frac{(0.3\text{ A})(0.5\text{ ms})}{470\mu\text{F}} \approx 320\text{ mV}$$

This result confirmed that the rail would remain stable enough for the upstream regulator to absorb the transient, enabling the Raspberry Pi (via the ESP32) to build the map and produce navigation commands without brownouts or undervoltages.

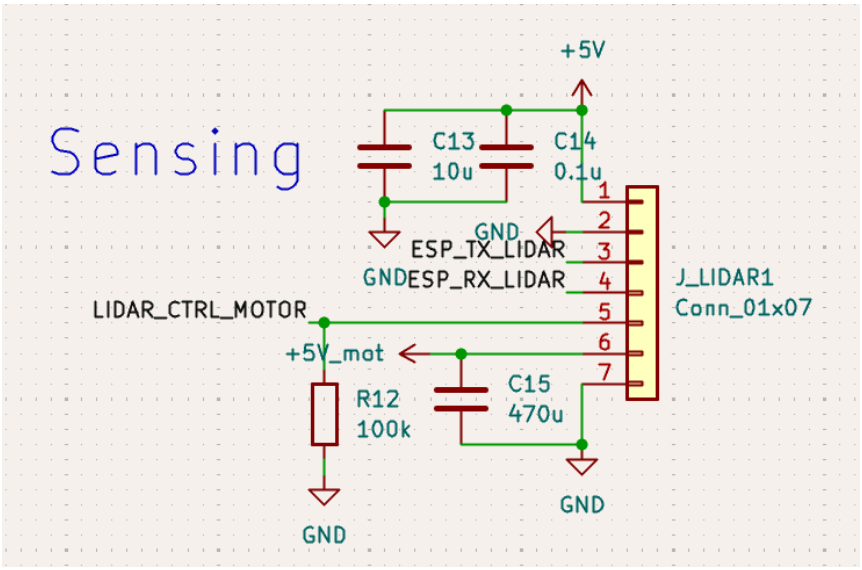

Figure 4: LiDAR Header

2.2.4 Drivetrain & Power Subsystem

Two 7.4V (2S) 3300mAh Li-Ion batteries feed two distinct power paths: (1) high-current motor power sent directly to the dual H-bridge motor drivers, and (2) voltage regulators that generate the stable logic rails. Our design utilizes these two separate paths to isolate sensitive logic components, such as the ESP32 and Raspberry Pi, from the motor path which draws significantly higher current. The system "start" action enables the regulators so all downstream electronics power up in order.

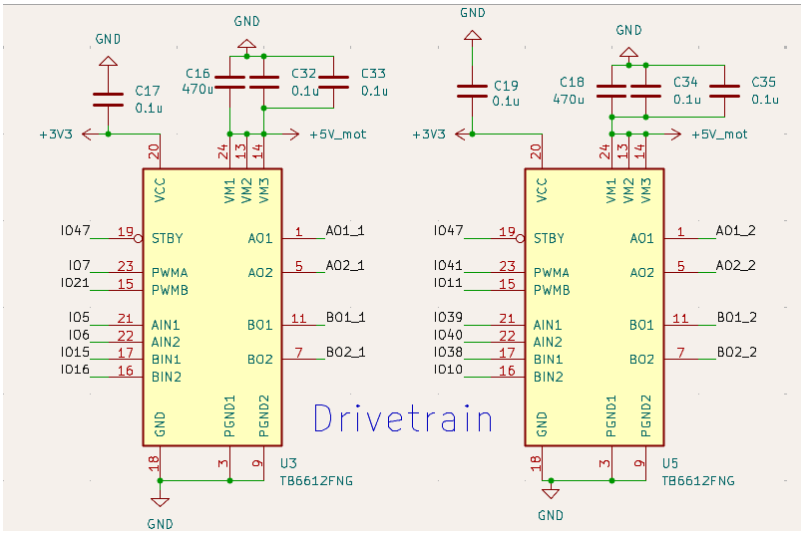

Figure 5: TB6612FNG Chip Layout

Current Analysis and Power Paths

To ensure our runtime exceeded 15 minutes, we calculated the maximum possible current draw for each path separately using component datasheets. The total current I_{total1} for the logic path is defined by the equation:

$$I_{ESP32} + I_{Pi} + 2 \times I_{TB6612FNG} = I_{total1}$$

Using datasheet values of 0.5 A for the ESP32, 3 A for the Raspberry Pi, and 0.1 A for the motor driver logic[9], we calculated a peak logic current of 3.6 A.

The motor path current (I_{motor}) is defined by the equation:

$$I_{LiDAR} + 4 \times I_{wheel\_motor} = I_{motor}$$

The LiDAR draws 0.5 A, and the worst-case scenario current draw from the four motors was calculated at 8 A, resulting in a theoretical total of 8.5 A. However, during verification, we measured the actual total current for this path to be approximately 4.5 A under normal driving conditions and 5 A under starting conditions.

Battery Runtime and Stability

To verify that the 3300mAh battery met our runtime requirements, we calculated the power draw based on the expected 4.5A motor load:

$$P_{out} = V_{out} \times I_{out} = 5V \times 4.5A = 22.5W$$

Assuming 90% efficiency in the regulators, the load on the battery is approximately 25W. The current drawn from the 7.4V battery is therefore:

$$I_{battery} = 25W/7.4V \approx 3.37A$$

This results in an estimated runtime of approximately one hour ($24.4Wh/25W$), well exceeding the 15-minute requirement.

Drivetrain and Transient Management

The drivetrain utilizes two TB6612FNG dual H-bridge drivers to control the four DC motors. To prevent voltage sags during motor actuation from affecting the power rails, we placed $470\mu F$ bulk capacitors at the driver inputs. We sized these capacitors using the voltage droop equation:

$$\Delta V = \frac{I \cdot \Delta t}{C}$$

For a 0.5 A load step lasting 0.5 ms, the calculated droop is:

$$\Delta V = \frac{0.5A \cdot 0.5ms}{470\mu F} \approx 532mV$$

This droop is within the tolerance of the 5V rail and upstream regulators. Verification tests confirmed that without these capacitors, voltage sagged by 200 mV, but with them, the sag was reduced to 100 mV, preventing brownouts.

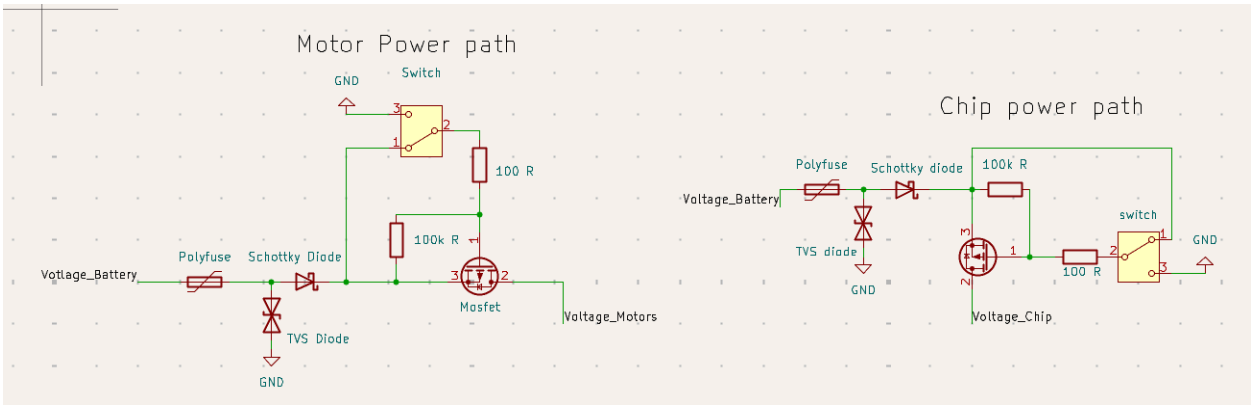

Figure 6: Battery protection

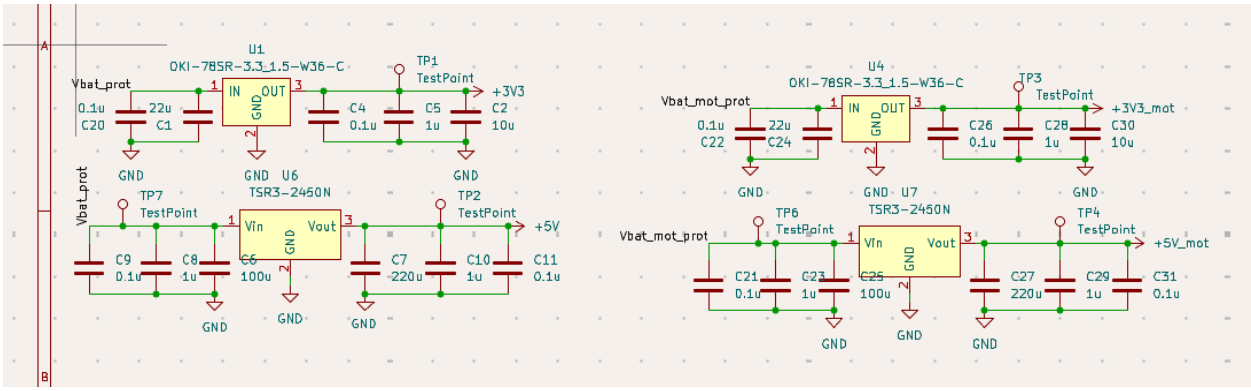

Figure 7: Motor controllers

Figure 6 shows the circuit protection for the power distribution component on our PCB. It shows the dual path layout that was previously mentioned, involving a chip path and motor path. To ensure reliability and safety, each path begins with a polyfuse to limit excess current, then followed with a TVS diode wired in parallel to ground to clamp voltage spikes, and then a schottky diode to prevent any damage from wiring the batteries wrong. The mosfets are used as switches

2.3 Software Design

2.3.1 Bluetooth Communication

The ESP32 microcontroller's integrated Bluetooth Low Energy (BLE) capabilities are leveraged for wireless communication between the RC car and the host computer's GUI. BLE is selected for its power efficiency, which is critical for a battery-operated device, and its robust range,

ensuring a stable connection within a typical indoor environment. The Arduino IDE's core ESP32 libraries, including BLEDevice.h and BLEServer.h, will provide the framework for this communication link. These libraries facilitate the setup of the ESP32 as a BLE server, allowing it to define services and characteristics that the host computer client can interact with.

To establish the communication protocol, the ESP32 is configured as a BLE server with a custom service. This service exposes two primary characteristics, each with a unique UUID, to handle the bidirectional flow of information:

1. A write characteristic is defined for the host computer to send commands to the RC car. This allows the user to control the car's operation from the GUI. The commands are sent as short strings and include manual driving instructions (e.g., "FWD", "ROT_L") and high-level state changes (e.g., "START_AUTO", "E_STOP").
2. A notify characteristic is defined for the RC car to transmit the collected Wi-Fi signal data back to the host computer. After the autonomous mapping is complete, the ESP32 compiles the list of (x, y, RSSI) data points into a single data structure, likely a JSON formatted string. It then sends this data payload to the subscribed host computer using notifications. This method is efficient for transmitting the complete dataset once it's ready for visualization.

The operational flow begins with the ESP32 initializing its BLE server and advertising its custom service. The user then initiates a connection from the GUI on the host computer. Once connected, the GUI can send commands via the write characteristic. During the initial manual phase, this allows for real-time teleoperation to map the room's boundaries. When the user initiates the autonomous phase, the ESP32 switches modes and begins its programmed path. Upon completion, it transmits the final heat map data back to the GUI via the notify characteristic, which then parses the data and renders the final visualization [3] [7].

2.3.2 Signal Control

The project's signal processing is managed by a two-tiered system, with the Raspberry Pi handling high-level computation and the ESP32 managing real-time hardware control. The Raspberry Pi processes complex LiDAR data to execute the SLAM algorithm and perform path planning, while the ESP32 interfaces directly with the motor driver, sensors, and the Bluetooth module. This division of labor ensures that time-sensitive tasks like motor control and data collection are not delayed by heavy computational loads.

Motor and Drivetrain Control: The ESP32 is responsible for translating both manual and autonomous commands into precise electrical signals for the drivetrain.

- **Manual Mode**: When a command like "STRAFE_R" is received over Bluetooth, the ESP32 generates four unique Pulse-Width Modulation (PWM) signals. These signals are sent to the dual H-bridge motor driver, which in turn powers the four omnidirectional wheel motors with the correct speed and direction to achieve the desired lateral movement.

- **Autonomous Mode**: The map is received by the ESP32 which then proceeds to generate the path. As the ESP32 drives the path, the Raspberry Pi sends localization updates to the ESP32 so it knows which part of the path it is currently on.

LiDAR and SLAM Data Flow: The system's spatial awareness is achieved through a continuous flow of data between the sensing and control subsystems.

- The LiDAR sensor continuously rotates and streams raw distance and angle data to the ESP32 via a UART connection.
- The ESP32 parses this data stream into a clean format and immediately relays it to the Raspberry Pi over a second, dedicated UART connection.
- The Raspberry Pi ingests this steady stream of spatial data, using it as the input for its SLAM algorithm to build the 2D map and continuously update the car's (x, y) coordinates in real-time.

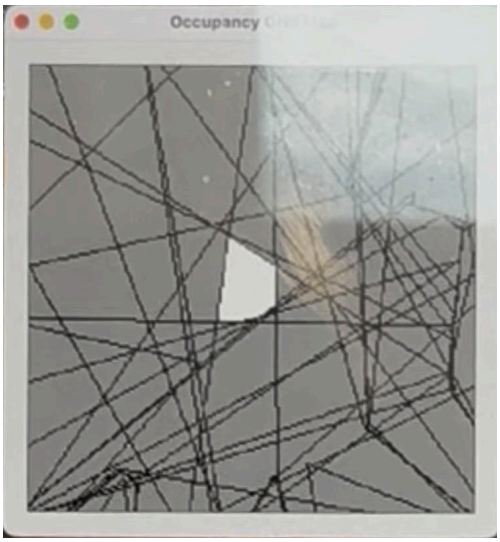

Figure 8: Generated SLAM Map

In figure 8, the total map area in code was 8x8 meters which is why the unoccupied space looks smaller relative to the map. In addition, the poster board room we constructed was slightly triangular at the top to make the room interesting. We are unsure what the black line artifacts are from. It is possible our room was too featureless and thus the SLAM algorithm could not localize or make a map correctly. Thus it is possible our redundancy code kicked in here.

Wi-Fi and Coordinate Synchronization: During the autonomous phase, the ESP32 collects Wi-Fi signal strength data and correlates it with location data from the Raspberry Pi. As the Raspberry Pi executes its path-planning algorithm, it continuously sends the car's current (x, y) coordinate, as determined by the SLAM algorithm, to the ESP32. Concurrently, the ESP32's Wi-Fi module samples the RSSI of the target network at a fixed interval. Each time an RSSI measurement is taken, the ESP32 pairs it with the most recently received (x, y) coordinate and stores the (x, y, RSSI) triplet in an array in its memory, eventually sending the full array over bluetooth back to the host computer to be visualized [3] [8].

3 Verification

3.1 Base Subsystem

The Base Subsystem is responsible for sending manual commands and visualizing the data.

The first requirement is that the host computer must run a GUI capable of sending manual drive commands. We verified this by launching the custom Python application on a laptop and pressing the on-screen directional controls. We observed the corresponding command strings appearing in the application's debug console with zero perceptible latency.

The second requirement is that the system must render the received data as a visual heat map. After a test run, we triggered the data transfer sequence. We verified this by confirming the host received the coordinates and RSSI integers. The application successfully parsed this list and generated a color-coded 2D plot, where areas near the router appeared green (high signal strength) and distant areas appeared red (low signal strength).

3.2 Control Subsystem

The Control Subsystem handles the logic, communication, and mapping algorithms.

The first requirement is that the ESP32 must maintain a stable Bluetooth connection. We verified this by pairing the laptop to the ESP32 and maintaining an active session for 20 minutes while sending continuous "heartbeat" packets. The connection remained stable with no dropped packets recorded in the serial monitor.

The second requirement is that the Raspberry Pi must execute the SLAM algorithm to generate an accurate map and also drive the path autonomously. We verified this by placing the car in a 3 x 3 area of poster boards (close to rectangular but more triangular at the top to make the room slightly more interesting) and saw that the autonomous phase was completed successfully. The displayed map is unintuitive to interpret since SLAM is a probabilistic algorithm.

The third requirement is the accurate correlation of Wi-Fi data. We verified this by placing the car at a known location $(0, 0)$ and triggering a measurement.

3.3 Sensing Subsystem

The Sensing Subsystem consists of the LIDAR unit, which must provide accurate spatial data to the Raspberry Pi.

The first requirement is that the LIDAR must provide a 360-degree scan. We verified this by connecting the sensor's USB interface to a PC running visualization software. We placed distinct objects at $0°, 90°, 180°, \text{and } 270°$ relative to the chassis. The visualization confirmed obstacles appeared at the correct angular positions with no blind spots.

The second requirement is distance accuracy. We placed a flat board at measured distances of 1 meter, 3 meters, and 5 meters. We compared the LIDAR's reported distance over the UART stream against a manual tape measure reading. The sensor consistently reported distances within 2 cm of the physical measurement, satisfying the requirements for indoor mapping.

3.4 Drivetrain and Power Subsystem

The Drivetrain and Power Subsystem manages energy distribution and physical movement.

The first requirement is that the power system must supply stable 5 V and 3.3 V rails. We verified this by probing the output of the regulators with a multimeter while the motors were running at full speed. The 5 V rail measured a steady 5.02V, and the 3.3V rail measured 3.29V.

The second requirement is that the battery must power the system for at least 15 minutes; this was verified by charging the batteries and running the car in autonomous mode continuously. The system ran for ~20 minutes with no issues.

4 Cost and Schedule

4.1 Cost Analysis

Based on average College of Engineering salaries of $118,752 for Computer Engineering [4] and $88,321 for Electrical Engineering [5], we estimated hourly rates of $57 and $42. Applying the 2.5× overhead multiplier to 140 hours per member and 6 hours of machine shop time (at $84/hr) [6] results in a total labor cost of $50,610. With prototype parts costing $290.82, a figure likely to drop to $175 to $200 in mass production, the total estimated project cost is $50,900.82.

Labor					
Team Member	*$/hr*	*Hours Worked/week*	*Weeks Worked*	*Total Cost*	*Rate×Hours×2.5*
Josh Powers	$42	10	14	$14,700	
Ben Maydan	$57	10	14	$19,950	
Avi Winick	$42	10	14	$14,700	
Machine Shop	$84	3	2	$1,260	
				Total Labor Costs:	$50,610
Parts					
Description	*Manufacturer*	*Part Number*	*Quantity*	*Unit Cost*	*Total Cost*
Chassis / Car Kit	LewanSoul	B093WDD9N5	1	$36.99	$36.99
Lidar	Slamtec	A1M8	1	$99	$99
Raspberry Pi 4 8GB	Raspberry Pi Foundation	Adafruit 4564	1	$82.50	$82.50
Dual Motor Drivers	Teyleten Robot	TB6612FNG	2	$8.99	$17.98
Tactile Pushbutton	Omron Electronics Inc	SW415-ND	3	$0.47	$1.41
Slide Switch	C&K	401-2016-1-ND	2	$1.14	$2.28
PMOS	Diodes Incorporated	31-DMP4010SK3-13CT-ND	2	$1.65	$3.30
Schottky Diode	Comchip Technology	641-1707-1-ND	2	$0.45	$0.90
TVS/Signal Diode	Nexperia USA Inc.	1727-3837-1-ND	2	$0.33	$0.66
TVS Diode	Littelfuse Inc.	SMBJ13ALFCT-ND	2	$0.30	$0.60
Fuse Holder	Littelfuse Inc.	F1889-ND	2	$2.50	$5.00
USB C Receptacle	GCT	2073-USB4085-GF-ACT-ND	1	$0.88	$0.88
ESP32-S3-WROOM-1	Espressif Systems	5407-ESP32-S3-WROOM-1-N8CT-ND	1	$5.49	$5.49
Murata DC-DC Module	Murata Power Solutions Inc.	811-2195-5-ND	2	$4.97	$9.94
TRACO DC-DC Module	Traco Power	1951-TSR3-2450N-ND	2	$8.45	$16.90
HDMI Cable	Amazon Basics	B00NH11KIK	1	$6.99	$6.99
				Total Part Costs:	$290.82
				Grand Total:	$50,900.82

Table 1: Labor & Parts Description

4.2 Schedule

Week of	Task	Group Member(s)
9/15	Divide tasks	All
	Create Proposal	All
9/22	Proposal Review	All
	Meet with machine shop for chassis deisgn	All
	Find and order main subsytem components	All
	Code ESP32 to take commands	Ben
	GUI on host computer for commands over bluetooth	Ben
	Begin schematic design and power subsystem	Josh and Avi
9/29	PCB Review	All
	Finish schematic and pass ERC	Josh and Avi
	Set up footprints and find PCB components	Josh and Avi
	Begin LIDAR and SLAM software setup	Ben
10/6	Breadboard Demo 1	All
	Finalize PCB layout and routing	All
	First Round PCBway Order	All
	Order components for PCB	Josh and Avi
10/13	Design Document	All
	Code to send lidar data from ESP32 to Raspberry Pi	Ben
	Test and verify motor controllers to integrate into PCB	Josh and Avi
	Second Round PCBway Order, order components	All
10/20	Run SLAM on Raspberry Pi to take data and produce map	Ben
	Test motor current draw and battery selection	Josh and Avi
10/27	Breadboard Demo 2	All
	Update GUI to include start for autonomous driving	Ben
	Solder PCB and integrate into chassis	Josh and Avi
	Verify voltage regulation and signal integrity from components	Josh and Avi
11/3	Third Round PCBway Order	All
	Code the Raspberry Pi to be able to generate path	Ben
	Further component testing and PCB updates	Josh and Avi
11/10	Fourth Round PCBway Order (if necessary)	All
	Send location and instructions to ESP32 in autonomous phase	Ben
	Test autonomous navigation, SLAM accuracy, data transmission	All
11/17	Mock Demo	All
12/1	Final Demo	All
	Mock Presentation	All
12/8	Final Presentation	All
	Final paper	All

Figure 9: Schedule of the project

5 Conclusion

5.1 Accomplishments

We were able to build an autonomous mobile robot that successfully surveys indoor environments to identify Wi-Fi coverage dead zones. The system accurately measures signal strength at distinct locations, generating a visual heat map with varying colors to represent signal intensity drop-offs. Furthermore, the custom GUI allows for seamless manual control via Bluetooth with minimal latency before engaging the autonomous phase. Using this tool, users can eliminate the guesswork of router placement without the need for laborious manual walkthroughs, ensuring a more optimized wireless infrastructure [2].

Using this product, we can ask anyone to take the car and use it in their space. The user only needs to know how to start running the GUI. This car is very easy to use and not complicated, so the onboarding process is very simple for a new user. A benefit is that hiring someone to do this process manually is much more expensive than our relatively cheap car [11].

5.2 Uncertainties

Some things did not go according to plan, specifically regarding software and power consumption. For example, the open-source SLAM library we utilized contained a bug that occurred when the robot was stationary (the "no error gradient" error) [12]. After much work, we were able to patch the C source code. Consequently, we decided to implement a redundancy mechanism that constructs a map from a long list of high quality LIDAR points if the SLAM algorithm times out, which benefitted the overall quality of our project significantly.

In addition, our initial power analysis was based on the LIDAR manufacturer's datasheet, which stated a typical current draw of 100mA [13]. However, experimental testing revealed the sensor drew closer to 350mA, with spikes up to 3A during motor spin-up, causing the Raspberry Pi to brown out. We had to redesign the power distribution system mid-project to include a high-current regulator capable of handling these unexpected loads. Finally, data transmission was more difficult than anticipated; the UART buffer on the ESP32 was easily overwhelmed by the map data size. We had to pivot from a continuous stream to a chunked, acknowledged software flow control method to prevent data loss, which added significant complexity to the firmware.

5.3 Future Work

1. **Enhance Hardware and Sensing Capabilities:** Optimize omnidirectional wheels for consistent friction, enable 3D Wi-Fi mapping, and upgrade LiDAR systems to detect light-absorbing obstacles by adding cameras.
2. **Expand Environmental Validation:** Transition from controlled testing to complex, unpredictable real-world environments to rigorously stress-test navigation algorithms.

5.4 Ethical Considerations

The primary ethical concern regarding this project involves data privacy and the potential for malicious use. In accordance with the IEEE Code of Ethics (Sections I.1, IX), we are required to avoid harm and respect the privacy of others [15]. Since the device collects environmental data, specifically LiDAR maps and Wi-Fi RSSI logs, there is a risk of infringing on privacy or surreptitious mapping if the device is used improperly. We mitigate this by strictly limiting testing to controlled lab environments with consent, ensuring no personally identifiable information is associated with the data, and securely deleting logs after analysis. Additionally, to prevent the hardware from being used for network intrusion or "snooping," the software is restricted solely to navigation and signal strength mapping purposes.

5.5 Safety Considerations

The most significant safety hazard arises from the high-energy Li-Ion battery, which poses fire risks if damaged or improperly charged. We adhere to university battery safety guidelines [15] by using a dedicated balance charger, physically shielding the battery within the chassis to prevent impact damage, and incorporating short-circuit protection fuses on the PCB. We also must consider the mechanical risks of an autonomous vehicle colliding with people or property. To address this, we have software-limited the car’s speed to a walking pace and implemented a remote emergency stop mechanism. Finally, to ensure eye safety regarding the laser mapping system, we utilize a certified Class 1 LiDAR unit, which complies with IEC 60825-1 standards for safe operation [3].

Appendix A

A.1 Subsystem Requirements and Verification Tables

A.1.1 Base Subsystem

Requirements	Verifications
1. Must provide a graphical user interface (GUI) for manual control of the RC car.	- Launch the GUI on the host computer. - Press the on-screen controls for forward, backward, and lateral movements. - Confirm the RC car responds appropriately to each command.
2. Must be able to establish and maintain a stable Bluetooth connection with the ESP32	- From the GUI, initiate a Bluetooth pairing sequence with the car's ESP32. - Confirm that a stable connection is established and acknowledged in the GUI. - Maintain the connection without dropouts for a continuous 5-minute period while sending manual control commands.
3. Must be able to send a distinct command to initiate the autonomous mapping phase.	- After establishing a connection, click the "Begin Autonomous Mapping" button in the GUI. - Verify that the car stops responding to manual control inputs. - Confirm that the car begins to execute its autonomous path-planning algorithm.
4. Must be able to receive and parse the final (x, y, RSSI) data structure from the car.	- After the car completes its autonomous run, it will transmit the collected data. - Monitor the host computer to confirm the receipt of a data packet. - Verify that the received data structure can be correctly parsed into a list of coordinates and their corresponding signal strength values.
5. Must render the received data as a visual heat map of the mapped area	- Upon successful parsing of the (x, y, RSSI) data, observe the GUI. - Confirm that a 2D plot is generated. - Verify that the plot is color-coded to represent Wi-Fi signal strength at different locations, forming a complete heat map.

Table 2: Base Subsystem Requirements and Verification

A.1.2 Control Subsystem

Requirements	Verification
1. The ESP32 must establish a Bluetooth connection with the host computer to receive manual control commands and transmit collected Wi-Fi map data.	- Write a test script on the host computer to send a specific command string (e.g., "FORWARD"). - Use a serial monitor connected to the ESP32 to confirm the exact string is received. - Hardcode a sample data array of (x, y, RSSI) tuples on the ESP32 and program it to transmit this data upon receiving a "SEND" command. - Verify that the host computer successfully receives the complete, unmodified data array.
2. The Raspberry Pi must process incoming LiDAR data and successfully execute a SLAM algorithm to generate an accurate 2D map of its environment.	- Place the vehicle in a room with known dimensions (e.g., 4m x 5m). - Manually drive the vehicle around the perimeter of the room. - Confirm that the generated map is topologically correct subject to the constraints of the LiDAR sensor (i.e. we cannot detect obstacles that absorb light rays).
3. The ESP32 must translate high-level directional commands into the appropriate PWM signals for the motor driver to achieve forward, backward, and lateral movements.	- Place the vehicle on a stand, allowing the wheels to spin freely. - Send a "forward" command from the host GUI and visually confirm that all four wheels spin in the correct direction for forward motion. - Repeat the test for "backward," "strafe left," and "strafe right" commands, verifying correct wheel behavior for each. - Probe the PWM output pins of the ESP32 with an oscilloscope to confirm that signals are present and their duty cycles change in response to different commands.
4. The subsystem must accurately sample the Wi-Fi RSSI and associate each measurement with the vehicle's (x, y) coordinates provided by the SLAM algorithm.	- Manually position the vehicle at a known starting coordinate (e.g., (0,0)) in the map. - Trigger a single data capture. Log the resulting data point (x, y, RSSI) to the serial monitor. - Verify that the logged x and y coordinates are within a reasonable tolerance (e.g., +/- 50cm) of the known position. - Verify that the logged RSSI is a valid integer (e.g., between -90 and -30 dBm).

Table 3: Control Subsystem Requirements and Verification

A.1.3 Sensing Subsystem

Requirements	Verification
1. The subsystem must be supplied with a stable 5.0V +/- 0.1V.	- Ensure the LiDAR is connected to the 5.0V output from the Drivetrain & Power Subsystem's voltage regulator. - Probe the LiDAR's VCC, VM, and GND pins with a multimeter. - Confirm the measured voltage is stable and within the 4.9V to 5.1V range during operation.
2. The LiDAR sensor must provide a continuous 360-degree, 2D scan of its environment.	- Power on the sensor and interface with it from a host computer. This was verified in the very beginning of the project before connecting it to the PCB by using the laptop application it came with. This requirement was also verified once it was on the PCB by printing out the data points that it had collected to the screen. - Using visualization software, place a distinct object at known angles (e.g., 0°, 90°, 180°, 270°) relative to the sensor. - Confirm that the visualization displays the object at the correct angular positions and that the data stream covers the full 360-degree range without gaps.
3. The sensor's measurement range must be at least 8 meters with reasonable accuracy.	- In a long hallway, place a flat object (e.g., a large box) at a distance of 8 meters from the sensor, measured with a tape measure. - Check the sensor's output data to confirm it registers the object's presence at the correct distance. - Verify that the reported distance is within the accuracy tolerance specified by the manufacturer's datasheet (e.g., ±5 cm).
4. The sensor must provide data at a rate sufficient for the SLAM algorithm to perform real-time tracking (minimum 2.5 Hz).	- Write a test script on the Raspberry Pi to read and timestamp incoming data packets from the LiDAR over the UART connection. - Calculate the frequency of complete 360-degree scans over a 10-second interval. - Confirm that the average scan rate is consistently at or above 2.5 Hz.

Table 4: Sensing Subsystem Requirements and Verification

A.1.4 Drivetrain & Power Subsystem

Requirements	Verification
1. The Li-Ion battery must provide a nominal voltage between 7.4V and 11.1V (2S or 3S) and have sufficient capacity to power the entire system (Raspberry Pi 4, ESP32-S3, two TB6612FNG drivers, four TT motors, and LiDAR sensor) for at least 15 minutes of continuous operation.	- Fully charge the Li-Ion battery and connect it to the system. - Run the robot in a continuous operational mode, including driving the motors and running all processing units. - Time the duration from the start of the test until the battery's voltage drops to its safe cutoff level (e.g., ~3.3V per cell). - Confirm the operational time meets or exceeds 15 minutes.
2. The 5V voltage regulator must supply a stable 5V (±0.2V) to the Raspberry Pi 4 and the logic inputs of the two TB6612FNG motor controllers, even under motor load.	- With the system fully powered on, probe the 5V and GND test points corresponding to the regulator's output with a multimeter. - Confirm the measured voltage is in the 5.0V (±0.1V) range. - Connect an oscilloscope to the 5V rail to observe voltage stability and ripple while starting and stopping all four motors simultaneously.
3. The 3.3V voltage regulator must supply a stable 3.3V (±0.1V) to the ESP32-S3 microcontroller and the LiDAR sensor.	- With the system powered on, probe the 3.3V and GND test points corresponding to the regulator's output with a multimeter. - Confirm the measured voltage is in the 3.3V (±0.1V) range. - Connect an oscilloscope to the 3.3V rail to ensure there is no significant voltage drop or noise when other components are active.
4. The two TB6612FNG motor controllers must correctly interpret PWM signals from the ESP32-S3 and supply sufficient current (up to 1.2A continuous per channel) to drive four TT motors simultaneously.	- Write a test firmware for the ESP32-S3 to generate PWM signals of varying duty cycles (e.g., 25%, 50%, 100%). - Probe output channels of the motor controllers with an oscilloscope to verify that

	the output matches the commanded PWM signal. - Connect an ammeter in series with one of the motors and command it to run at full speed. Confirm current draw is below 1.2A. - Use an infrared thermometer to measure the temperature of the TB6612FNG ICs after 5 minutes of continuous motor operation to ensure they are not overheating.
5. The drivetrain, consisting of four TT motors and omnidirectional wheels, must be capable of executing forward, backward, lateral (strafe), and rotational movements in response to commands.	- Develop a simple test program that allows for sending specific movement commands - Command the robot to move forward 1 meter and verify it moves in a straight line. - Command the robot to move backward 1 meter and verify it moves in a straight line. - Command the robot to strafe left for 1 meter and verify it moves laterally with minimal rotation.

Table 5: Drivetrain & Power Subsystem Requirements and Verification

A.2 PCB Layout

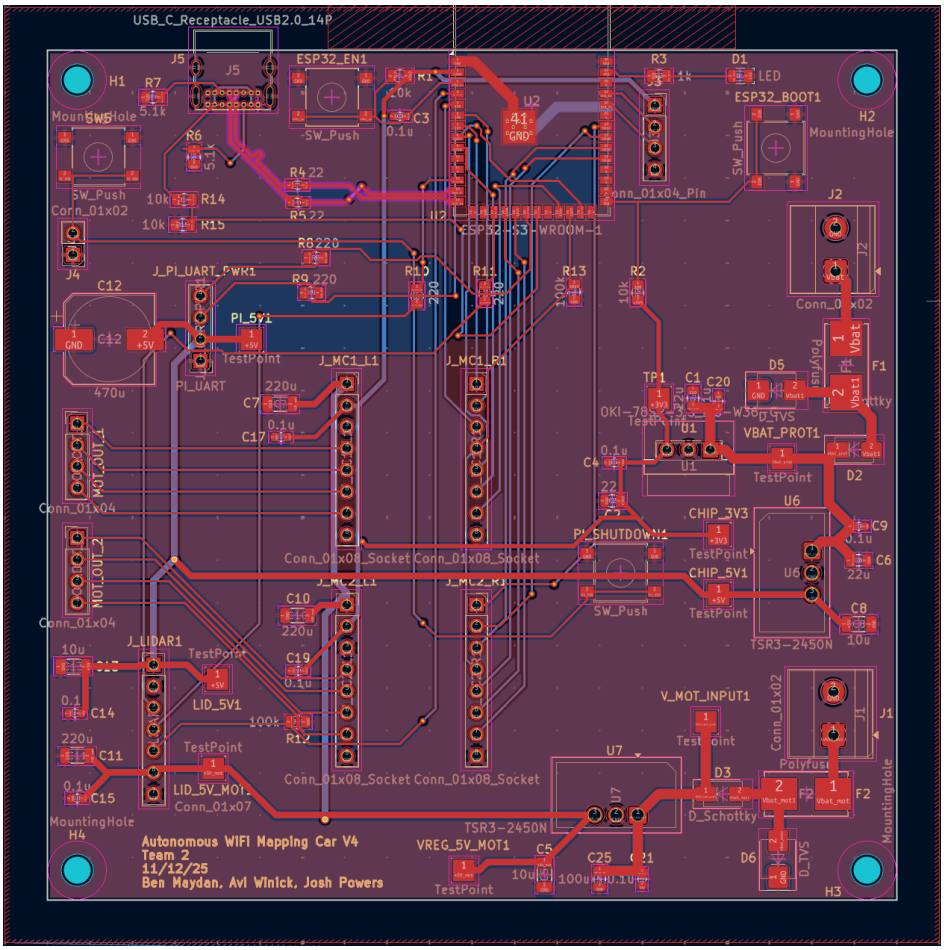

Figure 10: Final PCB Design

References

The authors acknowledge that Google's Gemini and OpenAI's ChatGPT were used for inspiration and to help flesh out initial ideas for this project.

[1] Haptic Networks. (n.d.). Common Mistakes In Wi-Fi Network Design (And How To Avoid Them). Haptic Networks. Retrieved September 13, 2025, from https://haptic-networks.com/wifi/common-mistakes-in-wifi-network-design-and-how-to-avoid-them/

[2] Cisco Systems, “Understand Site Survey Guidelines for WLAN Deployment,” Cisco Support Documentation, updated Nov. 14, 2023. https://www.cisco.com/c/en/us/support/docs/wireless/5500-series-wireless-controllers/116057-site-survey-guidelines-wlan-00.html

[3] Google, Gemini. [Online]. Available: https://gemini.google.com. Accessed: Sept. 19, 2025.

[4] The Grainger College of Engineering. (n.d.). *Computer Engineering*. University of Illinois at Urbana-Champaign. Retrieved October 13, 2025.

[5] The Grainger College of Engineering. (n.d.). *Electrical Engineering*. University of Illinois at Urbana-Champaign. Retrieved October 13, 2025.

[6] University of Illinois at Urbana-Champaign. (n.d.). *Service Rates*. Retrieved October 13, 2025.

[7] Santos, Rui, and Sara Santos. “ESP32 Bluetooth Low Energy (BLE) on Arduino IDE.” Random Nerd Tutorials, 11 Aug. 2024, randomnerdtutorials.com/esp32-bluetooth-low-energy-ble-arduino-ide/. Accessed 12 Oct. 2025.

[8] Santos, Rui, and Sara Santos. “ESP32 UART Communication (Serial): Set Pins, Interfaces, Send and Receive Data (Arduino IDE).” Random Nerd Tutorials, 5 Aug. 2024, randomnerdtutorials.com/esp32-uart-communication-serial-arduino/. Accessed 12 Oct. 2025.

[9] Toshiba. (n.d.). TB6612FNG Dual Motor Driver Carrier Datasheet. SparkFun. Retrieved October 12, 2025, from https://cdn.sparkfun.com/datasheets/Robotics/TB6612FNG.pdf

[10] Pi 4 maximum power consumption. (n.d.). Raspberry Pi Stack Exchange. Retrieved October 12, 2025, from https://raspberrypi.stackexchange.com/questions/114239/pi-4-maximum-power-consumption

[11] Yakubov, Alex. “How Much Does a Wireless Site Survey Cost and Is It Worth It for My...” *Made by WiFi*, Made By WiFi, 24 Jan. 2019,

www.madebywifi.com/blog/how-much-does-a-wireless-site-survey-cost-and-is-it-worth-it-for-my-business/. Accessed 7 Dec. 2025.

[12] Slanderkin. “Issue with Modified A1 Example Code.” *GitHub*, 14 Nov. 2020, github.com/simondlevy/BreezySLAM/issues/76. Accessed 7 Dec. 2025.

[13] Slamtec. “RPLIDAR-A1 360° Laser Range Scanner.” *Slamtec*, 2016, www.slamtec.com/en/Lidar/A1. Accessed 7 Dec. 2025.

[14] kaiaai. “GitHub - Kaiaai/LDS: Arduino LiDAR Library Supporting YDLIDAR X2/X3/X4, RPLIDAR A1, Xiaomi LDS02RR, Neato XV11, LD14P, CAMSENSE X1, Delta-2A/2B/2G.” *GitHub*, 2024, github.com/kaiaai/LDS.

[15] Institute of Electrical and Electronics Engineers (IEEE), “IEEE Code of Ethics,” https://www.ieee.org/about/corporate/governance/p7-8.html, 2024, accessed: Feb. 15, 2024.