ECE 56800 Research Project: RC Car Tracking

Parth Malhotra
College of Engineering
Purdue University
West Lafayette, USA
malhot82@purdue.edu

Kienen Nordlie
School of Interdisciplinary Engineering
Purdue University
West Lafayette, USA
knordlie@purdue.edu

Mayank Pratap
College of Engineering
Purdue University
West Lafayette, USA
pratapm@purdue.edu

***Abstract*—This project presents the development of an autonomous RC car capable of tracking and following a Bluetooth Low Energy (BLE) beacon using Received Signal Strength Indicator (RSSI) as the sole tracking signal. The system relies on real-time RSSI feedback to infer the relative proximity of the target beacon and continuously adjusts its motion to approach it. An ultrasonic sensor is integrated for obstacle detection and reactive avoidance, enabling safe navigation in dynamic environments. The goal is to demonstrate that RSSI-based tracking can be effectively utilized in low-cost robotics platforms without reliance on visual systems. The car is evaluated based on its ability to consistently move toward the beacon, maintain a stable range, and avoid collisions. Experimental results highlight the feasibility of RSSI-only tracking and provide insights into the practical challenges posed by RSSI signal noise and environmental interference. The complete source code is available at https://github.com/Mayankpr04/RC_Follower**

***Index Terms*—Bluetooth Low Energy (BLE), Received Signal Strength Indicator (RSSI), Tracking, Sensors, Architecture**

I. Introduction

A. Background and Motivation

The field of autonomous systems has witnessed rapid advancements in recent years, largely driven by sophisticated sensor technologies such as LiDAR, computer vision, and mmWave radar. While highly effective, these systems remain cost-prohibitive for low-budget or educational robotics projects. This project investigates the potential of using affordable alternatives—specifically, Bluetooth Low Energy (BLE) beacons and ultrasonic sensors—for autonomous object tracking and navigation.

A central challenge in this domain lies in achieving accurate localization and object following using minimal hardware. Ultrasonic sensors offer reliable short-range distance measurement, while BLE-based Received Signal Strength Indicator (RSSI) values can serve as a proxy for proximity. However, RSSI is notoriously noisy and environment-dependent. This project aims to assess the viability of using RSSI as the sole tracking input, supported by ultrasonic sensors for collision avoidance.

B. Bluetooth Low Energy and Received Signal Strength Indicator Overview

BLE is a wireless communication protocol optimized for low-power, short-range transmission. BLE beacons broadcast advertisement packets periodically without requiring pairing, making them ideal for simple proximity detection applications. These packets typically contain the device's MAC address, optional payload data, transmit (TX) power calibration values, and flags used by receiving devices. These packets of data can be transmitted at up to ranges of 100m depending on the transmit power of the beacon

RSSI represents the measured power of a received radio signal and is expressed in decibel-milliwatts (dBm). For BLE devices, RSSI values are always negative—closer to 0 indicates a stronger signal. For instance, an RSSI of -40 dBm suggests a strong signal, while -90 dBm reflects a weak signal. Since RSSI is measured on a logarithmic scale, even small changes (e.g., -70 dBm to -67 dBm) represent significant variations in signal strength.

The relationship between RSSI and distance can be approximated using the path loss model:

$$RSSI(d) = RSSI_0 - 10 \cdot n \cdot \log_{10}\left(\frac{d}{d_0}\right)$$

Where:

- $RSSI(d)$ is the measured signal strength at distance d
- $RSSI_0$ is the RSSI at reference distance d_0
- n is the path-loss exponent, which depends on the environment
- d is the distance between the transmitter and receiver in meters

In practice, BLE-based localization using RSSI is susceptible to multipath interference, signal attenuation, and reflection from objects in the environment. Despite these challenges, BLE remains a cost-effective option for proximity-based tracking in autonomous systems.

C. Project Objectives

The primary goal of this project is to develop a self-driving RC car that can:

- **Autonomously navigate to a target object** using BLE-based tracking.
- **Maintain a fixed distance from an object** using ultrasonic or BLE sensors.
- **Optimize sensor placement and selection** to improve tracking accuracy.

The system is implemented using an Arduino Uno R3, which controls the car's motion and integrates sensor data. An ESP32 module is used to scan and relay RSSI values to

the Arduino via serial communication. The Arduino processes this information in real time to estimate beacon proximity and issue motion commands.

D. Potential Applications

The ability to track objects autonomously using budget sensors has multiple **real-world applications** across various domains. Some of the most relevant use cases include:

- Autonomous Driving: Low-cost vehicle or pedestrian tracking for educational or small-scale self-driving applications.
- Manufacturing and Warehousing: Asset tracking and automated sorting in logistics.
- Surveillance and Healthcare: Indoor tracking and localization for security monitoring or elderly care.

By comparing the **sensor accuracy, tracking performance, and environmental adaptability**, this project will contribute valuable insights into the feasibility of **budget-friendly object tracking solutions** for real-world autonomous applications.

II. State-of-the-Art Research

A. Ultrasonic Sensors in Autonomous Navigation

Ultrasonic sensors are commonly used in budget-friendly autonomous systems for obstacle detection and distance measurement due to their affordability and reliability in short-range applications. These sensors operate by emitting high-frequency sound waves and measuring the time-of-flight (ToF) of the returning echo to estimate distance. A study on sensing technologies for indoor autonomous mobile robots highlights their effectiveness in simple navigation tasks. However, angle limitations, inconsistent reflections, and environmental interference remain challenges that affect their precision. [1]

B. Bluetooth Low Energy Beacons and Received Signal Strength-Based Indoor Localization

BLE beacons are commonly used for indoor localization in environments where GPS is unavailable. These systems estimate proximity based on the RSSI, but accuracy is limited by signal attenuation, interference, and multipath propagation. Recent work proposed a low-cost RSSI-based positioning system for UAVs that used differential distance correction techniques to enhance tracking reliability and mitigate noise-related inaccuracies. [2]

C. Sensor Fusion for Robust Tracking

To overcome the individual weaknesses of ultrasonic and BLE sensors, researchers have explored sensor fusion algorithms that combine multiple data sources for improved tracking accuracy. A smart parking system integrated BLE and ultrasonic sensors to enable real-time vehicle tracking, utilizing RSSI-based triangulation for long-range localization and ultrasonic measurements for precise short-range detection. This approach demonstrates the potential of multi-sensor systems in budget-constrained environments. [3]

D. Algorithmic Approaches to Sensor Fusion

A critical area of research focuses on developing algorithms that effectively merge data from multiple sensors to enhance tracking performance. Some notable techniques include:

- **Kalman Filters:** Used for predicting and correcting sensor readings in dynamic tracking environments. A Kalman Filter can smooth RSSI fluctuations and improve distance estimation by being resistant to the inherent noise of RSSI values.
- **Particle Filters:** Applied when the system has nonlinear motion and uncertainty in sensor data. Particle filtering is useful for tracking moving targets in cluttered environments.
- **Weighted Sensor Fusion:** A technique that assigns different weights to each sensor based on environmental conditions and confidence levels to dynamically adjust tracking accuracy.
- **Neural Network-Based Fusion:** Machine learning models trained on RSSI and ultrasonic distance data can be used to learn complex sensor dependencies and improve localization robustness.

Although recent studies have shown that neural networks can reduce RSSI noise and improve real-time localization accuracy, this project does not utilize machine learning approaches as the 2KB memory of the Arduino Uno R3 would not be sufficient for implementation. The final architecture using the ESP32 to preprocess the RSSI values could work, but neural networks are not the focus of this project. Therefore, these techniques are not explored in detail in this work.

III. Methodology

A. System Architecture and Hardware Components

The hardware system comprises two primary processing units: an ESP32 Feather V2 and an Arduino Uno R3, along with multiple peripheral components for actuation and sensing. The ESP32 is responsible for scanning BLE signals and extracting RSSI values, while the Arduino handles motion control and obstacle avoidance decision making.

The car platform was built using a standard Arduino car kit that includes an Arduino Sensor Shield V5, L298N motor driver, four DC motors with wheels, a battery holder, and a chassis frame. An HC-SR04 ultrasonic sensor mounted on a servo was used to perform obstacle detection with a sweeping field of view with an experimental response range of approximately 30cm. The BLE beacon used was a BlueCharm BLE device, which periodically broadcasts advertisement packets containing the TX power value and MAC address.

Originally, the design aimed to use an HM-10 BLE module for direct Arduino-based RSSI scanning. However, due to firmware limitations (modules running firmware versions below v707), the AT commands required for RSSI retrieval were not supported. As flashing third-party firmware risked rendering the modules useless and the time constraints of the project, a design change was made to offload BLE scanning to the ESP32.

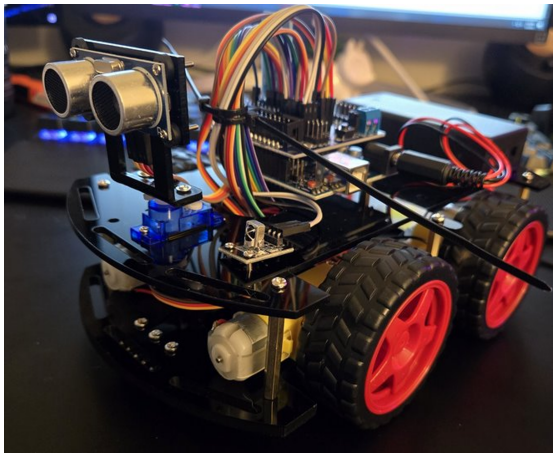

Fig. 1. Assembled Car kit

The ESP32 runs MicroPython and is programmed via the Thonny IDE, while the Arduino is programmed using the Arduino IDE. The ESP32 and Arduino communicate via the UART protocol. The ESP32 transmits filtered RSSI values to the Arduino, which uses them to make movement decisions.

The diagram in Figure 2 shows the interaction between all major components. The ESP32 receives BLE signals, processes them, and communicates results to the Arduino. The Arduino takes these RSSI values and, along with ultrasonic input for obstacle detection, controls the motor driver to actuate the motors.

B. Software and Programming Environment

The software system was developed using two separate environments to accommodate the two microcontroller platforms involved: the ESP32 Feather V2 and the Arduino Uno R3.

The **ESP32 Feather V2** was programmed using **MicroPython** via the **Thonny IDE**. MicroPython was selected for its simplicity in handling BLE operations and asynchronous scanning tasks. The ESP32 was responsible for continuously scanning BLE advertisement packets using the Generic Access Profile (GAP) protocol. From these packets, the RSSI values corresponding to the target beacon were extracted and filtered using signal preprocessing techniques (discussed in a later section). The processed RSSI values were then transmitted to the Arduino Uno via the UART serial protocol.

The **Arduino Uno R3** was programmed using the **Arduino IDE**, with the code written in standard C/C++ syntax. The Arduino handled all real-time control decisions, including:

- Receiving filtered RSSI data over UART from the ESP32;
- Executing the control algorithm to interpret RSSI changes;
- Performing obstacle detection using the HC-SR04 ultrasonic sensor mounted on a servo;
- Controlling the direction and speed of the motors via PWM signals through the L298N motor driver.

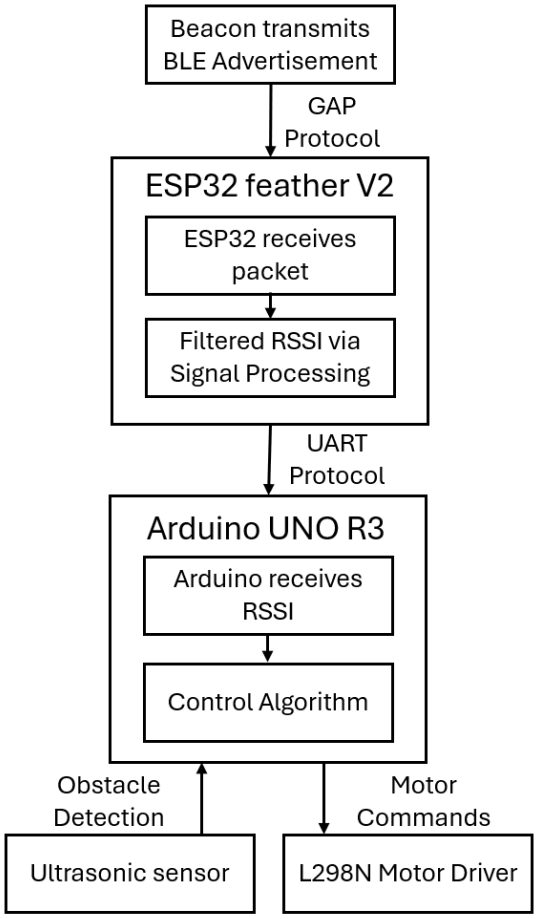

Fig. 2. System architecture showing data flow between BLE beacon, ESP32, Arduino, and peripheral components.

Both devices communicated using **UART at a baud rate of 9600**, with the ESP32 acting as the transmitter and the Arduino as the receiver. A custom message format was implemented to ensure proper parsing of RSSI values on the Arduino side.

This separation of processing responsibilities—signal acquisition and filtering on the ESP32, and control logic execution on the Arduino—allowed for modular development and easier debugging.

TABLE I
SOFTWARE STACK AND TOOLCHAIN

Component	**Language/IDE**	**Function**
ESP32 Feather V2	MicroPython / Thonny	BLE Scanning, Filtering
Arduino Uno R3	Arduino C / Arduino IDE	Control Logic Motor Commands

C. RSSI Pre-Processing and Filtering

Raw RSSI values obtained from BLE advertisements are inherently noisy and unreliable due to various environmental factors, including:

- Multipath interference caused by signal reflections from walls and objects,
- Signal attenuation due to obstacles or body shadowing,
- Variability introduced by antenna orientation and device movement.

To improve the reliability of distance estimation, we implemented two filtering strategies on the ESP32 using MicroPython before transmitting the data to the Arduino.

1) Outlier Mean Filter: In the initial approach, the ESP32 collected a buffer of RSSI readings over a fixed interval (typically 3–5 seconds). Outlier values, defined as those falling beyond a statistical threshold (e.g., beyond 1.5 times the interquartile range), were removed. The remaining values were averaged to produce a stable RSSI estimate. This approach provided basic noise reduction and prevented erratic directional changes.

While this method improved short-term decisions, it introduced latency due to the required window size for filtering.

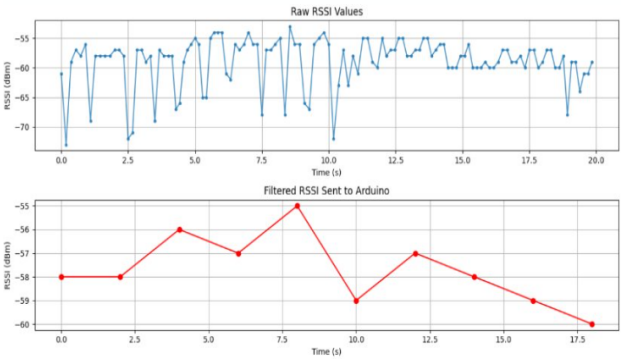

Fig. 3. Raw vs Filtered RSSI values at a fixed distance of 35cm

2) Kalman Filter: To improve responsiveness while maintaining filtering strength, a Kalman filter was adopted. The Kalman filter models RSSI as a time-series with Gaussian noise, updating its estimate at each time step using the equation:

$$x_k = x_{k-1} + K_k(z_k - x_{k-1})$$

Where x_k is the estimated RSSI at time step k, z_k is the measured RSSI, and K_k is the Kalman gain.

This method significantly reduced jitter while retaining fast convergence in both static and dynamic tracking scenarios. Three test cases were used to validate its performance:

- **Test 1: Static beacon at 2 feet** — Figure 4
- **Test 2: Static beacon at 24 feet with interference** — Figure 5
- **Test 3: Moving beacon target** — Figure 6

Each case shows raw RSSI (orange), mean-filtered RSSI (blue), and Kalman-filtered RSSI (green). Kalman filtering consistently provided smoother and more accurate tracking performance.

Fig. 4. RSSI filtering at 2 ft: raw vs mean vs Kalman filtered

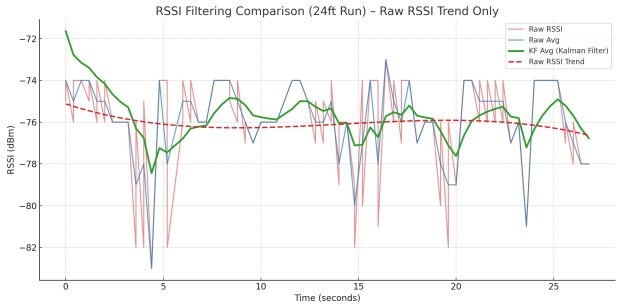

Fig. 5. RSSI filtering at 24 ft with interference

Overall, the Kalman filter dramatically improved signal stability, enabling the control algorithm to make more consistent directional decisions, even in environments with noise and multipath interference.

D. Control Algorithm and Navigation Strategy

The Arduino Uno is responsible for executing the control logic that determines the robot's movement based on RSSI readings and ultrasonic obstacle detection. The algorithm is structured into multiple decision phases:

Phase 1: Obstacle Detection: Before making any movement decisions, the robot checks for immediate obstacles using the front-facing ultrasonic sensor. If an object is detected within 30 cm, it triggers the obstacle avoidance routine.

Phase 2: PID-Based RSSI Control: If the path is clear, the robot processes the RSSI value received from the ESP32. A PID controller is used to calculate an error value based on the difference between the current RSSI and a target proximity value. The PID controller components are:

- P: Proportional gain based on immediate RSSI error
- I: Integral of accumulated error over time
- D: Derivative of the change in error

The resulting PID output is used to modulate the PWM motor speed:

$$\text{PWM}_{adjusted} = \text{Base Speed} - |\text{PID}_{sum}|$$

This allows the robot to dynamically slow down as it nears the beacon, preventing overshooting and enabling smooth convergence.

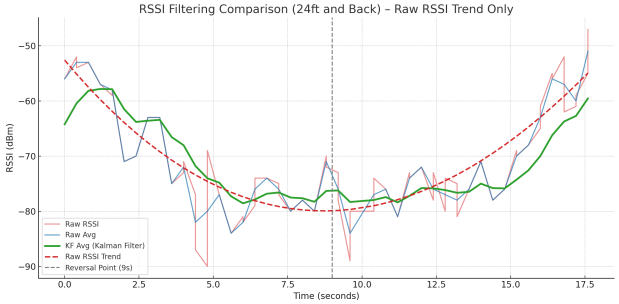

Fig. 6. RSSI filtering during beacon movement, 0-9s is moving away, and 9s onward is moving toward

Phase 3: Directional Scanning: To determine whether the beacon lies to the left or right, the robot uses a servo-mounted ultrasonic sensor to rotate the ESP32 slightly and take RSSI readings at two servo angles:

- 150°: to measure left-side RSSI
- 30°: to measure right-side RSSI

Based on which side has the stronger signal, the robot turns in that direction. This off-center mounting of the ESP32 allows for passive direction estimation using just one RSSI sensor — a key novelty in this system where most other solutions rely on multiple antennae or arrays of modules to calculate an Angle of Arrival (AoA) for direction finding. [4]

Phase 4: Obstacle Avoidance Routine: If an obstacle is encountered mid-motion, the robot stops and scans both left and right using the servo-mounted ultrasonic sensor. The servo rotates to 20° (left) and 20° (right), collecting distance measurements. The robot turns toward the direction with the most clearance. If both sides are clear, it randomly chooses a direction to continue.

This layered logic combining PID control with environmental scanning allows the robot to track the BLE beacon while dynamically avoiding obstacles.

E. Evolution of ESP32 Placement for Directional Sensing

Initially, the ESP32 was fixed at the geometric center of the car chassis. This central mounting was simple to implement, but it introduced ambiguity in determining the direction of the BLE beacon. Since RSSI signals are scalar in nature and omnidirectional at close range, having the sensor in the center led to a circular symmetry problem — multiple possible directions could yield the same RSSI value.

To address this, the ESP32 was repositioned to an **off-center location** on the right side of the chassis. This configuration introduced a passive directional bias: when the car rotated slightly, the ESP32 would experience different RSSI strengths depending on whether it was turning toward or away from the beacon. While this provided better performance than the centered setup, it was still limited by the static nature of the sensor and extreme fluctuations of RSSI values in comparison to the change caused by turning the car.

The final and most effective configuration involved **mounting the ESP32 on an extension attached to a servo motor**,

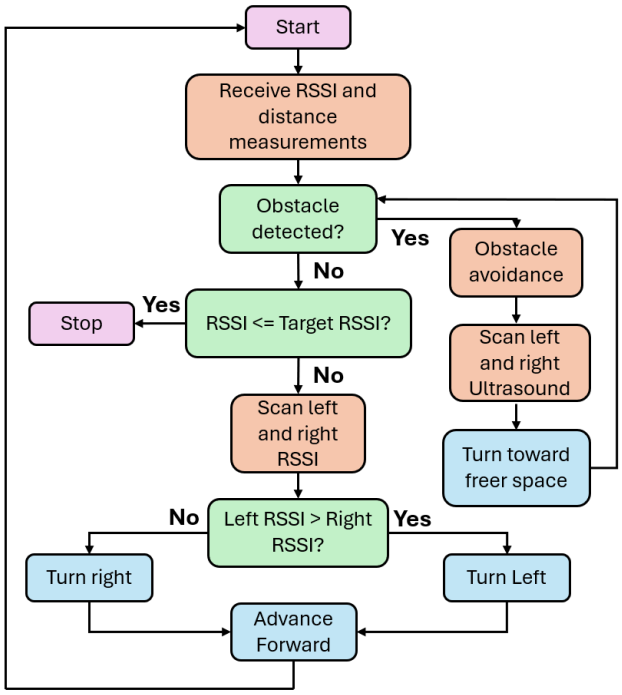

Fig. 7. Flowchart of RSSI-driven PID control and obstacle avoidance logic

enabling it to actively scan the environment. During scanning phases, the servo rotated the ESP32 to predefined angles (e.g., 30° left and 30° right), allowing it to take directional RSSI readings. This effectively mimicked a two-sensor setup using only one BLE receiver, greatly improving directional estimation.

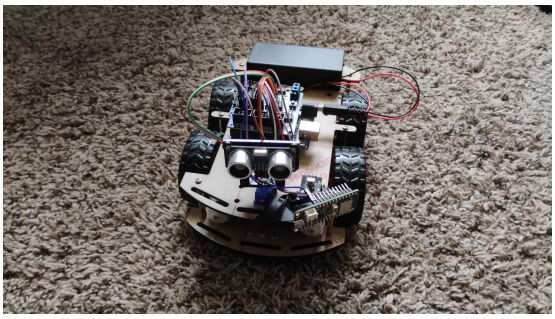

Fig. 8. Setup: ESP32 mounted off-center of car chassis

This evolution of ESP32 placement was a critical aspect of the project's success. It allowed for better estimation of beacon direction without requiring additional hardware or triangulation algorithms. Combined with signal filtering and PID control, this enhancement enabled the robot to make more consistent and correct movement decisions using RSSI alone.

IV. Contribution

At the beginning of the project, the team attempted to assign distinct roles to each member—such as hardware assembly, firmware development, and sensor integration—to streamline

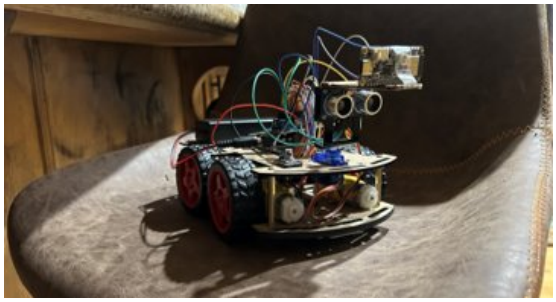

Fig. 9. Final setup: ESP32 mounted on servo extension for directional scanning

development. However, as the project progressed, this rigid division proved inefficient due to integration challenges, hardware inconsistencies across builds, and the complexity of debugging concurrent systems.

In response, the team adopted a more collaborative and iterative workflow. All members worked together to assemble and validate their cars to ensure consistent baselines, and then contributed collectively to each phase of development. This included BLE scanning, RSSI filtering, PID control, obstacle avoidance, and servo-mounted directional sensing.

The team held regular meetings (weekly and as needed) to align on progress, debug system-level issues, and integrate new features. Code was jointly written and reviewed, with each member contributing to testing, tuning, and refining behaviors across multiple iterations.

This adaptive and inclusive approach enabled faster problem resolution and deeper understanding across the team, resulting in a more robust and well-integrated final system that consistently made meaningful progress each week.

V. Evaluation and Results

For all testing procedures, an obstacle course was set up as shown in Figure 10. This course included two cardboard obstacles placed between the RC car's starting point and the beacon, the final position. Each trial began with the RC booting its logic, an initial scan to detect the beacon, and drive sequence to search and track the beacon to within one foot of its resting position. Once an obstacle is detected, the RC car will evaluate the most clear path to unblock itself and continue with the search and track operation. Two configurations of tests were conducted: a 6-second scan cycle (3 seconds scanning to the left and 3 seconds to the right) and a 2-second scan cycle (1 second in each direction). A total of seven tests were performed for each scanning configuration.

The results of these tests are shown in the box plots in Figure 11 and Figure 12, which display the total travel time for the RC car to detect the beacon, navigate around obstacles, and move toward the beacon until it came to a complete stop within one foot of the beacon. As indicated in the data, the 6-second scan cycle was more time efficient by outperforming the 2-second scan cycle in terms of total travel time. While initially counter-intuitive, this shows that meaningful and correct RSSI

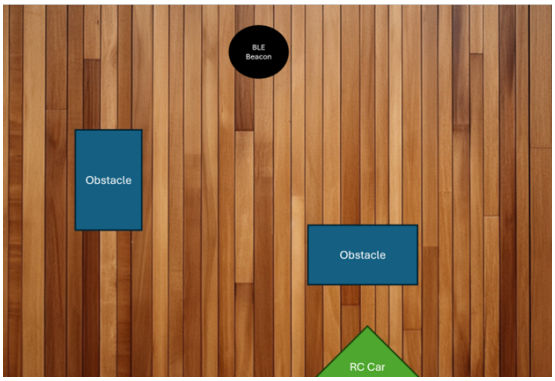

Fig. 10. Final setup: ESP32 mounted on servo extension for directional scanning

measurements are vital to the robot making correct decisions to make progress towards the beacon. On average, the RC car reached its destination significantly faster when it had a longer scanning duration, which shows the longer periodic scans allows for better path planning. Please note that the first and third quartiles are much tighter and precise for the 6s scan cycle shown in Figure 11, when compared to the 2s scan cycle in shown Figure 12. In the test result for the 6s scan cycle, there is one outlier that is marked in red.

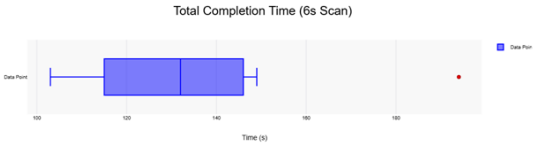

Fig. 11. Final setup: ESP32 mounted on servo extension for directional scanning

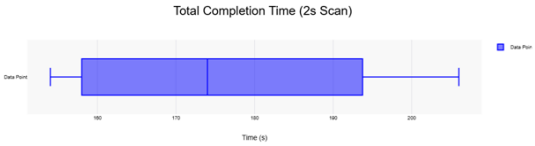

Fig. 12. Final setup: ESP32 mounted on servo extension for directional scanning

Further analysis was performed using an accuracy metric defined as the percentage of wrong turns out of the total number of turns, multiplied by 100. This is a unique approach to measuring the effectiveness of our system, since we do not have any sophisticated path-planning algorithms. This metric, along with the corresponding travel times, is shown in Figure 13 and Figure 13. The results demonstrate that the 6-second scan not only resulted in a lower error percentage, meaning fewer incorrect turns that result in better directionality, but also maintained shorter overall travel times. As a result, this confirms that the extended periodic scanning routine improved both receiving an accurate RSSI reading

and better path direction to the beacon.

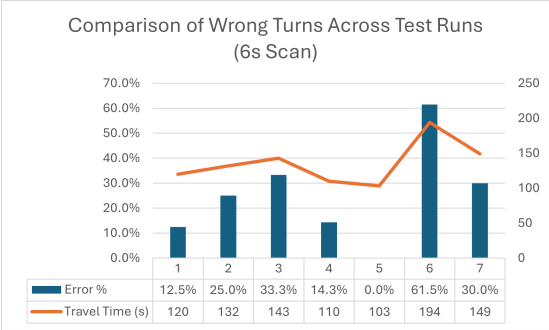

Fig. 13. Final setup: ESP32 mounted on servo extension for directional scanning

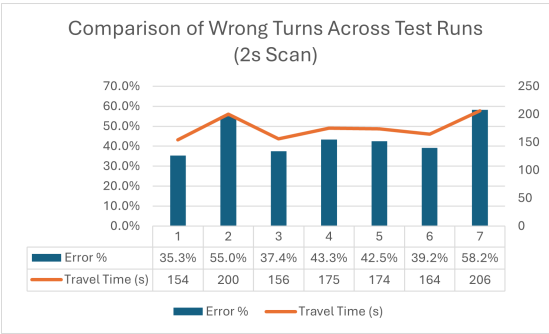

Fig. 14. Final setup: ESP32 mounted on servo extension for directional scanning

VI. Challenges

A. Signal Interference and Multipath Effects

BLE tracking using RSSI is affected by interference from walls, furniture, and other electronic devices signals such as WiFi. Multipath effects, where signals reflect off surfaces before reaching the receiver, can cause fluctuations in RSSI values, leading to inaccurate distance estimations. Methods such as filtering and averaging may be required to improve accuracy.

B. Ultrasonic Sensor Limitations

Ultrasonic sensors work well for short-distance measurement but have limitations. Soft surfaces may absorb sound waves, reducing detection accuracy. The sensors have a limited range and a fixed detection angle, which may cause tracking errors in complex environments.

C. Sensor Fusion and Processing Constraints

Combining BLE and ultrasonic sensor data requires an effective fusion algorithm. BLE signals fluctuate more than ultrasonic readings, so proper data weighting is necessary. Real-time processing on microcontrollers like Arduino or ESP32 may be limited by computational constraints.

VII. Future Work

This project demonstrated that RSSI-based tracking using a single BLE beacon and a microcontroller-based system can enable basic object-following behavior in a mobile robot utilizing a singular BLE receiver. However, several opportunities exist to extend and improve the system in future iterations:

- **Advanced RSSI Processing:** Current filtering techniques (mean and Kalman filters) provide basic noise reduction. More robust signal processing techniques—such as adaptive filters, hysteresis smoothing, or machine learning-based denoising—could further improve reliability in cluttered environments.
- **Path Planning Integration:** The current approach follows a greedy ”move-toward-beacon” behavior. Future systems could incorporate grid-based path planning (e.g., A*, D*) or potential fields to navigate toward the beacon while dynamically avoiding obstacles.
- **ROS-Based Navigation:** Integrating with the Robot Operating System (ROS) would provide access to a modular navigation stack, real-time visualization, SLAM capability, and better scalability for future multi-agent or multi-sensor systems.
- **Directional Sensing via Multi-RSSI Inputs:** Incorporating an additional ESP32 or RSSI receiver on the robot would allow for passive triangulation or gradient-based direction estimation, improving the accuracy of movement decisions without relying on servo-mounted scanning.
- **Hardware Comparisons:** A future version could evaluate upgraded HM-10 BLE modules flashed with v707 firmware, enabling direct RSSI access via Arduino without a separate ESP32. This would allow for a fully Arduino-based implementation and a hardware performance comparison between ESP32 and HM-10 in BLE scanning accuracy and stability.

These enhancements could significantly improve both the robustness and scalability of the system, enabling deployment in more complex and dynamic environments.

References

[1] Y. Liu, S. Wang, Y. Xie, T. Xiong, and M. Wu, “A review of sensing technologies for indoor autonomous mobile robots,” *Sensors*, vol. 24, no. 4, 2024. [Online]. Available: https://www.mdpi.com/1424-8220/24/4/1222

[2] S. Ponte, G. Ariante, A. Greco, and G. Del Core, “Differential positioning with bluetooth low energy (ble) beacons for uas indoor operations: Analysis and results,” *Sensors*, vol. 24, no. 22, 2024. [Online]. Available: https://www.mdpi.com/1424-8220/24/22/7170

[3] S. Jeon and D.-E. Seo, “Smart parking system based on an ultrasonic sensor and bluetooth low energy in the internet of things,” *Journal of System and Management Sciences*, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:234352746

[4] F. Furfari, M. Girolami, F. Mavilia, and P. Barsocchi, “Indoor localization algorithms based on angle of arrival with a benchmark comparison,” *Ad Hoc Networks*, vol. 166, p. 103691, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1570870524003020