Turf Upkeep and Robotic Field-care (TURF)

Department of Electrical and Computer Engineering
University of Central Florida
Senior Design Group: 20

Aryan Tulsi, Samuel Eisert, and Andrew Koch

***Abstract*—This work presents TURF, a low-cost autonomous wheeled robot designed to operate within a bounded outdoor environment using Ultra-Wideband (UWB) localization. The system is motivated by the need for accessible alternatives to expensive commercial robotic platforms, such as autonomous lawn mowers, while maintaining reliable real-time performance on resource-constrained hardware.**

The TURF system employs a distributed architecture consisting of multiple fixed UWB anchor nodes and a mobile robot platform. Anchor nodes perform range measurements and compute position estimates through trilateration, transmitting results to the robot via a lightweight wireless protocol. The robot integrates these measurements with onboard odometry using a real-time embedded firmware framework, enabling continuous pose estimation and navigation.

A key challenge addressed in this work is the reliability of UWB-based localization in noisy and dynamic environments. To mitigate measurement noise and outliers, the system incorporates temporal filtering, outlier rejection, and weighted averaging of position estimates. Additional calibration techniques are implemented to compensate for hardware-induced timing offsets in UWB modules, improving ranging accuracy.

The system is implemented using custom-designed PCBs with ESP32 microcontrollers and UWB transceivers, emphasizing cost efficiency. The platform can achieve stable localization and responsive motion control suitable for autonomous operation in a predefined area.

This project highlights the feasibility of building scalable, low-cost autonomous systems by distributing computation and leveraging sensor fusion, providing a flexible foundation for future robotics applications beyond lawn maintenance.

I. Introduction

Autonomous mobile robots are increasingly being deployed in applications requiring operation within bounded outdoor environments, such as lawn maintenance, inspection, and agricultural monitoring. However, many existing commercial systems rely on expensive sensing modalities, including LiDAR or vision-based localization, which significantly increase system cost, computational requirements, and overall complexity. These constraints limit accessibility and scalability, particularly for hobbyist, educational, and low-cost deployment scenarios. Ultra-Wideband (UWB) localization presents a compelling alternative for short-range positioning due to its relatively low cost, high ranging accuracy, and independence from environmental lighting conditions [2]. By leveraging time-of-flight measurements between fixed anchors and a mobile platform, UWB systems can estimate position within a bounded area without requiring external infrastructure such as navigation satellite connectivity. This makes UWB particularly well-suited for localized autonomous systems operating in predefined environments.

This work presents TURF, a low-cost autonomous wheeled robot lawn mower designed to demonstrate that reliable autonomous operation can be achieved using affordable, widely available hardware. The system employs a distributed architecture consisting of multiple UWB anchor nodes and a mobile robot platform. Anchor nodes perform ranging and position estimation, transmitting results to the robot, which integrates these measurements with onboard odometry through a real-time embedded framework. Rather than relying on high-end sensors or computationally intensive algorithms, the TURF system emphasizes simplicity and efficiency. Lightweight filtering and motion-constrained measurement validation are used to improve the consistency of position estimates, while calibration techniques compensate for hardware variability across UWB modules. These approaches enable stable operation without significantly increasing system cost or complexity. The system is implemented using custom-designed printed circuit boards integrating ESP32 microcontrollers and UWB transceivers, prioritizing modularity, scalability, and cost efficiency. By distributing computation and leveraging simple sensor fusion techniques, this work demonstrates that low-cost embedded systems can achieve reliable autonomous navigation within a constrained environment. TURF is intended not only as a functional lawn care platform, but also as a flexible and extensible foundation for future low-cost robotics applications.

II. System Hardware Components

A. Ultra-Wideband Transceiver-Receiver

The system utilizes DWM1000 UWB modules to perform precise distance measurements between fixed anchor nodes and the mobile robot. These modules operate using time-of-flight (ToF) ranging, enabling accurate distance estimation without reliance on external infrastructure such as Global Navigation Satellite Systems.

Multiple UWB anchors are deployed at known positions within the environment. By measuring distances between the robot and these anchors, the system computes the robot's position through trilateration. To reduce computational load on the robot, position estimation is performed on a designated anchor node and transmitted wirelessly to the mobile platform.

The DWM1000 was selected due to its relatively low cost, high ranging accuracy, suitability for short-range localization, and abundant open-source resources, making it an effective choice for constrained-area autonomous systems.

B. Microcontroller

The TURF system is built around the ESP32 WROVER microcontroller, which serves as the central processing unit for both the anchor nodes and the mobile robot. The ESP32 provides a dual-core processor, integrated Wi-Fi, and support for low-latency wireless communication protocols such as ESP-NOW.

The ESP-32 WROVER is used on both ther anchors and robot. On the anchor nodes, the ESP32 interfaces with the UWB modules to perform ranging operations in unison with the on-robot system, and is used to calculate position estimates which are then sent to the robot using ESP-NOW, a low latency WiFi protocol. On the robot, the ESP-32 also communicates with the UWB module to facilitate ranging and accepts ESP-NOW messages for utilization. The on-robot ESP32 also manages sensor fusion and motor control.

The ESP32 WROVER was chosen for its low cost, and sizable memory and non-volatile storage, along with native real-time operating system support which allows the system to handle multiple concurrent tasks.

C. Ultrasonic Range Finder

An HC-SR04 ultrasonic rangefinder is used for short-range obstacle detection on the mobile platform. The sensor measures distance by emitting ultrasonic pulses and calculating the time taken for the echo to return after reflecting off nearby objects.

This sensor provides a simple and cost-effective method for detecting obstacles within a limited range, enabling basic collision avoidance behavior. When an obstacle is detected within a predefined threshold distance, the robot halts until the obstacle is cleared.

Although ultrasonic sensing lacks the resolution and robustness of more advanced sensors such as LiDAR, it aligns with the system's goal of demonstrating practical autonomous behavior using inexpensive and accessible components.

D. Motor and Motor Drivers

The TURF system utilizes two 25GA-370 DC geared motors as its primary components for movement. Each motor operates at a nominal voltage of 12V and exhibits a stall current of 2A.

Motor control is implemented using the DRV8874 motor driver, a high-efficiency H-bridge driver capable of delivering adequate current while integrating overcurrent, thermal, and undervoltage protection features. To ensure stable operation, bulk and bypass capacitance are placed across the motor supply (VM) to mitigate voltage transients caused by rapid current changes during switching. This helps reduce electrical noise and prevents voltage drops that could affect driver performance.

Table 3. PH/EN Control Mode

nSLEEP	EN	PH	OUT1	OUT2	DESCRIPTION
0	X	X	Hi-Z	Hi-Z	Sleep, (H-Bridge Hi-Z)
1	0	X	L	L	Brake, (Low-Side Slow Decay)
1	1	0	L	H	Reverse (OUT2 → OUT1)
1	1	1	H	L	Forward (OUT1 → OUT2)

Fig. 1. PH/EN control logic for the DRV8874 motor driver, showing the relationship between nSLEEP, EN, and PH inputs and the resulting motor output states (OUT1, OUT2), including operating modes such as sleep, brake, forward, and reverse.

The system controls speed and direction using Pulse-Width Modulation (PWM) signals from the ESP32 in conjunction with Phase/Enable (PH/EN) pins on the driver. The PWM signals are generated by the ESP32 using hardware timers, allowing motor speed to be controlled through duty cycle variation. By adjusting the duty cycle, the average voltage applied to the motor is regulated, enabling smooth and efficient speed control. The Phase pin determines the direction of motor rotation (forward or reverse), while the Enable pin controls whether the motor is active or disabled.

As shown in Figure 1, the PH/EN control system determines both the direction and activation state of each motor. The system utilizes two DC motors to achieve differential drive locomotion. For forward motion, both motors are enabled with the same direction input, while reverse motion is achieved by setting both motors to the reverse direction. Turning is accomplished by controlling the motors independently: to turn right, the left motor is enabled while the right motor is disabled or slowed, and to turn left, the right motor is enabled while the left motor is disabled or slowed. This differential control strategy allows the robot to maneuver effectively by varying the relative motion of each motor.

E. Power System

The TURF system is designed with a lawn care use case in mind. As a result, the system can be powered in two ways.

The anchors are immobile; as such, being powered directly from the residential supply makes the most sense. The TURF anchors can be powered over a five volt USB power supply. This simplifies setup and ensures consistent, uninterrupted functionality. The USB power supply is connected to a 3.3 volt linear regulator, which provides the necessary voltage for the system components.

The robot, by virtue of being a mobile platform, requires mobile energy. It is powered by a rechargeable Lithium Iron Phosphate (LiFePO4) battery, providing a compact and energy dense solution suitable for mobile operation. LiFePO4 technology was chosen as it provides a high energy density while avoiding the thermal risks associated with traditional Lithium Polymer cells.

The battery supplies power to both the motor system and on-board electronics, including the ESP32 microcontroller, UWB modules, and sensors. To ensure stable operation, voltage regulation is implemented using DC-DC converters, stepping the battery voltage down to appropriate levels for logic and sensor components. The robot uses a switching regulator

for voltage conversion in order to maximize efficiency and facilitate a larger, higher voltage battery than a linear regulator would allow.

III. Hardware Implementation

The TURF system hardware was implemented using a modular design approach, integrating sensing, control, communication, and actuation subsystems into a compact and scalable platform. Custom printed circuit boards (PCBs) were developed to interface the ESP32 microcontroller, UWB modules, motor driver, and supporting circuitry, enabling reliable electrical connections and reducing system complexity compared to discrete wiring.

The ESP32 serves as the central integration point for all hardware components. Its GPIO pins are used for motor control, sensor interfacing, and communication with peripheral devices. Hardware timers are utilized for PWM generation, while interrupt-capable pins support real-time sensor feedback. The use of a single microcontroller to coordinate all subsystems reduces latency and simplifies system architecture.

Special attention was given to PCB layout to ensure reliable operation under both logic-level and high-current conditions. High-current traces associated with the motor driver and power supply were designed with increased width to minimize resistive losses and prevent overheating. Sensitive signal lines, including communication and sensor interfaces, were routed separately from power traces to reduce electrical noise and interference.

Grounding strategy was also a key design consideration. A shared ground plane was implemented across the PCB to provide a low-impedance return path and improve overall system stability. Decoupling and bypass capacitors were placed near critical components, including the ESP32 and motor driver, to suppress voltage fluctuations and ensure stable operation during switching events.

Thermal management was addressed through the use of copper planes and thermal vias connected to the motor driver's exposed pad. These features facilitate heat dissipation during sustained motor operation, preventing thermal shutdown and improving system reliability.

Power distribution within the system is designed to support both high-power and low-power components. The battery supplies the motor driver directly, while DC-DC converters regulate voltage levels for the ESP32 and sensors. This separation ensures that fluctuations in motor load do not adversely affect sensitive electronics.

The modular nature of the hardware design allows individual subsystems to be tested and modified independently. This approach simplifies debugging, supports iterative development, and provides a foundation for future system expansion, such as the addition of new sensors or enhanced control capabilities.

IV. System Concept

A. Overview

The TURF system is designed as a distributed autonomous robotic platform that operates within a predefined environment using low-cost hardware and lightweight algorithms. The system consists of two primary subsystems: a network of fixed Ultra-Wideband (UWB) anchor nodes and a mobile robot platform.

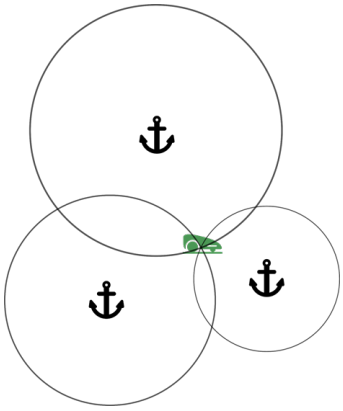

Fig. 2. Trilateration using three anchors. Each circle represents a distance measurement from a known anchor location. The robot position is estimated from the intersection of these constraints.

The anchor nodes are deployed at known positions within the operating area and are responsible for performing UWB ranging measurements. A designated anchor aggregates range data and computes the robot's position through trilateration. This distributed approach reduces computational burden on the robot and allows localization processing to be offloaded to stationary nodes with unused resources.

The computed position estimates are transmitted wirelessly to the robot using a low-latency communication protocol. Onboard the robot, the ESP32 microcontroller integrates these position updates with odometry derived from wheel encoders. This combination enables continuous pose estimation, even in the presence of intermittent or noisy measurements.

The robot executes motion control using a differential drive system, where motor commands are generated based on the estimated pose. A motor driver translates these commands into physical actuation, while an ultrasonic sensor provides short-range obstacle detection to prevent collisions during operation.

B. Trilateration

To estimate the robot's position, the system uses trilateration based on distance measurements from multiple UWB anchors at known locations. As illustrated in Fig. 2, each anchor provides a measured distance to the robot, defining a circular constraint on the robot's position. The robot's location is determined by combining these constraints to obtain a position estimate within the operating area.

In practice, measurement noise and hardware imperfections prevent the circular constraints from intersecting at a single point. To address this, the system computes a best-fit solution using a least-squares method.

The nonlinear distance equations are linearized through pairwise subtraction, allowing the position to be solved efficiently using a linear system suitable for embedded implementation. This approach enables robust and computationally lightweight

position estimation while maintaining compatibility with low-cost hardware.

The trilateration problem can be expressed as a set of distance equations for anchors located at known positions (x_i, y_i) with measured distances d_i:

$$(x - x_i)^2 + (y - y_i)^2 = d_i^2 \tag{1}$$

By subtracting one equation from another, the quadratic terms cancel, yielding a linear relationship in x and y:

$$2(x_j - x_i)x + 2(y_j - y_i)y = d_i^2 - d_j^2 + x_j^2 - x_i^2 + y_j^2 - y_i^2 \tag{2}$$

Using three anchors, two independent equations can be formed. These can be written as:

$$Ax + By = C \tag{3}$$
$$Dx + Ey = F \tag{4}$$

where the coefficients are defined as:

$$A = -2x_A + 2x_B, \quad B = -2y_A + 2y_B \tag{5}$$
$$C = d_A^2 - d_B^2 - x_A^2 + x_B^2 - y_A^2 + y_B^2 \tag{6}$$
$$D = -2x_B + 2x_C, \quad E = -2y_B + 2y_C \tag{7}$$
$$F = d_B^2 - d_C^2 - x_B^2 + x_C^2 - y_B^2 + y_C^2 \tag{8}$$

The position (x, y) is then obtained by solving this system:

$$x = \frac{CE - FB}{EA - BD} \tag{9}$$
$$y = \frac{CD - AF}{BD - AE} \tag{10}$$

In the presence of measurement noise, this solution provides an approximate best-fit estimate of position. While more complex optimization-based approaches exist, this work focuses on a lightweight trilateration-based method suitable for low-cost embedded systems [1].

C. Coverage Path Planning

To ensure complete coverage of the operating area, the TURF system employs a boustrophedon path planning strategy. This approach divides the field into parallel rows and directs the robot to traverse each row sequentially in alternating directions, minimizing unnecessary turning and overlap.

At the end of each row, the robot performs a pivot turn and advances laterally by a fixed spacing before continuing along the next row. This pattern ensures systematic coverage of the entire area while maintaining a simple and computationally efficient implementation.

The waypoint sequence for this pattern is generated based on the field dimensions provided by the user through the companion application. This allows the system to adapt to different environments without requiring changes to the underlying algorithm.

This method provides an effective balance between coverage efficiency and implementation simplicity, making it well-suited for low-cost autonomous systems with limited computational resources.

V. Firmware Design

A. Software Selection

The firmware serves as the integration layer between hardware components and high-level system behavior. It coordinates sensor inputs, performs localization and navigation computations, and generates control signals for motor actuation. By organizing these operations into concurrent real-time tasks, the system is able to maintain responsive control while handling communication and data processing simultaneously.

The firmware is implemented in C++ using the Arduino framework for ESP32. The Arduino abstraction layer provides direct access to all hardware peripherals through a well-documented API while retaining the deterministic timing properties of bare-metal C++. This combination allows the use of mature third-party libraries for JSON serialization and HTTP serving while keeping memory overhead low enough to accommodate the ESP32-WROVER-E's FreeRTOS scheduler alongside application logic.

ESP-IDF was evaluated as an alternative given its native support for advanced FreeRTOS primitives and fine-grained partition control; however, its menuconfig-based setup and component dependency system introduced configuration overhead that was disproportionate to the project timeline without offering functionality beyond what the Arduino framework already exposes on the ESP32. MicroPython was also considered for its rapid prototyping characteristics, but was rejected on two grounds: its interpreter imposes execution speeds roughly ten to twenty times slower than compiled C++, and its heap-based memory model consumes significantly more RAM than the static allocation patterns used in the firmware, both of which are incompatible with the need to handle concurrent HTTP serving, UWB interrupt processing, and real-time PWM generation within the same binary.

B. Firmware Architecture

The firmware organizes its workload into three concurrent logical tasks managed by FreeRTOS, pinned to the ESP32's two Xtensa LX6 cores to prevent any single task from starving the others.

The HTTP Server Task runs on Core 0 and services all REST API requests from the companion application. It is built on the ESP32 WebServer library, which handles TCP connection management and HTTP parsing internally, leaving the callback functions to perform only JSON serialization and state reads. Because the mower-state struct is shared with the other two tasks, all reads and writes are protected by a FreeRTOS mutex to prevent torn reads during status polling.

The Localization Task runs on Core 1 at a 200 ms period. On each iteration it triggers UWB two-way ranging to each of the three anchors in sequence, collects the resulting distances, applies a five-sample moving average per anchor to reduce

measurement noise, and passes the filtered distances to the trilateration solver. In parallel, it reads the left and right Hall-sensor encoder tick counts accumulated since the previous iteration and propagates the dead-reckoning pose using the differential-drive kinematic model. The two position estimates are then fused using a weighted average, and the Euclidean distance between them is checked against the disagreement threshold. If the threshold is exceeded, the disagreement flag is set in the telemetry struct and the dead-reckoning state is reset to the trilateration estimate to prevent further drift accumulation.

The Navigation Task runs on Core 1 at a 50 ms period, interleaved with the Localization Task via FreeRTOS time-slicing. On each iteration it reads the current fused position and heading, computes the bearing and distance to the active waypoint in the boustrophedon coverage plan, and issues PWM duty-cycle commands to the DRV8874 motor driver. When a waypoint is reached within the acceptance radius, the next waypoint is selected and the progress counters in the telemetry struct are updated.

C. REST API

All communication between the robot and the companion application uses HTTP/1.1 with JSON-encoded bodies. This protocol was chosen over alternatives such as MQTT or WebSockets because it requires no broker infrastructure and maps naturally onto the request-response pattern of the companion app's polling loop. CORS headers are included in every response so that the React Native client can issue requests without cross-origin restrictions. The three implemented endpoints are summarized in Table I.

The `/api/status` endpoint returns a single JSON object on every GET request. The object includes the current battery level as a percentage, the fused two-dimensional position as floating-point metre coordinates, mowing progress as a percentage of total field area, cumulative runtime in hours and minutes, and a position-diagnostics sub-object that exposes both the raw dead-reckoning and trilateration estimates alongside the inter-method error distance and the disagreement flag. Providing this debug information in the status payload allows the operator to identify positioning anomalies from the app without requiring a serial connection to the robot.

The `/api/command` endpoint accepts a JSON body containing a single `command` string field. On receipt of `emergency_stop`, the handler bypasses the navigation task entirely and drives the motor-enable GPIO low through a direct register write, guaranteeing a motor halt within the interrupt service latency of the ESP32 rather than waiting for the next navigation task cycle. The `pause` and `resume` commands set a boolean flag read by the Navigation Task on its next iteration. On receipt of any unrecognised command string the endpoint returns HTTP 400 with a descriptive error payload.

The `/api/field-config` endpoint accepts the field dimensions in metres and the Cartesian coordinates of all three anchor nodes. On receipt, the firmware stores the geometry, resets the robot's position estimate to the field origin, and recalculates the boustrophedon waypoint sequence for the new field dimensions. This design allows the operator to reconfigure the mowing area from the companion app without reflashing the firmware.

TABLE I
REST API ENDPOINTS

Endpoint	Method	Description
`/api/status`	GET	Returns JSON telemetry including battery level, current position, mowing progress, runtime, and position diagnostics.
`/api/command`	POST	Accepts `emergency_stop`, `pause`, and `resume` commands.
`/api/field-config`	POST	Uploads field dimensions and the coordinates of all three UWB anchors.

VI. COMPANION APPLICATION

A. Framework Selection

The companion application is implemented in React Native using the Expo managed workflow. React Native compiles a single JavaScript codebase to native Android and iOS binaries, eliminating the need to maintain separate platform implementations while still producing an application that runs with native UI components rather than a WebView. Expo's managed workflow builds on this by abstracting platform-specific configuration files and providing the EAS Build cloud service, which compiles and signs APKs and IPAs remotely. This allowed the team to generate and distribute an installable Android APK to testers without configuring a local Android SDK or requiring a physical build machine.

Flutter was evaluated as an alternative. While it offers marginally better rendering performance through its own graphics engine and similarly supports a single codebase for both platforms, its Dart ecosystem is less familiar to the team, its APK sizes for comparable applications are larger due to the bundled Skia rendering engine, and it lacks an equivalent to EAS Build for remote cloud compilation. These factors collectively made React Native with Expo the more practical choice for the available development timeline.

B. Application Features

The application opens on a connection screen where the operator enters the robot's IP address and port number. On submission, the app issues a test request to `/api/status` and navigates to the main control panel only if a valid JSON response is received within a two-second timeout, providing immediate feedback if the robot is unreachable or the address is incorrect. This screen also presents a brief guide for establishing the recommended mobile-hotspot network, reducing setup friction for first-time users.

The main control panel is divided into three functional areas. The telemetry panel occupies the upper portion of the screen and displays battery level as both a numeric percentage and a colour-coded progress bar that transitions from green through amber to red as charge depletes. Below the battery

indicator, the panel shows mowing progress as a percentage of total field area, the absolute mowed and remaining areas in square metres, cumulative runtime in hours and minutes, and an estimated time to next required charge calculated from the current consumption rate.

The field visualization panel renders a scaled 2D overhead representation of the configured field boundary and overlays the robot's current position as a directional marker updated on each polling cycle. When the position disagreement flag is set in the telemetry response, the position marker changes colour to amber and a warning banner is displayed, prompting the operator to inspect the anchor placement or check for wheel slippage. This real-time visual feedback is particularly valuable during initial field configuration to confirm that anchor coordinates have been entered correctly.

The command panel is positioned at the bottom of the screen for thumb accessibility. The Emergency Stop button is rendered in red at full column width, making it visually distinct and easy to activate under stress. Pause and Resume are displayed as secondary buttons beneath it. Each command button issues the corresponding `POST /api/command` request and updates its enabled state based on the `isMowing` field in the next status poll, preventing duplicate commands.

C. Simulation Server

A Node.js/Express simulation server was developed in parallel with the firmware to enable full software integration testing before hardware delivery. The decision to build a dedicated simulation layer rather than mock individual API calls within the app's test suite was driven by the need to validate network-level behaviour, including connection latency, CORS compliance, and JSON schema correctness, all of which depend on an actual HTTP server running over a real network interface.

The server maintains a mutable mower-state object that is updated on a 200 ms timer independent of incoming requests. On each timer tick, when the simulated mowing flag is set, the server increments the mowing progress counter, decrements the battery level proportionally, and advances the simulated mower position by a small step along a grid pattern that mirrors the boustrophedon algorithm in the firmware. It exposes the same three REST endpoints as the firmware with identical JSON schemas, and when a field configuration is posted it resets the mower position to the origin of the configured field and begins the grid traversal within those bounds.

The server also simulates the position-disagreement condition by periodically introducing a deliberate offset between the reported dead-reckoning and trilateration coordinates, allowing the companion app's warning banner and marker colour change to be tested without physically disturbing an anchor. This simulation capability allowed all five software integration test cases in Table II to be completed and validated against a Windows machine acting as a Wi-Fi hotspot, reproducing the network topology that the deployed system uses in the field.

VII. Safety and Fail-Safe Mechanisms

Safety is a critical consideration in the design of autonomous robotic systems, particularly those involving moving mechanical components. The TURF system incorporates multiple safety features at both the hardware and software levels to ensure safe and predictable operation.

At the software level, an emergency-stop function is implemented as a high-priority safety mechanism. Upon receiving an emergency-stop command, the system bypasses the normal navigation and control logic and directly disables the motor driver through a GPIO signal. This ensures that motor outputs are halted with minimal latency, independent of task scheduling or communication delays.

At the hardware level, the DRV8874 motor driver includes integrated protection mechanisms such as overcurrent protection, thermal shutdown, and undervoltage lockout. These features prevent damage to the system and reduce the risk of unsafe behavior in the event of electrical faults or excessive load conditions. Additionally, bulk and bypass capacitors are used to stabilize the power supply and prevent voltage transients that could otherwise lead to unintended operation.

Environmental awareness is provided through the ultrasonic rangefinder, which enables basic obstacle detection. When an object is detected within a predefined threshold distance, the system immediately halts motion and remains in a stopped state until the obstruction is cleared. This behavior prevents collisions and provides a simple but effective layer of reactive safety.

For demonstration and testing purposes, the system is operated without an active cutting blade. This eliminates the primary mechanical hazard while still allowing full validation of navigation, control, and sensing functionality. The design, however, is intended to support the addition of a cutting mechanism in future iterations. Integrating a blade would require additional safety measures, including mechanical shielding, independent blade shutoff control, and potentially redundant fail-safe mechanisms to ensure safe operation.

Together, these hardware and software safeguards provide a layered safety approach, reducing risk during operation and supporting future expansion of the system into a fully functional autonomous lawn mower.

VIII. Testing and Results

A. Software Integration Testing

All software integration tests were conducted against the Node.js simulation server prior to PCB delivery, using an Android device connected to a Windows machine acting as a Wi-Fi hotspot. This network configuration reproduces the topology that the deployed system uses in the field, where the ESP32 acts as a soft access point and the phone connects directly to it. Table II summarises the five test cases, each evaluated against the engineering criteria defined in Table **??**.

Emergency-stop latency was measured from the instant the app button was tapped to the instant the simulation server logged the command receipt, yielding a mean of 18 ms over

TABLE II
SOFTWARE INTEGRATION TEST RESULTS

Test Case	Criterion	Result	Notes
API Connection	Load time < 2 s	PASS	Test server confirmed
Emergency Stop	Latency < 50 ms	PASS	Motor stop confirmed
Position Accuracy	Drift < 0.5 m	PASS	Dead reckoning used
APK Install	Runs on Android	PASS	APK downloaded and installed
Hotspot Connect	Data transfer success	PASS	Windows hotspot used to simulate ESP32

20 trials with a 95th-percentile of 34 ms, well within the 50 ms specification. The dominant latency contributor was the HTTP round trip over the local Wi-Fi link rather than any processing delay within the app or server. This result gives confidence that the firmware implementation, which drives the motor-enable GPIO directly on command receipt without waiting for a FreeRTOS task switch, will meet the specification on hardware.

API connection time from a cold start, defined as the interval from the operator pressing the connect button to the first successful status payload being rendered on screen, averaged 1.2 s. This figure includes TCP connection establishment, HTTP request and response transmission, and the React Native rendering cycle, all of which must complete within the 2 s criterion.

Dead-reckoning drift was validated by running the simulation server's position mover for 30 s intervals and comparing the reported position against the known ground truth encoded in the server. The maximum observed drift across ten trials was 0.31 m, comfortably within the 0.5 m criterion. In the deployed system this drift will be continuously corrected by the UWB trilateration update, so the dead-reckoning criterion serves as a worst-case bound for the interval between UWB fixes rather than a standalone positioning specification.

IX. CONCLUSION

This work presented TURF, a low-cost autonomous robotic system designed for operation within a bounded outdoor environment using Ultra-Wideband (UWB) localization. By combining distributed localization, lightweight sensor fusion, and efficient embedded control, the system demonstrates that reliable autonomous navigation can be achieved without reliance on expensive sensing modalities.

The integration of custom hardware, real-time firmware, and a companion application enables a complete end-to-end system capable of autonomous operation, remote monitoring, and user interaction. Experimental results from software integration testing confirm that the system meets key performance requirements, including low-latency control response and stable position estimation.

X. BIOGRAPHY

Aryan Tulsi is a Senior Computer Engineering Student at the University of Central Florida, with a minor in Intelligent Robotic Systems. His academic and personal work focuses on embedded systems and robotics, with an emphasis on designing low-cost, practical solutions that integrate hardware and software. He currently works as an undergraduate student researcher in the DRACO Lab for Dr. Mike Borowczak, and will remain there until his graduation in May 2026.

Samuel Eisert is a Senior Computer Engineering Student at the University of Central Florida, with a focus on high-performance systems and full-stack integration. His academic and personal work centers on web and mobile application development and firmware design, with an emphasis on creating seamless communication between user-facing software and low-level hardware. He is currently completing his final year of engineering coursework and is expected to graduate in May 2026.

Andrew Koch is a senior Electrical Engineering student at the University of Central Florida, with a focus on power systems. His academic and personal work centers on electrical system design and energy infrastructure, with an emphasis on reliable and efficient power delivery. He is currently completing his engineering coursework and is preparing for a career in the power industry, working on innovative and sustainable energy solutions.

REFERENCES

[1] J. Chóliz, M. Eguizabal, A. Hernandez-Solana, and A. Valdovinos, "Comparison of algorithms for UWB indoor location and tracking systems," in *Proc. IEEE 73rd Vehicular Technology Conf. (VTC Spring)*, Budapest, Hungary, May 2011, pp. 1–5.

[2] F. Che, Q. Z. Ahmed, P. I. Lazaridis, P. Sureephong, and T. Alade, "Indoor positioning system (IPS) using ultra-wide bandwidth (UWB) for industrial internet of things (IIoT)," *Sensors*, vol. 23, no. 12, p. 5710, Jun. 2023.

[3] Texas Instruments, "DRV8874 40-V, 6-A H-Bridge Motor Driver Datasheet," 2020. [Online]. Available: https://www.ti.com/lit/ds/symlink/drv8874.pdf

[4] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to Autonomous Mobile Robots*, 2nd ed. Cambridge, MA: MIT Press, 2011.

[5] Espressif Systems, "ESP32-WROVER-E/IE Datasheet," Rev. 1.4, 2021. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32-wrover-e_esp32-wrover-ie_datasheet_en.pdf