ECE 445 - Spring 2025

Senior Design Final Report

Antweight Battlebot

Team 2
Nandika Vuyyuri (vuyyuri2), Gauthami Yenne (gyenne2), Jingyu Kang (jingyuk2)

TA: Haocheng Bill Yang

Abstract:

This report presents the design, development, and testing of an antweight battlebot for a 1v1 combat environment. This robot features a dual-motor drivetrain and an active spinning weapon mechanism, both controlled via an ESP32 microcontroller. A custom software interface enables wireless communication between the robot via wiFi. The chassis was modeled using CAD and fabricated with 3D printing to optimize weight and durability while adhering to the 2 lb weight constraint. A custom PCB was designed to integrate the electrical components efficiently. The report discusses the system architecture, mechanical and electrical integration, and performance testing under simulated battle conditions.

Table Of Contents:

1. Introduction

1.1. Problem

Combat robots offer a hands-on platform for applying concepts from mechanical design, embedded systems, and real-time communication. The antweight class, with its strict size and weight limitations, presents a unique engineering challenge that emphasizes creativity, integration, and electrical and computer engineering concepts.

The motivation behind this project was to design and build a fully functional antweight battlebot that could perform competitively in a 1v1 combat environment while also serving as a platform for applying embedded hardware and wireless control concepts. While all teams were required to create custom PCBs without relying on breadboards or development kits, our project stood out by implementing a fully functional web application as the robot's controller. This allowed for real-time, platform-independent control via WiFi, eliminating the need for physical remotes or proprietary apps.

Although our custom PCB did not function as intended due to hardware integration issues, the experience deepened our understanding of embedded system design, debugging, and contingency planning. We successfully pivoted to a fallback configuration using modular components to keep the system operational and demonstrate the web-based control interface. The combination of robust mechanical design, wireless control architecture, and adaptability under pressure defined the core strengths of our battlebot project.

The goal of this project was to design and build a functional and competitive antweight battlebot that could effectively operate in a 1v1 combat environment. The bot needed to meet specific performance criteria while adhering to strict size and weight constraints. A key challenge was ensuring the robot's timely and accurate control through wireless communication, alongside the integration of a powerful weapon system that could perform consistently during battle conditions. The design needed to achieve a balance of control, power, and durability while remaining under the 2-pound weight limit, using components that were both lightweight and reliable under impact and stress.

1.2. Solution

To build a competitive antweight battlebot, we developed a web-controlled robot that utilizes an ESP32 for wireless communication, allowing real-time control through any device with a browser. The robot features a modular design with lightweight, durable components to stay under the 2-pound weight limit while maintaining strength and resilience. The weapon system consists of a three-speed spinning blade, powered by regulated voltage for consistent performance. Despite initial integration issues with the

custom PCB, we successfully implemented a fallback solution using modular components, ensuring functionality.

1.4. High-Level Requirements

- Wireless Control: The BattleBot must be controllable wirelessly via either an Android app or a GameCube controller, with the communication system delivering timely and accurate control to allow precise manipulation of the robot's speed and direction.
- Spinning Blade Weapon: The robot's weapon system must be a three-speed spinning blade, powered through regulated voltage to ensure consistent performance across all operating speeds.
- Weapon Speed and Stability: The spinning blade must achieve speeds of over 500 RPM, maintaining stable control even during high-speed motion and physical impacts typical of combat environments.
- Weight Constraint: The BattleBot must remain under the 2-pound weight limit, achieved through careful selection of lightweight and durable components that ensure both performance and resilience.

2. Design

2.1. Block Diagram

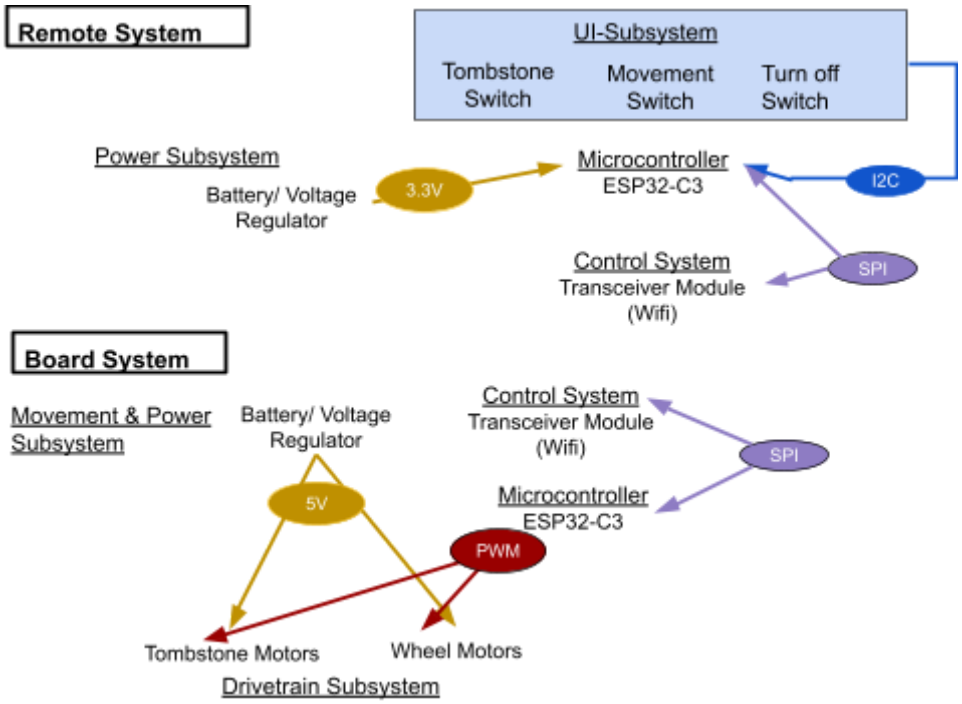

Figure 1. Block Diagram

2.2. Physical Design of Robot

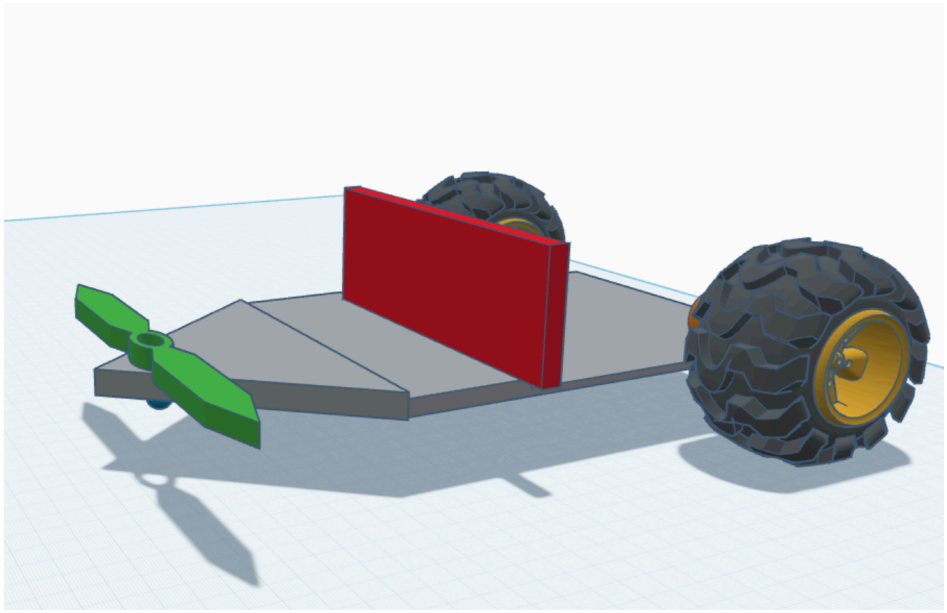

Figure 2: CAD Drawing of Battlebot

2.3. Subsystem

2.3.1. Power System

Overview:

The Power System uses an Energizer Max 9V alkaline battery as the main power source. Two voltage regulators are used to distribute appropriate voltages to the system components: one steps down the 9V to 5V for the DC motors and DRV8833 motor driver, and the other steps the voltage down to 3.3V for the ESP32-C3 microcontroller. The ESP32-C3 operates at 3.3V logic level, while the DRV8833 motor driver and motors operate optimally at 5V.

The Energizer Max 9V battery supplies a nominal 9V and can provide brief bursts of up to ~500–600 mA, which is sufficient for short operation durations typical of BattleBot matches. Voltage regulators were chosen to handle these loads and maintain voltage stability under motor startup and load conditions. To ensure stable operation, decoupling capacitors were placed near the ESP32-C3 and motor driver to minimize voltage ripple.

Total peak current draw was estimated to assess worst-case performance requirements:

- Total = I_weapon + (2 × I_wheel) + I_ESP32
- = 750 mA + (2 × 650 mA) + 0.5 A = 2.55 A

At this peak current, power dissipation can be approximated as:

- P = V × I_total = 9 V × 2.55 A = 22.95 W

This analysis shows that while average current consumption remains around 630 mA under normal conditions, the power subsystem must tolerate significantly higher instantaneous loads during simultaneous weapon spin-up and movement surges. Future iterations may consider higher-capacity power sources or capacitor banks to better handle these peaks.

RV Table:

Requirement	Verification
The voltage must be stepped down from 9V to 5V for motors and DRV8833	Use a voltmeter to verify voltage regulator outputs at 5V
The voltage must be stepped down from 9V to 3.3V for ESP32-C3	Measure output of 3.3V regulator with a multimeter
Power subsystem must provide at least 500 mA at 5V ±0.1V	Use a DC power supply to measure the current readings

Battery voltage must remain stable during operation	Measure voltage sag during motor activity with a multimeter
DRV8833 input must remain within 2.7V–10.8V and deliver up to 1A per channel	Measure input voltage using the voltmeter and current measurements with the DC motor supply
Battlebot should maintain consistent power delivery throughout the match's duration	Measure the required measurements over multiple trials to verify consistency

2.3.2. Communications Subsystem

Overview:

The communication system manages both the programming and real-time control of the battlebot. Initially, the ESP32-C3 microcontroller was programmed using a USB-to-USB-C cable connected to a laptop running the Arduino IDE. This allowed us to upload the firmware directly and verify serial output for debugging.

For real-time control during matches, the system uses Wi-Fi communication. A localhost web interface hosted on the ESP32-C3 receives HTTP commands from a browser-based UI running on a control device (such as a laptop or phone). These commands are parsed by the ESP32-C3 and translated into PWM signals on GPIO pins, which control the motors via the DRV8833 motor driver.

Max observed latency: 282 ms
Typical human reaction time: ≈200–250 ms
Total response delay (human + system): ≈0.48–0.53 s

This is acceptable for a user-controlled combat bot, where commands are issued manually. However, the latency would be too high for precise, real-time autonomous reactions, which require much lower system delays (typically <100 ms) to respond effectively to fast-changing environments.

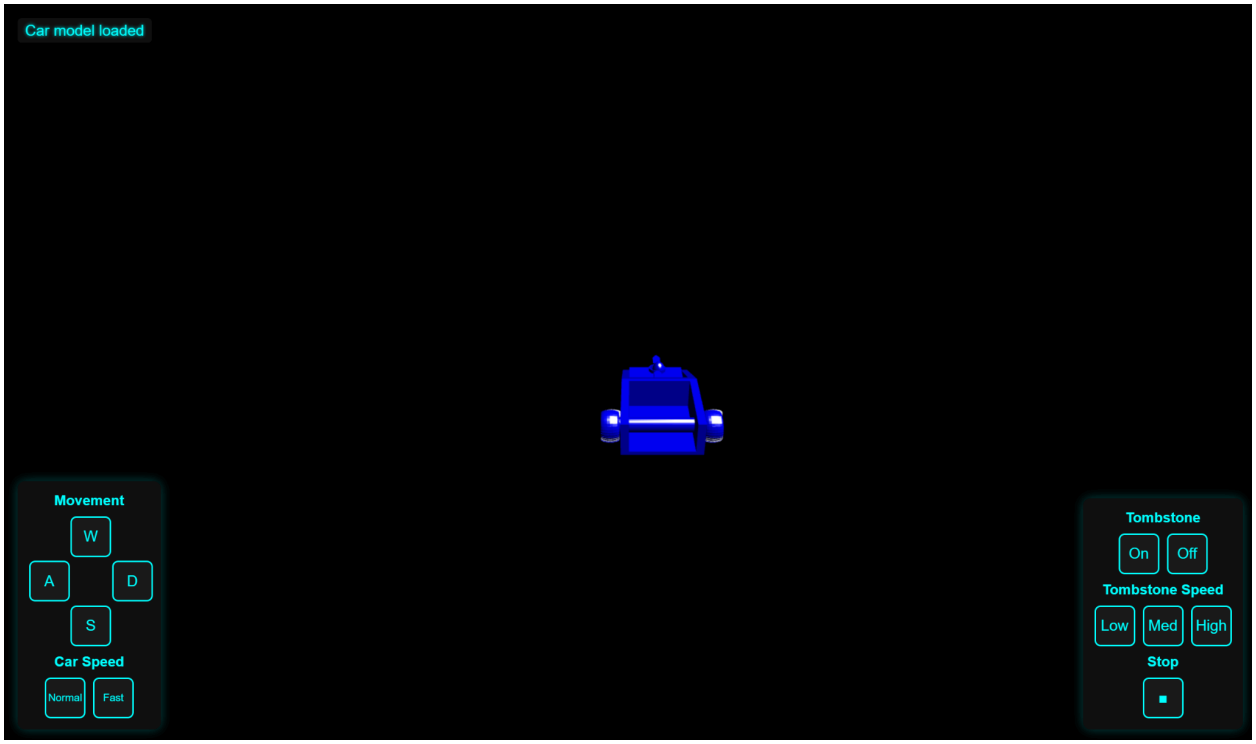

Figure 3. UI Control Interface

RV Table:

Requirement	Verification
ESP32-C3 must be programmable via USB-C for firmware upload	Upload code using Arduino IDE and verify the serial monitor output
ESP32-C3 shall host a local web server to receive HTTP commands	Open the browser interface and confirm a successful server response
Bot control shall be wireless via Wi-Fi with low latency	Measure the delay between button press and motor response
PWM signals must be generated correctly to control the DRV8833 driver	Observe motor behavior and verify against command inputs
Communication latency shall be under 100 ms for responsiveness	Use video timestamps or serial logs to measure end-to-end delay
Localhost interface must support full motion and weapon control	Verify the functionality of each control through the UI

Only authorized devices on local Wi-Fi shall access the bot	Test with restricted IP range or private network settings

2.3.3. Control Subsystem

Overview:

The ESP32-C3 microcontroller is the central control unit of the battlebot. It was initially programmed using a USB-to-USB-C cable through the Arduino IDE. For real-time control, the ESP32-C3 hosts a localhost web server, allowing commands to be sent wirelessly over Wi-Fi from a browser-based user interface.

Upon receiving movement and weapon control inputs, the ESP32-C3 translates them into PWM signals output through its GPIO pins. These signals control the DRV8833 motor driver, which powers both the Greartisan drivetrain motors and the weapon motor. The ESP32-C3's built-in voltage regulator ensures a stable 3.3V logic level for consistent operation.

RV Table:

Requirement	**Verification**
ESP32-C3 must communicate wirelessly via Wi-Fi to receive user commands	Use the browser-based UI to send commands and verify reception via serial monitor or bot response
ESP32-C3 must generate correct PWM signals to control the DRV8833 motor driver	Observe the motor driver output or use an oscilloscope to confirm the PWM waveform from ESP32-C3 GPIO pins
ESP32-C3 must be programmable via USB-C for code upload	Upload code through Arduino IDE and verify successful deployment and serial debug output

2.3.4. Weapon Subsystem

Overview:

For our weapon mechanism, we used a Greartisan 12V 100 RPM DC motor instead of the originally planned EMAX RS2205 brushless motor. The Greartisan motor is a high-torque, low-RPM gear motor with a 1:298 reduction ratio, making it suitable for

rotating a heavy weapon bar without requiring complex electronic speed control. It is powered via the regulated 5V output and controlled by the DRV8833 motor driver, which receives PWM signals from the ESP32-C3.

The motor is securely mounted to the chassis to ensure stability during high-load operation. Although precise speed control was not required, the weapon can be toggled on or off and ramped via PWM duty cycle adjustment if necessary. We estimated the torque requirement and compared it with the motor's output to verify our weapon motor's suitability for spinning the tombstone blade.

DC Motor: Greartisan 12V 100RPM DC Motor

- Gear Ratio: 1:298
- Rated Torque: 2 kg·cm = 0.196 Nm
- Rated Speed: 100 RPM
- Operating Voltage: 12V (powered at 5V in our design, leading to lower speed/torque)

Motor Driver: DRV8833 Dual H-Bridge

- Operating Voltage Range: 2.7V–10.8V
- Peak Current per Motor Channel: Up to 9600 mA (9.6 A)
- Used for: Driving the weapon motor via PWM from ESP32-C3

Estimated Torque Calculation:

$T = r \times F = 0.07\ m \times (0.12\ kg \times 9.8\ m/s^2) = 0.0823\ Nm$

Angular Velocity (ω):

$\omega = 2\pi \times (100[rpm] / 60) \approx 10.47\ rad/s$

Moment of Inertia (I):

$I = (1/2) \times m \times r^2 = 0.5 \times 0.12\ kg \times (0.07\ m)^2 \approx 0.000294\ kg{\cdot}m^2$

Rotational Kinetic Energy (E):

$E = (1/2) \times I \times \omega^2 = 0.5 \times 0.000294 \times (10.47)^2 \approx 0.0161\ J$

RV Table:

Requirement	Verification

Weapon motor must turn on/off in response to ESP32-C3 control signals	Weapon subsystem must turn off completely if kill switch is activated or power is disconnected
DC motor must be powered at 5V $\pm 5\%$ at 600 mA or above	Use a multimeter to measure the voltage and current with DC power supply over multiple trials
Motor must be securely mounted to withstand rotational force	Perform mechanical tests by spinning weapon at full speed and observing for vibration or mounting failure

2.3.5. Drivetrain Subsystem

Overview:

The drivetrain subsystem consists of two Greartisan 12V 100 RPM high-torque DC motors connected to 48mm Mecanum wheels. These motors are controlled using a DRV8833 dual H-bridge motor driver, which allows for independent control of each motor's speed and direction through PWM signals generated by the ESP32-C3 microcontroller.

The use of Mecanum wheels enabled the battlebot to execute agile movements, including pivot turns and enhanced maneuverability. The motors are powered by a regulated 5V source from the power subsystem. All control signals are handled directly by the ESP32-C3 GPIO pins, and no external ESCs or signal level shifting were necessary.

RV Table:

Requirement	**Verification**
The battlebot should be able to move with a minimum speed of 1.5 m/s^2.	Noted the distance the bot travelled in 10 seconds and then divided this distance by 10 to get the speed
The movement of the battlebot should not interfere with the weapon subsystem while they are functioning together.	The controller should be able to control the direction and speed of the battlebot and the speed of the weapon precisely at the same time.
The battlebot must operate smoothly on various surfaces without any problems.	The battlebot was tested on lab bench tables, lab floors, study room bench tables, and hallway floors, and operated flawlessly

3. Verification

3.1. Communication Subsystem

Testing Approach:

- **WiFi Connection:** The system must establish a successful connection between the ESP32 and the computer with a latency of 1000 ms or less.
- **Weapon Speed Levels:** The different PWM duty cycles should output different weapon motor speeds and are verified using the oscilloscope.
- **Kill Switch:** The kill switch shuts down the battlebot and is verified by visually inspecting the robot.

Results:

- WiFi Latency vs. Trial

Trial	WiFi Latency of Mono Speed Testing (ms)	WiFi Latency of Different Speed Testing (ms)
1	12	124
2	28	282
3	24	186
4	18	142
5	16	94

- PWM duty cycle vs. Trigger Position

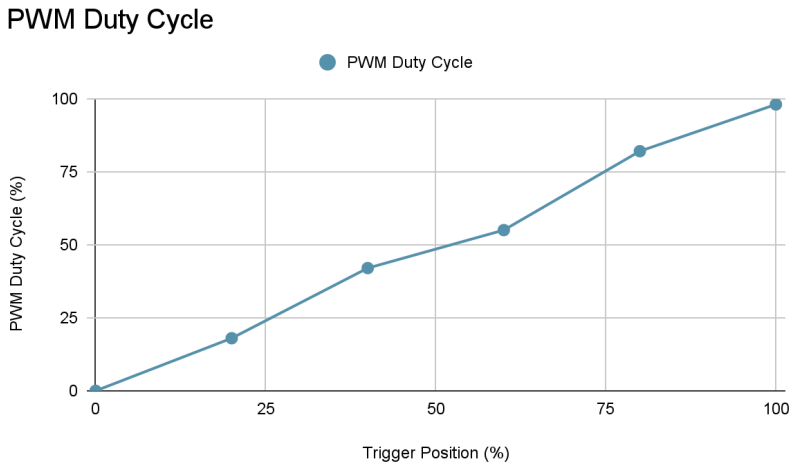

Figure 4. PWM Duty Cycle vs. Trigger Position

- Power cut-off times vs. Trial

Trial	Cut-Off Time (s)
1	0.48
2	0.32
3	0.44
4	0.40
5	0.36

Validation:

- All latency tests passed with values under 1000 ms.
- Although the latency values increased dramatically for the different weapon speed level testing, the latency values were still within 1000 ms.
- The measured PWM duty cycle values were within $\pm 3\%$ of the expected value.
- Kill switch constantly shuts down the battlebot system with a less than 500 ms cut-off time.

3.2. Drivetrain Subsystem

Testing Approach:

- **Motor Speed:** Measure RPM of the wheels using marking on the wheels as well as visually verifying the speed change depending on the different PWM duty cycle inputted.
- **Acceleration:** The battlebot must operate smoothly on various surfaces without any problems.

Results:

- The RPM measurements come out to an average of 247 RPM over 10 measurements.
- The battlebot was tested on lab bench tables, lab floors, study room bench tables, and hallway floors, and operated flawlessly.

Validation:

- The wheels achieved an average of 247 RPM on various surfaces tested.

3.3. Weapon Subsystem

Testing Approach:

- **Blade Dimensions:** The tombstone blade should be a proper dimension that would fit perfectly at the 3D printed chassis as well as must be mounted properly.

- **Blade Speed:** Measure RPM change for different weapon speed levels by checking it visually.
- **Blade Turn On/Off:** The tombstone blade should turn off completely with the switch on the program as well as when the battlebot is completely turned off.

Results:

- Blade dimensions: 0.12 kg mass with 14 cm diameter
- Tested the tombstone blade speed control with WiFi connection by changing the PWM duty cycle and checking the change of speed visually, and verified successfully.
- The tombstone blade turns on and off accordingly to the turn switch in the program, as well as turns off completely with a kill switch.

Validation:

- The tombstone blade was maintained under high-level requirements for the weight as well as dimensions suitable for our 3D printed chassis design.
- The tombstone blade reached 264 RPM.

3.4. Power Subsystem

Testing Approach:

- **Voltage Stability:** The voltage value was measured using a voltmeter at the 5-volt and 3.3-volt input to the ESP32 microcontroller.
- **Kill Switch:** The power should completely cut off when the power is disconnected,and is measured with the voltmeter.

Results:

- Output Voltage vs. Trial

Trial	3.3V Output Voltage (V)	5V Output Voltage (V)
1	3.267	4.967
2	3.324	5.021
3	3.261	5.011
4	3.274	4.974
5	3.295	4.965

- Power cut-off times vs. Trial

Trial	Cut-Off Time (s)

1	0.06
2	0.11
3	0.08
4	0.10
5	0.08

Validation:

- The input voltage for the ESP32 microcontroller was measured within $\pm 5\%$ of the expected values.
- The kill switch completely shuts down the power subsystem within 1 second on average, consequently shutting down the whole system.

4. Cost and Schedule

4.1. Cost Analysis

Labor :

- Estimated at $30/hour per team member
- Software - 20 hours
- Electrical Design - 40 hours
- Mechanical Design - 20 hours
- ($30 x 80 hours) x 3 members = $7200

Parts :

Description	Manufacturer	Part #	Quantity	Cost	Total Cost
XIAO ESP32-C3 Dev Board (Wi-Fi/BLE)	Seeed Studio	XIAO-ESP32-C3	1	$9.8	$9.8
DRV8833 Motor Driver	Texas Instruments	DRV8833	2	$6.99	$13.98
9V Battery	Energizer Max	TP325-3SR70J	2	$10.99	$21.98
Greartisan DC 12V 100RPM	Greartisan	N20	3	$11.99	$35.97
Mecanum Wheel	Hyuduo	Hyuduoktbudiczay1241-12	4	$4.38	$17.52
3-D Printing/Wiring	N/A	N/A	N/A	$5.79	$5.79
Total Cost					$107.73

Schedule :

Week	Task
3/3	Begin full breadboard testing with motor control via ESP32 & DRV8833. Debug power and logic errors during Breadboard Demo week

3/10	Finalize the first trial of the PCB design, ensuring all necessary circuit components are correctly placed. Complete the mobility system of the robot, including motor control, sensor integration, and basic movement testing. Begin preliminary debugging of mobility issues.
3/17	Conduct initial testing on the PCB trial #1, identifying issues with power distribution, signal integrity, and communication. Make necessary revisions based on test results. Continue refining robot mobility and responsiveness. Prepare preliminary documentation for midterm evaluation.
3/24	Finalize the tombstone function (movement mechanism, weight distribution, stability testing). Print and assemble the first trial of the robot's physical structure. Begin integration of the structure with electronics and mobility components. Test overall system stability and basic functionality.
3/31	Finalize the Final PCB design with necessary adjustments based on trial #1 feedback. Send the design for manufacturing. Perform software and firmware debugging while awaiting PCB arrival. Refine the mechanical structure if needed.
4/7	Integrate all system components: final PCB, robot structure, sensors, and software control. Conduct full-system testing, including power-on diagnostics, sensor accuracy, and motion reliability. Begin stress testing and identifying failure points. Prepare for the final demonstration.
4/14	Make final adjustments to both hardware and software. Ensure the robustness and reliability of the system. Conduct full-scale demo rehearsals and troubleshoot potential presentation/demo issues. Finalize poster and report submission. Practice Final Presentation & Demo for ECE 445.
4/21	Mock Demo with TA and test to finalize our PCB
4/23	Used ABS material to print, which was over 2 pounds, so we redesigned the Chassis to be more compact and durable.
4/25	The new PCB is not arriving in time, so we set up the battle bot functionally on a breadboard

5. Conclusion

Our final BattleBot design successfully met the core objectives of the project: developing a compact, fully functional combat robot capable of wireless control, integrated motion, and weapon operation, all within a strict weight limit. By using the ESP32-C3 microcontroller with Wi-Fi-based control, we achieved responsive real-time commands through a custom localhost web interface. The drivetrain, powered by Greartisan DC motors and DRV8833 drivers, provided reliable movement using 48mm Mecanum wheels, while the weapon subsystem delivered consistent performance under load.

Throughout the development process, we overcame several hardware and design challenges, including power distribution issues, PCB manufacturing delays, and motor selection constraints. These challenges strengthened our ability to adapt quickly, redesign subsystems under time pressure, and debug both electrical and mechanical systems in an integrated environment.

This project gave us valuable hands-on experience with embedded systems, PCB design, wireless communication, and full-stack robot integration. Looking ahead, we plan to refine our system by optimizing power efficiency, redesigning the chassis for better durability, and exploring advanced features such as motion sensors and autonomous response. Most importantly, the skills gained from this capstone experience will carry over to future engineering challenges and careers.

5.1. IEEE Code of Ethics #1: Safety

Safety is a critical aspect of our battle bot's design, ensuring the well-being of operators, spectators, and the surrounding environment. We took proactive measures to make sure the bot operates in a controlled and secure environment, minimizing potential hazards. First, we were careful to store the batteries properly, checking for any signs of swelling or damage, and ensured they were disposed of safely after use. We also prioritized motor safety by including a kill switch in our UI design to immediately disable the motor in emergencies. Some of the physical safety measures we took included testing in a designated area with proper barriers to protect spectators. Additionally, any sharp edges or exposed components were covered or enclosed to reduce the risk of injury.

By following these safety practices, we aim to minimize risks and ensure that our battle bot operates responsibly and safely.

5.2. IEEE Code of Ethics #9: Privacy and Security Concerns

Given that our battle bot relies on Wi-Fi connectivity, it is crucial to implement robust security measures to prevent unauthorized access and ensure operator-only control. Our Wi-Fi-controlled bot was operated via a localhost interface on a private network. Only authorized users could send commands to prevent outside interference. By addressing

these security concerns, we ensure that the bot operates in a safe, controlled, and private manner without risk of external interference.

5.3. ACM Code of Ethics 2.2: Fair Competition

To uphold fair competition standards, we will strictly comply with all rules and regulations set by the competition organizers. Ethical participation ensures fairness, integrity, and a level playing field for all competitors. We will adhere to the Competition Guidelines. We will carefully review and follow the competition's official rulebook, ensuring all design and operational aspects comply with event policies.

We will also ensure transparency in Design and Performance. We will honestly report our bot's capabilities and limitations without falsifying data or performance metrics. All test results and competition performances will be documented and reported accurately.

By following these ethical standards, we will ensure that our participation is honest, respectful, and in the spirit of fair competition.

6. References

[1]“ESP32C3 Series Datasheet UltraLowPower SoC with RISCV SingleCore CPU Supporting IEEE 802.11b/g/n (2.4 GHz WiFi) and Bluetooth ® 5 (LE) Including.” Available: https://www.espressif.com/sites/default/files/documentation/esp32-c3_datasheet_en.pdf
[2]Thunder Power RC, “Thunder Power RC,” *Thunder Power RC*, 2019. https://www.thunderpowerrc.com/products/tp325-3sr70j (accessed Mar. 06, 2025).
[3]ESPRESSIF, “ESP32-C3-DevKitM-1 - ESP32-C3 - — ESP-IDF Programming Guide v5.0 documentation,” *Espressif.com*, 2016. https://docs.espressif.com/projects/esp-idf/en/v5.0/esp32c3/hw-reference/esp32c3/user-guide-devkitm-1.html (accessed Mar. 06, 2025).
[4]DrUAV, “4PCS Emax RS2205 2300KV 2600KV 2205 CW/CCW 3-4S Brushless Motor for RC FPV Racing Drone Quad Motor FPV Multicopter With Box,” *Dr.UAV*. https://druav.com/products/4pcs-emax-rs2205-2300kv-2600kv?variant=49281606582485&country=US&currency=USD&utm_medium=product_sync&utm_source=google&utm_content=sag_organic&utm_campaign=sag_organic&gad_source=1&gclid=CjwKCAiAh6y9BhBREiwApBLHC24JyV_lzqQbjjeo0NuXc5qmwhL8mpw_yISX2KHZv3LGWYs7LA8HfhoCELQQAvD_BwE
[5]Texas Instruments, “DRV8833 Dual H-Bridge Motor Driver,” *Texas Instruments*. https://www.ti.com/lit/ds/symlink/drv8833.pdf?ts=1739085202074&ref_url=https%253A%252F%252Fwww.google.com%252F
[6]High, “Pololu 99:1 100rpm 25D High Power 12V Geared Motor,” *Technobotsonline.com*, 2025. https://www.technobotsonline.com/pololu-99-1-100rpm-25d-high-power-12v-geared-motor.html (accessed Mar. 06, 2025).
[7]Association for Computing Machinery, “Code Code Code The Code ACM Code of Ethics and Professional Conduct,” *ACM Code of Ethics and Professional Conduct*, 2018, doi: https://doi.org/10.1145/3274591.
[8]IEEE, “IEEE Code of Ethics,” *ieee.org*, Jun. 2020. https://www.ieee.org/about/corporate/governance/p7-8.html
[10]“Reddit - Dive into anything,” *Reddit.com*, 2021. https://www.reddit.com/r/battlebots/comments/q6udz7/practicing_digital_art_by_drawing_the_robot/ (accessed Mar. 06, 2025).

7. Appendix

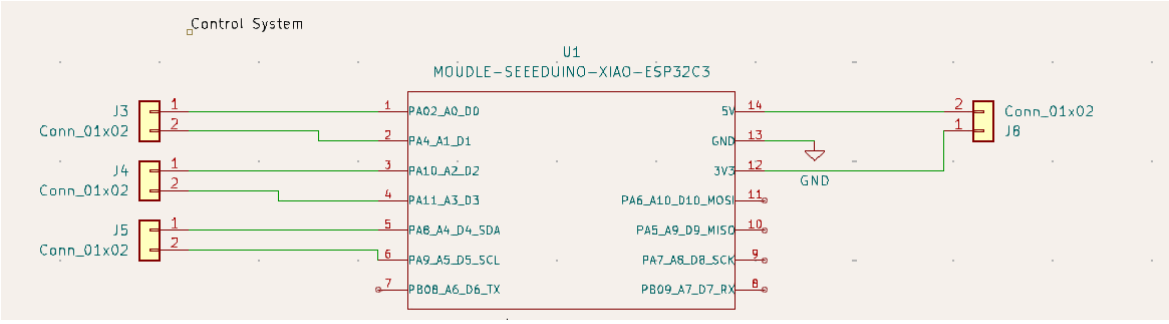

Figure 5. PCB Schematic of ESP32-C3

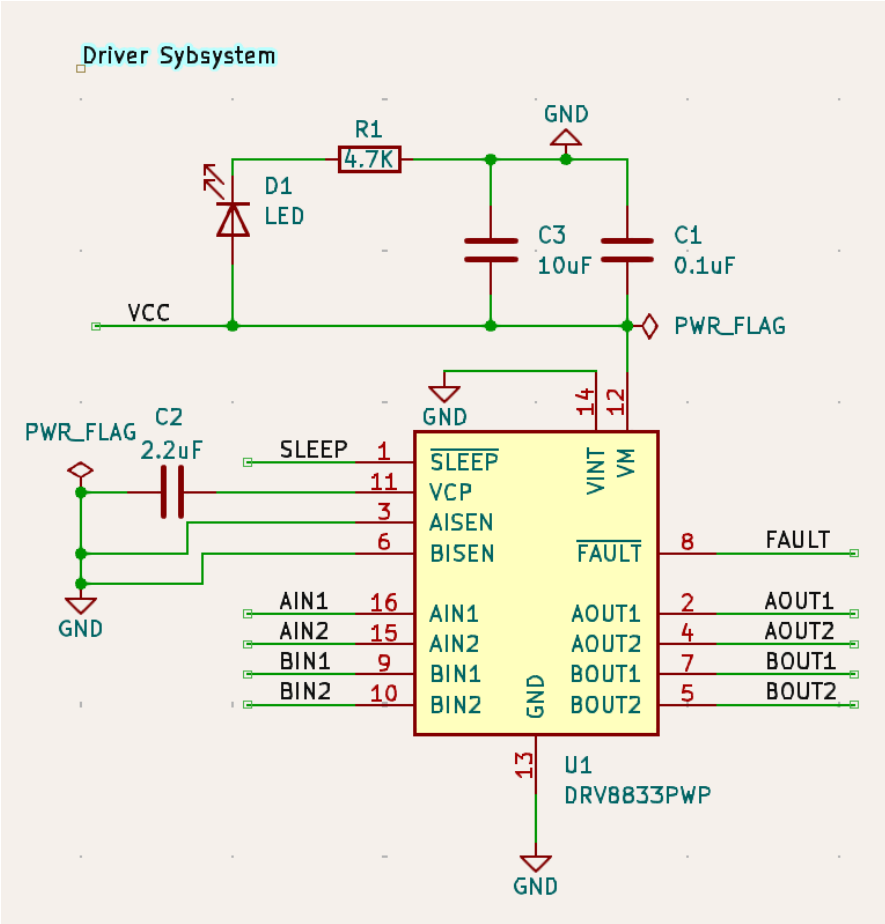

Figure 6. PCB Schematic of DRV8833 Motor Driver

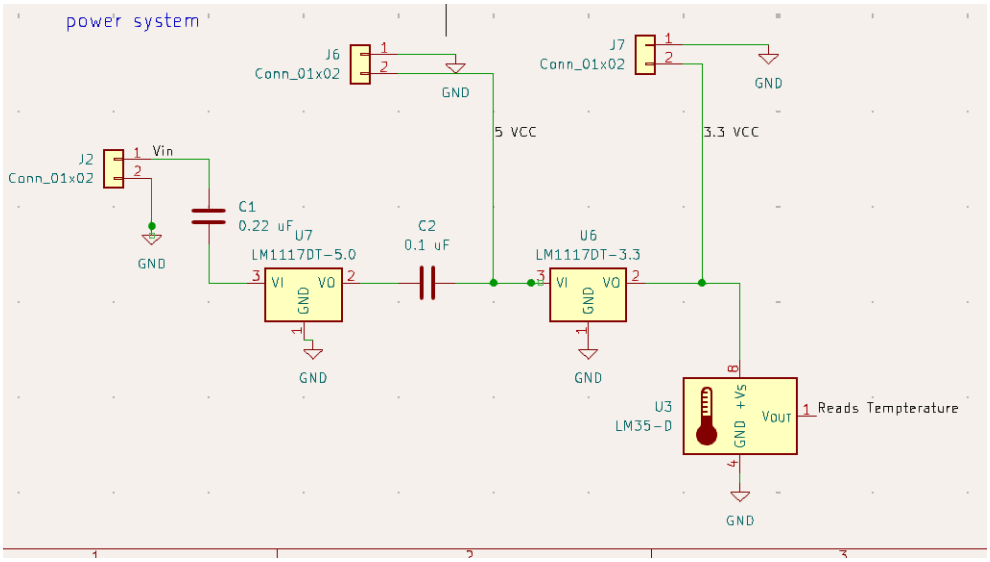

Figure 7. PCB Schematic of Power Subsystem

```
#include <WiFi.h>
#include <WebServer.h>

const char* ssid = "ESP32C3_AP";
const char* password = "password123";

// Create a web server on port 80
WebServer server(80);

bool motorsRunning = false;


// Motor control pins
const int motorPin1 = 2; // left motor
const int motorPin2 = 3; // left motor
const int motorPin3 = 4; // right motor
const int motorPin4 = 5; // right motor
const int motorPin5 = 6; // tombstone
const int motorPin6 = 7; // tombstone

// Motor speed (0-255)
int motorSpeed = 255;      // Full speed for motors 1 and 2
int motorSpeedSlower = 245; // Slightly slower speed for motors 3 and 4
int tombstoneSpeed = 80;

```

Figure 8. Arduino Code of DC Motor Setup

```
52   // Forward command
53   server.on("/forward", []() {
54     analogWrite(motorPin1, motorSpeed);  // Motor 1 forward (left)
55     analogWrite(motorPin2, 0);           // Motor 2 forward (left)
56     analogWrite(motorPin3, motorSpeedSlower);  // Motor 3 forward (right)
57     analogWrite(motorPin4, 0);           // Motor 4 forward (right)
58     motorsRunning = true;
59
60     server.send(200, "text/plain", "Moving forward");
61   });
62
63   // Backward command
64   server.on("/backward", []() {
65     analogWrite(motorPin1, 0);           // Motor 1 backward (left)
66     analogWrite(motorPin2, motorSpeed); // Motor 2 backward (left)
67     analogWrite(motorPin3, 0);           // Motor 3 backward (right)
68     analogWrite(motorPin4, motorSpeedSlower);  // Motor 4 backward (right)
69     motorsRunning = true;
70
71     server.send(200, "text/plain", "Moving backward");
72   });
```

Figure 9. Arduino Code of Forward/Backward Movements

```
74   // LEFT command
75   server.on("/left", []() {
76     analogWrite(motorPin1, 0);           // Motor 2 backward (left)
77     analogWrite(motorPin2, motorSpeed); // Motor 3 backward (left)
78     analogWrite(motorPin3, motorSpeedSlower); // Motor 4 forward (right)
79     analogWrite(motorPin4, 0);           // Motor 5 forward (right)
80     motorsRunning = true;
81     server.send(200, "text/plain", "Moving left");
82   });
83
84   // RIGHT command
85   server.on("/right", []() {
86     analogWrite(motorPin1, motorSpeed); // Motor 2 forward (left)
87     analogWrite(motorPin2, 0);           // Motor 3 forward (left)
88     analogWrite(motorPin3, 0);           // Motor 4 backward (right)
89     analogWrite(motorPin4, motorSpeedSlower); // Motor 5 backward (right)
90     motorsRunning = true;
91     server.send(200, "text/plain", "Moving right");
92   });
93
94   // STOP command
95   server.on("/stop", []() {
96     analogWrite(motorPin1, 0);
97     analogWrite(motorPin2, 0);
98     analogWrite(motorPin3, 0);
99     analogWrite(motorPin4, 0);
100    motorsRunning = false;
101
102    server.send(200, "text/plain", "Stopped all motors");
103  });
```

Figure 10. Arduino Code of Left/Right/Stop Movements

```
105   server.on("/tombstone_on", []() {
106     analogWrite(motorPin5, tombstoneSpeed); // Tombstone motor forward
107     analogWrite(motorPin6, 0);
108     motorsRunning = true;
109     server.send(200, "text/plain", "Tombstone spinning slowly");
110   });
111
112   server.on("/tombstone_off", []() {
113     analogWrite(motorPin5, 0);
114     analogWrite(motorPin6, 0);
115     motorsRunning = false;
116
117     server.send(200, "text/plain", "Tombstone stopped");
118   });
```

Figure 11. Arduino Code of Weapon On/Off Commands

```
1   import requests
2
3   ESP32_IP = "192.168.4.1"
4
5   def send_command(command):
6       url = f"http://{ESP32_IP}/{command}"
7       try:
8           response = requests.get(url, timeout=3)
9           print(f"[{command.upper()}] {response.text}")
10      except requests.exceptions.RequestException as e:
11          print(f"Error sending {command}: {e}")
12
13  if __name__ == "__main__":
14      while True:
15          cmd = input("Enter command (forward, backward, left, right, stop, tombstone_on, tombstone_off, exit): ").strip().lower()
16          if cmd in ["forward", "backward", "left", "right", "stop", "tombstone_on", "tombstone_off"]:
17              send_command(cmd)
18          elif cmd == "exit":
19              print("Exiting...")
20              break
21          else:
22              print("Invalid command. Try again.")
23
```

Figure 12. Python Code for Testing

Total Estimation		
Total Filament:	76.40 m	231.54 g
Model Filament:	76.40 m	231.54 g
Cost:	5.79	
Prepare time:	7m20s	
Model printing time:	3h44m	
Total time:	3h51m	

Figure 13. 3D Printing Estimation

Figure 14. 5 Volts Input Measurement

Figure 15. 3.3 Volts Input Measurement