Antweight Battlebot Design Document

Deepika Agrawal
Megha Esturi
Ishanvi Lakhani

Final Report for ECE 445, Senior Design, Fall 2024
TA: Surya Vasanth

10th December 2024
Project No. 37

Abstract

This project presents the design and development of an antweight battle bot divided into four primary subsystems: power, control, drivetrain, and a destabilizing ramp. The power subsystem utilizes a 9V battery with step-down converters to supply stable voltage to the motors, motor drivers, and ESP-32 microcontroller. The control subsystem enables wireless operation via Wi-Fi, ensuring precise communication between the bot and a user's PC. The drivetrain subsystem facilitates smooth mobility through carefully calibrated motors and motor drivers. Initially conceived as a flipper arm, the destabilizing subsystem was refined into a steep ramp capable of flipping or destabilizing opponent bots upon impact, providing a robust and simplified solution. Throughout the semester, each subsystem was optimized to meet functional requirements, resulting in reliable voltage regulation, enhanced communication responsiveness, and effective destabilization performance. This cohesive integration ensures the bot achieves its intended functionality and competitive goals.

Contents

1. Introduction

Our passion for robotics inspired us to take on the challenge of designing a battlebot, combining creativity and technical skills to build a competitive, functional robot. This project aims to create a remote controlled battlebot capable of competing in an arena while adhering to strict competition guidelines. The bot is designed to be agile, durable, and equipped with offensive and defensive mechanisms. We designed and built a battlebot entirely from PLA plastic, weighing under 2 pounds. The bot connects wirelessly to a PC via Wi-Fi for seamless control. Its defense mechanism features a ramp at the front, designed to destabilize and potentially flip opponent bots during combat. The bot is highly maneuverable, capable of navigating forward, backward, and turning left or right with precision. By focusing on lightweight construction, strategic functionality, and reliable connectivity, our battlebot embodies agility and innovation, making it ready for competitive performance.

2. Design

2.2.1 Block Diagram

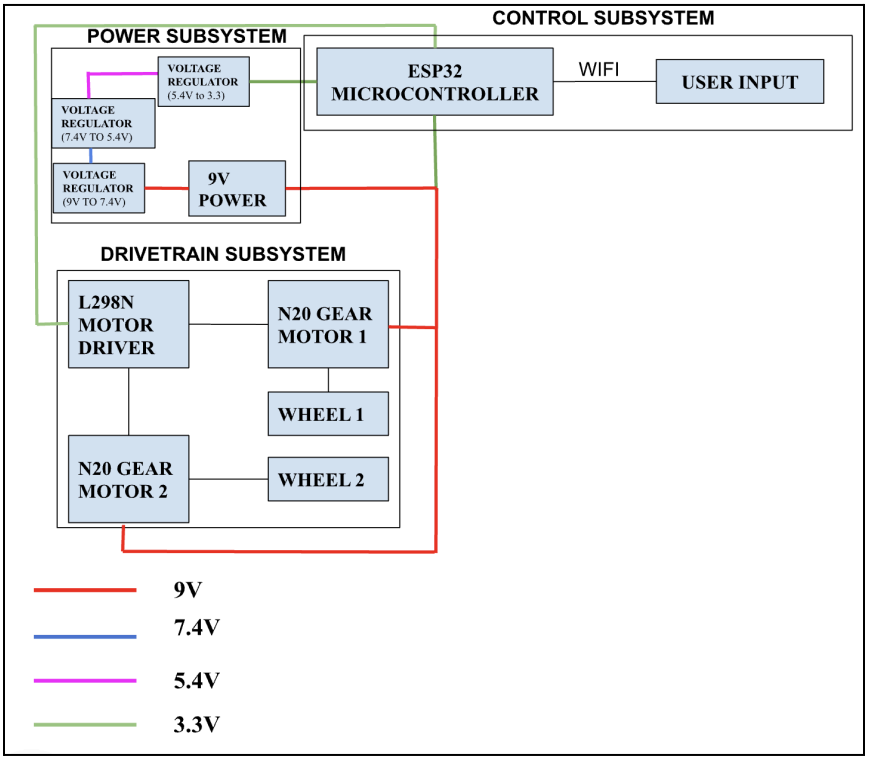

Figure 1. Block diagram containing each subsystem of the battle bot.

2.2.1 Design Procedure

Our design decisions for each subsystem were guided by functionality, stability, and efficiency.

For the drivetrain subsystem, we initially started with a two-wheel design, placing the wheels at the back of the bot. During testing, we discovered that this configuration left the bot unbalanced and unable to carry its weight effectively. To address this, we added two additional wheels, creating a four-wheel design that significantly improved the bot's stability and ensured that it could handle its weight without compromising mobility.

The destabilizing subsystem was originally designed to include a flipper arm capable of lifting and flipping opponent bots. However, we realized that generating sufficient torque for this functionality would require a stronger motor, which would exceed the bot's 2 lb weight limit. As a result, we changed our design to include a steep ramp attached to the bot. This ramp takes advantage of the opponent bot's speed, causing it to destabilize or flip when it contacts the ramp.

This solution not only simplifies the design but also keeps it lightweight and within the weight constraint while achieving a similar strategic outcome.

For the power subsystem, we selected a 9V battery as our power source due to its affordability and availability. The design includes multiple step-down converters to regulate the voltage for different components. For instance, the ESP-32 requires a 3.3V input, while the motor driver can take up to 9V. Stepping down the voltage in stages—from 9V to 7.4V, then to 5.4V, and finally to 3.3V—minimizes heat generation and prevents the chips from overheating. For example, directly stepping down from 7.4V to 3.3V could result in significant heat, potentially exceeding the temperature threshold of the chip. This staged approach ensures efficient and safe power delivery to all components.

Finally, for the control subsystem, we implemented a client-server Wi-Fi communication system. The PC acts as the client, sending control signals to the ESP-32, which is connected via Wi-Fi. This setup allows for precise, wireless control of the bot, ensuring responsiveness and ease of operation during use.

By exploring alternative approaches and iterating on our design, we developed a system that balances performance, stability, and simplicity while staying within project constraints.

2.2.2 Design Details

2.2.2.1 Hardware

The hardware design incorporates multiple step-down converters and an L298N motor driver. For the power subsystem, we chose a 9V battery. The design features step-down converters to regulate the voltage for various components. For example, the ESP-32 requires a 3.3V input, while the motor driver can handle up to 9V. Voltage is stepped down in stages—from 9V to 7.4V, then to 5.4V, and finally to 3.3V. This staged approach reduces heat generation and helps prevent the chips from overheating. Stepping down directly from 7.4V to 3.3V could generate excessive heat, potentially surpassing the chip's temperature tolerance.

We decided to use the AZ1117D-ADJ chip since it was easily available in the Electronic Services Shop (E-shop). Looking at the datasheet, this is the set of equations we needed to follow to step down to the voltage we wanted:

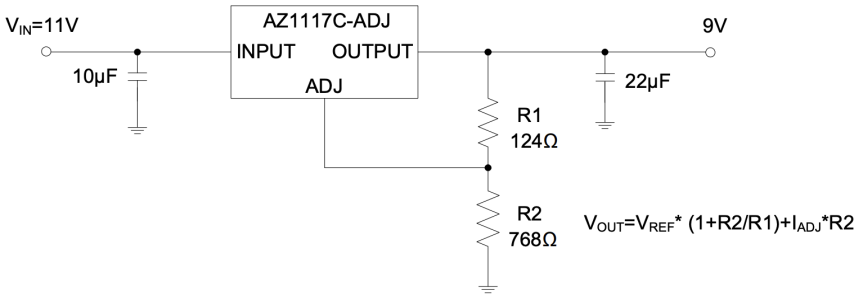

Figure 2. The schematic of a voltage converter from the AZ1117C datasheet. [2]

Using equation Vout = Vref * (1+R2/R1) + Iadj*R2

Vref was typically 1.25V as per the datasheet. Iadj was negligible so we removed that term. We kept R1 fixed as 1000 ohms as these resistors were also available at the E-shop)

Therefore, the equation essentially became Vout = 1.25(1+R2/1000) + negligible

Therefore R2 = (Vout/1.25 - 1) * 1000

The following is converting from 9V to 7.4V, so R2 = (7.4/1.25 - 1) * 1000 = 6200 ohms

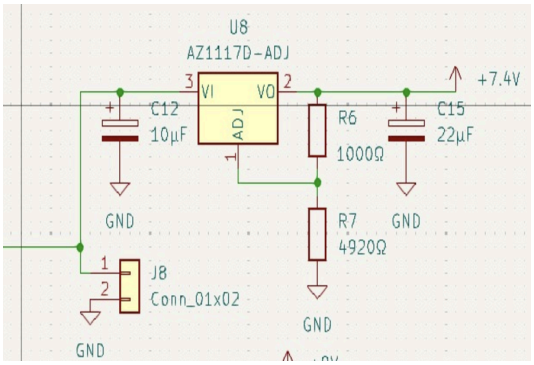

Figure 3. Our voltage converter schematic for 9V to 7.4V

The following is converting from 7.4V to 5.4V, so R2 = (5.4/1.25 - 1) * 1000 = 3320 ohms

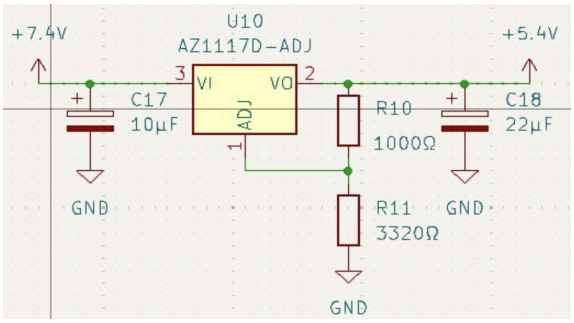

Figure 3. Our voltage converter schematic for 7.4V to 5.4V

The following is converting from 5.4V to 3.3V, so R2 = (3.3/1.25 - 1) * 1000 = 1640 ohms

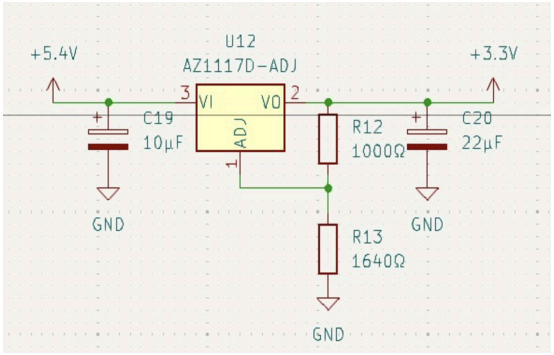

Figure 3. Our voltage converter schematic for 5.4V to 3.3V

Finally, for the L298N motor driver, we found the application circuit on the datasheet and used it to make our L298N motor driver bridge. We had 2 motor drivers each controlling 2 wheels. [6]

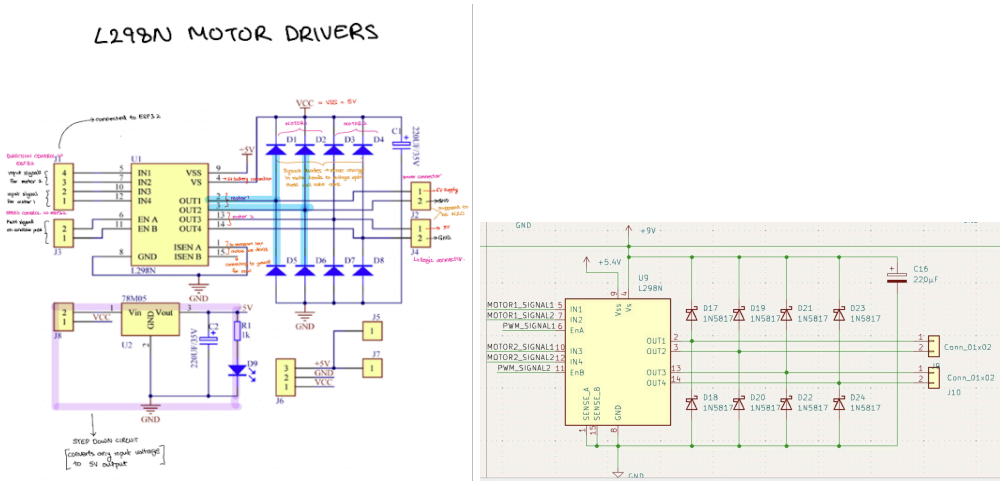

Figure 4. The schematic for our L298n motor drivers.

Additionally, we built a programming circuit for the ESP-32, enabling us to boot the ESP-32 and upload code to it. To achieve this, we followed the guidelines on the ECE 445 wiki page.

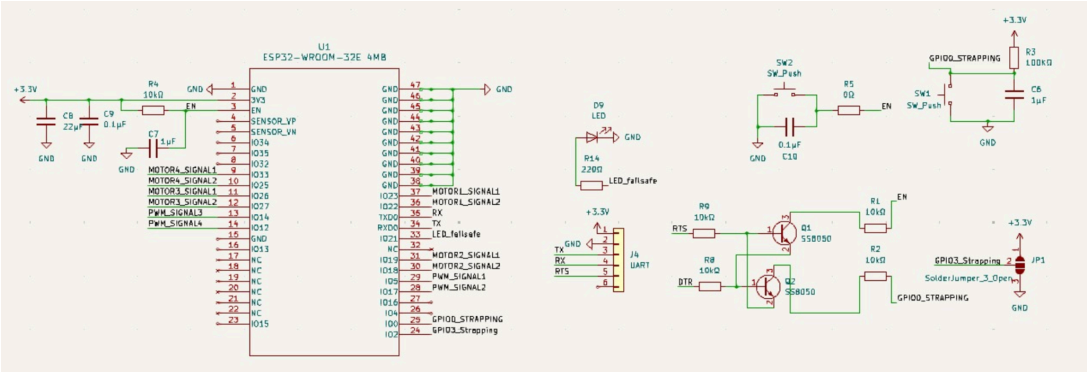

Figure 5. The schematic of of our ESP32

2.2.2.2 Software

The main software platforms used were an Arduino IDE and VS Code. The Arduino code, written in C for the ESP-32 microcontroller, implements the wireless control system and motor management for the battle bot.

The ESP-32 is configured to connect to a Wi-Fi network for communication with a PC, which we used a hotspot to do. An asynchronous web server handles the commands sent over WebSocket, ensuring low-latency control.

The software is structured into the following key modules:

1. Wi-Fi Connectivity [4]
   The ESP-32 operates in station mode, connecting to a specific Wi-Fi network using the provided SSID and password.

```
void connectToWiFi() {
  WiFi.mode(WIFI_STA); // Set ESP32 to Station mode
  WiFi.begin(ssid, password); // Connect to hotspot

  Serial.println("Connecting to WiFi...");
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.print(".");
  }

  Serial.println("\nConnected to WiFi!");
  Serial.println("ESP32 IP Address: " + WiFi.localIP().toString());
}
```

Figure 6. The code that allows us to connect to wifi with the ESP-32

2. Motor Control [1]
   For the motor control, we have six GPIO pins that control the four motors via an H-bridge motor driver. Two pins, per motor set, control each motor's direction, while an enable pin regulates the speed using a pulse-width modulation (PWM) signal

3. Commands and Actions [1]
   Commands are received via WebSocket and interpreted as single characters representing actions:
   - `'U'`: Accelerate forward
   - `'D'`: Accelerate backward
   - `'L'`: Turn left
   - `'R'`: Turn right
   - `'S'`: Immediate stop

   Each command triggers the corresponding motor control function, ensuring smooth transitions and precise movements.

The acceleration function occurs when the up arrow is pressed. In order to accelerate forward function, we needed all the wheels to have a consistent high to low output. We also made it so that the more the character is pressed, the PWM increases, allowing the bot to go faster.

```
void accelerateForward() {

  digitalWrite(motor1Pin1, LOW);
  digitalWrite(motor1Pin2, HIGH);
  digitalWrite(motor2Pin1, LOW);
  digitalWrite(motor2Pin2, HIGH);

  if (currentDutyCycle < maxDutyCycle) {
    currentDutyCycle += accelerationStep;
  }
  digitalWrite(enable1Pin, currentDutyCycle);
  digitalWrite(enable2Pin, currentDutyCycle);
}
```

Figure 7. The code that shows the function for forwards acceleration

The accelerate backward is also the same as the accelerate forward, however, the motor pin outputs were revered for high and low outputs.

```
void accelerateBackward() {
  digitalWrite(motor1Pin1, HIGH);
  digitalWrite(motor1Pin2, LOW);
  digitalWrite(motor2Pin1, HIGH);
  digitalWrite(motor2Pin2, LOW);

  if (currentDutyCycle < maxDutyCycle) {
    currentDutyCycle += accelerationStep;
  }

  digitalWrite(enable1Pin, currentDutyCycle);
  digitalWrite(enable2Pin, currentDutyCycle);
}
```

Figure 8. The code that shows the function for backwards acceleration

The turn right and turn left are essentially the same, but they allow for only one set of wheels, either right or left, to run at a time. This includes making sure that only one set of wheels and one pin in that set is put as high output.

```
void turnLeft() {
  digitalWrite(motor1Pin1, LOW);
  digitalWrite(motor1Pin2, HIGH);
  digitalWrite(motor2Pin1, LOW);
  digitalWrite(motor2Pin2, LOW);

  digitalWrite(enable1Pin, turnDutyCycle / 2);
  digitalWrite(enable2Pin, turnDutyCycle);
}
```

```
void turnRight() {
  digitalWrite(motor1Pin1, LOW);
  digitalWrite(motor1Pin2, LOW);
  digitalWrite(motor2Pin1, LOW);
  digitalWrite(motor2Pin2, HIGH);

  digitalWrite(enable1Pin, turnDutyCycle);
  digitalWrite(enable2Pin, turnDutyCycle / 2);
}
```

Figure 9. The code that shows the function for left and right acceleration

The immediate stop function was put in to make sure that the bot would be able to not hit walls if it was in line to hit one. For this function, all motor outputs needed to be low to turn off motors.

```
void immediateStop() {
    Serial.println("Immediate stop triggered");
    currentDutyCycle = 0; // Reset duty cycle
    digitalWrite(motor1Pin1, LOW);
    digitalWrite(motor1Pin2, LOW);
    digitalWrite(motor2Pin1, LOW);
    digitalWrite(motor2Pin2, LOW);
    // turn off motors
    digitalWrite(enable1Pin, 0);
    digitalWrite(enable2Pin, 0);
}
```

Figure 10. The code that shows the function for stopping

4. Failsafe Mechanism

   A failsafe mechanism ensures that the motors all stop if no command is received within a 1-second timeout. This makes it so that no motions that were not intended would stop occurring so a new command can be passed in. The system also regularly checks the Wi-Fi connection, automatically reconnecting if the connection drops.

On the client side, we created a website to interface with the ESP-32 microcontroller. The bot controller is a web-based interface designed to send directional commands to an ESP-32 microcontroller via WebSocket communication. Key features include:

1. Responsive Interface + Visual Feedback
   We created a visually appealing layout styled with CSS, featuring a gradient background and interactive buttons for directional controls. When a user presses a button, the appearance of the button changes, providing immediate user feedback and visual confirmation.

2. WebSocket Communication [3]
   The WebSocket establishes a persistent connection with the ESP-32, supporting real-time command transmission. It also automatically attempts to reconnect in case of a connection loss.

3. Command Mapping
   Since there are multiple commands, buttons trigger specific commands to be sent to the arduino code, which then processes what to do on its side. The keyboard support enables intuitive control using arrow keys and the ‘S’ key for Stop.

4. Feedback Mechanisms
   By incorporating logging and error messages, we were able to troubleshoot and debug easier.

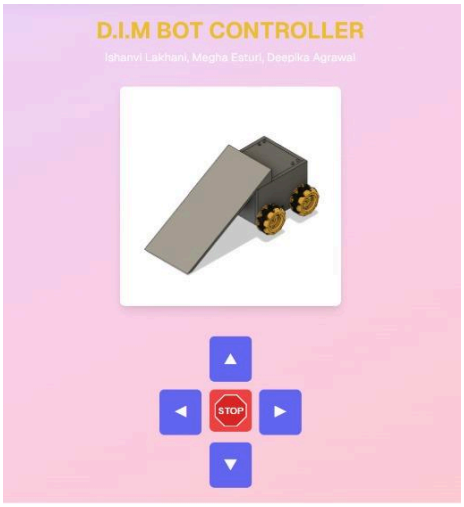

Figure 11. The website that acts as a remote control for our battle bot

The software design complements the hardware subsystem, allowing for real-time, wireless operation of the battle bot.

2.2.3 Verification

For testing the completed project and its major subsystems, we evaluated each subsystem to ensure it met the design goals. Below are the testing methodologies, data, and results for each subsystem.

2.2.3.1 DriveTrain Subsystem

The drivetrain subsystem had two primary requirements:

1. The motors and motor drivers must operate efficiently with a 9V battery.
2. The wheels must enable the robot to achieve an RPM of approximately 50–460.

To test the first requirement, we used a multimeter in the lab to measure the voltage across various components under different operating conditions. We observed stable voltage levels throughout the system, confirming that the motors, motor drivers, and other components functioned efficiently with a 9V power supply.

To test the second requirement, we calculated the actual RPM of the motors by conducting a speed test. The battlebot was made to travel a distance of 10 meters, and the time taken was recorded. Using the wheel diameter of 48 mm (0.048 meters), we calculated the wheel's circumference and applied the following formulas to derive the actual RPM:

$\text{Circumference} = \pi \times \text{Diameter} = \pi \times 0.048\text{m} \approx 0.1508\text{m}$
$\text{Speed} = \text{Time} / \text{Distance}$

Rotations per Second = Circumference / Speed
RPM = Rotations per Second × 60

By applying these calculations to our recorded data, we compiled the following table comparing the theoretical and actual RPM values:

Table 1. Theoretical vs. actual RPM values for our wheels

Weight Added	Distance (meters)	Time (sec)	RPM (Theoretical)	RPM (Actual)
None	10	22.1	450	180
1.8lbs	10	49.74	450	80

2.2.3.2 Power Subsystem

For the power subsystem, the primary requirement was:

1. The 9V battery must power the bot effectively for 3–5 minutes, which we estimated to be the typical duration of a battle.

To verify this, we conducted a test where the battlebot was run continuously, and we monitored the voltage supplied by the battery over time. Using a multimeter, we measured the voltage drop during operation and observed a specific pattern as the battery discharged. A critical voltage threshold of 6V was set, as this is the ideal voltage required for the motors to function optimally. The goal was to ensure that the battery could maintain a voltage above this threshold for the entire duration of the expected battle time.

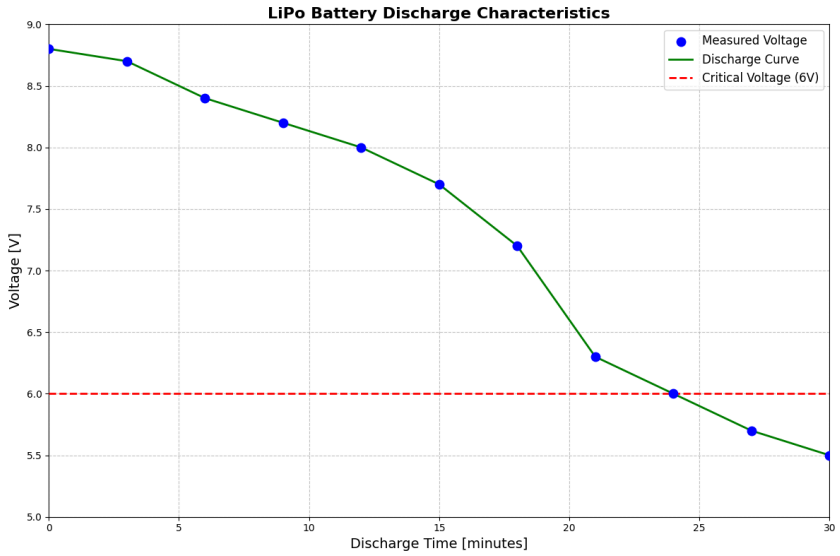

Figure 12. A graph showing the discharge of our 9V battery over a period of 25 minutes

From the data recorded during testing, the battery demonstrated the ability to supply at least 6V for a sufficient period, confirming its suitability for the competition. The graph of voltage over time clearly shows that the power subsystem will sustain the bot's performance during a 3–5 minute match, meeting the requirements effectively.

2.2.3.3 Destabilizing Subsystem

For the destabilizing subsystem, the primary requirement was:

1. Generate sufficient destabilizing force to unbalance and potentially flip the opponent robot.

This was critical to ensure the bot could effectively disrupt the movement of opposing bots during a match.

To test this requirement, we experimented with various objects, including a Lego car, our ECE 110 car, and other objects weighing 2 lbs or slightly more. As demonstrated in our final video submission, the ramp consistently destabilized these objects upon contact, causing them to lose balance or skid. The results confirmed that the flipper subsystem successfully meets the requirement of destabilizing objects weighing 2 lbs or more, making it an effective part of the bot's design.

2.2.3.4 Control Subsystem

1. The battlebot should stop operating if the Wi-Fi connection is lost as a safety precaution (kill switch).

2. The battlebot should be able to receive commands and execute them up to a distance of 8.48 feet.
3. The latency should be between 50–100 tenths of a second for responsive control.

To meet the first requirement, we implemented a stop button that allows the bot to halt operations manually without losing Wi-Fi connection. Additionally, we tested the system by disconnecting the bot from Wi-Fi, which resulted in the bot immediately ceasing movement. This confirmed that the kill switch worked effectively as a safety feature.

For the second requirement, the minimum communication distance was calculated to be 8.48 feet, based on the arena size of 6 feet by 6 feet. To ensure reliability, we tested the bot's communication capability up to 10 meters. The bot successfully received and executed commands at this distance, demonstrating that it exceeded the required range. This functionality was also captured in our final video submission.

Finally, for the third requirement, we measured command latency by printing it on the web interface. For each command (straight, left, right, and back), we recorded latency values six times, collecting 24 data points in total. The average latency was calculated to be approximately 58 tenths of a second, which is well within the required range of 50–100 tenths of a second. These results confirm that the control subsystem meets all specified requirements.

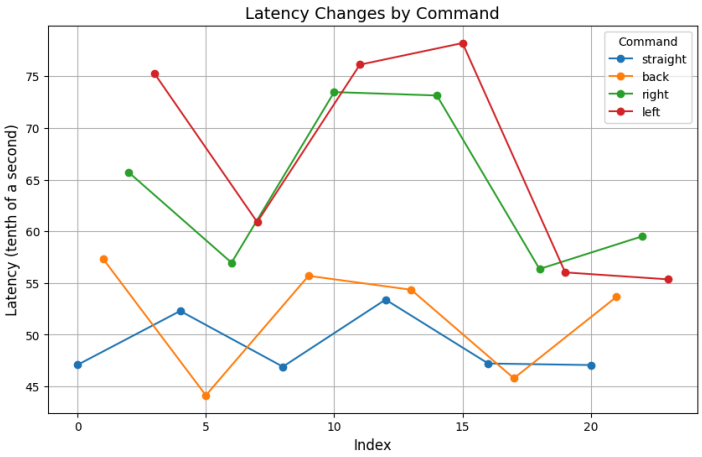

Figure 13. A graphical representation of the latencies measured between various different PC commands and movement of the battle bot

3. Cost

For this project, we used the $150 budget provided by the ECE department. However, due to multiple design changes, we purchased additional parts out-of-pocket. Electronics components

were obtained free of charge from the Eshop, and the metal wheel connectors were sourced at no cost from the Machine shop.

If the project were to be commercially viable, bulk-purchasing parts could significantly reduce costs. For example, components like motors and connectors, if bought in bulk, could lower the per-unit cost. This could reduce the overall production cost, making the design more scalable for mass production. By estimating costs in both a project-specific and commercial context, we gained insights into the economic feasibility of the design.

3.1 Parts

Table 2. A table showing the cost breakdown of all the parts we ordered

Part	Quantity	Manufacturer	Cost Per ($)	Cost Total ($)
Connectors	1	Amazon	5.98	5.98
USB Uart	1	Amazon	9.78	9.78
Resistor - 332 ohms (311-332LRCT-ND)	2	DigiKey	0.10	0.20
Resistor - 360 ohms (RMCF1206JT360RCT-ND)	2	DigiKey	0.10	0.20
Resistor 324 Kohms (311-324KCRCT-ND)	10	DigiKey	0.02	0.15
Capacitor - 4.7pF (311-1624-6-ND)	10	DigiKey	0.03	0.26
Step Down Convertor (AZ1117D-ADJTRE1DICT-ND)	10	DigiKey	0.41	4.06
22 uF Polarized (493-6180-1-ND)	5	DigiKey	0.41	2.05
Step Down Convertor	2	Digikey	0.54	1.08

220 ohm	2	DigiKey	1.06	2.12
1.64 kOhms resistor (2019-RN73R1ETTP1641B25CT-ND)	3	DigiKey	0.19	0.57
3.32 kOhms (RMCF0402FT3K32CT-ND)	3	DigiKey	0.09	0.27
4.9 kOhms (541-3072-1-ND)	3	DigiKey	8.08	8.08
LED surface mount (UHD1110-FKA-CL1A13R3Q1BBQFM3CT-ND)	3	DigiKey	0.09	0.27
22 uF capacitor (1276-1100-1ND)	3	DigiKey	0.11	0.33
ESP32-WROOM-32E-4MB (1965-ESP32-WROOM-32E-N4CT-ND)	2	DigiKey	2.50	5
Motor Chip (497-1395-5-ND)	2	DigiKey	10.61	21.22
10 uF capacitor (732-8295-1-ND)	5	DigiKey	0.19	0.95
Step Up Convertor (LT1615ES5#TRPBFCT-ND)	5	DigiKey	0.10	0.50
Resistor - 1M ohms (311-1.00MLRCT-ND)	5	DigiKey	0.10	0.50
Capacitor - 10 uF (1276-1096-1-ND	50	DigiKey	0.04	1.75

)				
Step Down Convertor (AZ1117D-ADJTRE1DICT-ND)	10	DigiKey	0.36	3.64
Step Up Convertor (LT1615ES5#TRPBFCT-ND)	10	DigiKey	7.69	76.90
Resistor - 100 Ohm (311-100LRCT-ND)	10	DigiKey	0.01	0.09
Resistor - 332 Ohms (311-332LRCT-ND)	5	DigiKey	0.10	0.50
Capacitor - 4.7 pF (311-1624-1-ND)	10	DigiKey	0.03	0.29
PLA Filament 1.75mm	1	Amazon	13.99	13.99
Gear Motors	3	Amazon	17.59	52.77
9-Volt Battery Duracell	5	Target	4.80	24.00
URGENEX 2000mAh 7.4 V Li-ion Battery	1	Amazon	19.99	19.99
150Pcs 1N5819 Schottky Diode	1	Amazon	7.99	7.99
Century Spring C-143	1	Amazon	6.66	6.66
2Pcs DC 12V 2000RPM GA12 N20 High SpeedMotor	1	Amazon	13.99	13.99

4pcs BOJACK L298N Motor DC Dual H-Bridge Motor Driver	1	Amazon	9.89	9.89
1 Pair Mecanum Wheels	2	Amazon	9.89	19.78
[4-Pack] MG996R 55g Metal Gear Torque Digital Servo	1	Amazon	17.99	17.99
TOTAL				$333.79

3.2 Labor

Each team member contributed an average 10 hours per week over a 10-week project timeline, amounting to 100 hours per person. Using an ideal hourly salary of $22, the labor cost for each partner was calculated using the formula: ideal salary (hourly rate) * actual hours spent * 2.5. For one partner, the cost would be 22 * 100 * 2.5 = USD $5500. This for three partners is about USD $16500.

4. Conclusion

To conclude, the design of our battle bot incorporates a lightweight, durable chassis, a reliable drivetrain, and a ramp-style destabilizing mechanism, all powered by an efficient control and power system. This project was a rewarding experience where we not only met our goals but also had a lot of fun and learned valuable skills along the way.

4.1 Accomplishments

We successfully met all the high-level requirements for our bot, including maintaining a total weight under 2 lbs, operating efficiently within a voltage range of 9 volts, and achieving a Wi-Fi communication range of 8 to 25 feet with a response time of 50 to 100 tenths of a second. Additionally, we fulfilled the specific requirements for each subsystem, ensuring stable and effective performance across all aspects of the bot's operation.

4.2 Uncertainties

There are certain things within our project that did not work as we expected them to. For example, we were not able to use our PCB for our final product, as we included voltage converters on it, and they did not properly convert our battery voltage to different voltages. Therefore, we had to use separate 3.3 volt batteries along with our 9V battery. We could have worked on our PCB earlier to account for these issues. Additionally, our bot did not move as fast as we would've liked due to the motors and wheels that we used. Our n20 gear motors moved fast, but with the weight of the bot and the electrical component within, they were not able to handle the weight and moved a lot slower. We also used Mecanum wheels which are made for diagonal movement, which we did not have. This also slowed down the speed of our bot. If we had used different wheels and bigger motors, the weight of our bot would not have hindered the speed and movement of our bot.

4.3 Ethical Considerations

There are a few ethical considerations we kept in mind throughout this project. As stated in the IEEE Code of Ethics, there are several areas of consideration when it comes to lab ethics. These include safety, conflict avoidance, honesty, respect, privacy, and support. Our main goal was prioritizing the safety and welfare for all participants and complying with safety standards to minimize risks. Further, we used sustainable PLA plastic to print our design, and we avoided potential risks while building our battlebot. We treated all team members and competitors with respect, avoiding discrimination, harassment, and injury. We also ensured that we had the necessary skills and sought help when needed in the lab, while making sure that all of our work was our own, and that we did not unfairly plagiarize others. Finally, our project falls under the IEEE Code of Ethics 1.2 as we created a project that integrates technologies that we can demo and compete with [5].

4.4 Future Work

If we were to continue this project, there are a few things we would incorporate for improved functionality and design. First, we would implement more commands into the wifi code so that the bot can move in additional directions, such as diagonal. Another consideration would be adding a boost function or even incorporating a sensor that avoids obstacles. In terms of the build of the battle bot, we would want to create a stronger offense/defense mechanism. Currently, there are a lot of ways an opponent can beat us, so we would want to have something that would allow us to properly defend ourselves, as well as attack an opponent.

5. References

[1] Alan et al., “ESP32 with DC motor - control speed and direction,” Random Nerd Tutorials, https://randomnerdtutorials.com/esp32-dc-motor-l298n-motor-driver-control-speed-direction/.

[2] AZ1117C, https://www.diodes.com/assets/Datasheets/AZ1117C.pdf.

[3] Droopyboy, “Arduino - websocket: Arduino Tutorial,” Arduino Forum, https://forum.arduino.cc/t/arduino-websocket-arduino-tutorial/1282126/3.

[4] Duncan et al., “Installing ESP32 in Arduino IDE (windows, mac OS X, linux),” Random Nerd Tutorials, https://randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-windows-instructions/.

[5] IEEE - IEEE Code of Ethics, https://www.ieee.org/about/corporate/governance/p7-8.html.

[6] L298_H_Bridge.PDF, https://www.sparkfun.com/datasheets/Robotics/L298_H_Bridge.pdf.

6. Appendix

Schematic

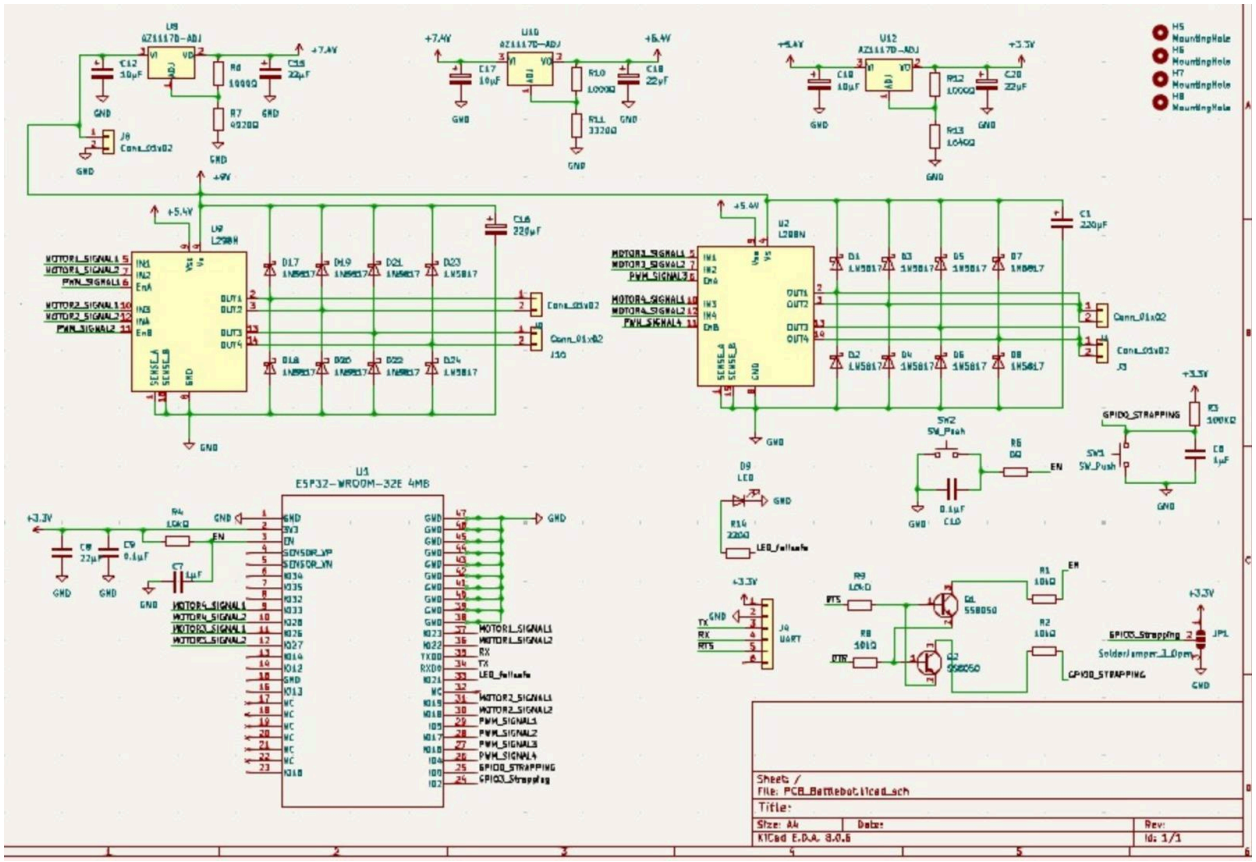

Figure 14. Our PCB design schematic

Arduino Code

```
#include <WiFi.h>

#include <ESPAsyncWebServer.h>

#include <AsyncTCP.h>

const char * ssid = "V30+_6936";
const char * password = "doraboots";
//// motor 1 pins
//int motor1Pin1 = 27;
//int motor1Pin2 = 26;
//int enable1Pin = 14;
//
//// motor 2 pins
int motor2Pin1 = 25;
int motor2Pin2 = 33;
int enable2Pin = 32;
// motor 1 pins
int motor1Pin1 = 18;
int motor1Pin2 = 16;
int enable1Pin = 17;
const int freq = 30000;
//const int pwmChannel1 = 0;
//const int pwmChannel2 = 1;
const int resolution = 8;
int dutyCycle = 0;
const int maxDutyCycle = 240; // capping voltage to motors at 6V
const int accelerationStep = 5;
const int decelerationStep = 5;
int currentDutyCycle = 0;
int turnDutyCycle = 280;
AsyncWebServer server(80);
AsyncWebSocket ws("/ws");
unsigned long lastCommandTime = 0; // Store last command timestamp
const unsigned long commandTimeout = 1000; // 1-second timeout
void onWebSocketEvent(AsyncWebSocket * server, AsyncWebSocketClient * client, AwsEventType
type, void * arg, uint8_t * data, size_t len);
void setup() {
  pinMode(motor1Pin1, OUTPUT);
  pinMode(motor1Pin2, OUTPUT);
```

```
  pinMode(motor2Pin1, OUTPUT);
  pinMode(motor2Pin2, OUTPUT);
  pinMode(enable1Pin, OUTPUT);
  pinMode(enable2Pin, OUTPUT);
  Serial.begin(115200);
  connectToWiFi();
  ws.onEvent(onWebSocketEvent);
  server.addHandler( & ws);
  server.begin();
  Serial.println("Started Server");
}
void connectToWiFi() {
  WiFi.mode(WIFI_STA); // Set ESP32 to Station mode
  WiFi.begin(ssid, password); // Connect to hotspot
  Serial.println("Connecting to WiFi...");
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.print(".");
  }
  Serial.println("\nConnected to WiFi!");
  Serial.println("ESP32 IP Address: " + WiFi.localIP().toString());
}
void accelerateForward() {
  digitalWrite(motor1Pin1, LOW);
  digitalWrite(motor1Pin2, HIGH);
  digitalWrite(motor2Pin1, LOW);
  digitalWrite(motor2Pin2, HIGH);
  if (currentDutyCycle < maxDutyCycle) {
    currentDutyCycle += accelerationStep;
  }
  digitalWrite(enable1Pin, currentDutyCycle);
  digitalWrite(enable2Pin, currentDutyCycle);
}
void accelerateBackward() {
  digitalWrite(motor1Pin1, HIGH);
  digitalWrite(motor1Pin2, LOW);
  digitalWrite(motor2Pin1, HIGH);
  digitalWrite(motor2Pin2, LOW);
  if (currentDutyCycle < maxDutyCycle) {
    currentDutyCycle += accelerationStep;
  }
  digitalWrite(enable1Pin, currentDutyCycle);
  digitalWrite(enable2Pin, currentDutyCycle);
}
```

```
void turnLeft() {
  digitalWrite(motor1Pin1, LOW);
  digitalWrite(motor1Pin2, HIGH);
  digitalWrite(motor2Pin1, LOW);
  digitalWrite(motor2Pin2, LOW);
  digitalWrite(enable1Pin, turnDutyCycle / 2);
  digitalWrite(enable2Pin, turnDutyCycle);
}
void turnRight() {
  digitalWrite(motor1Pin1, LOW);
  digitalWrite(motor1Pin2, LOW);
  digitalWrite(motor2Pin1, LOW);
  digitalWrite(motor2Pin2, HIGH);
  digitalWrite(enable1Pin, turnDutyCycle);
  digitalWrite(enable2Pin, turnDutyCycle / 2);
}
void decelerate() {
  if (currentDutyCycle > 0) {
    currentDutyCycle -= accelerationStep;
  }
  digitalWrite(motor1Pin1, LOW);
  digitalWrite(motor1Pin2, LOW);
  digitalWrite(motor2Pin1, LOW);
  digitalWrite(motor2Pin2, LOW);
  digitalWrite(enable1Pin, currentDutyCycle);
  digitalWrite(enable2Pin, currentDutyCycle);
}
void immediateStop() {
  Serial.println("Immediate stop triggered");
  currentDutyCycle = 0; // Reset duty cycle
  digitalWrite(motor1Pin1, LOW);
  digitalWrite(motor1Pin2, LOW);
  digitalWrite(motor2Pin1, LOW);
  digitalWrite(motor2Pin2, LOW);
  // turn off motors
  digitalWrite(enable1Pin, 0);
  digitalWrite(enable2Pin, 0);
}
char currentCommand = 'S'; // Default to 'S' for stop
void onWebSocketEvent(AsyncWebSocket * server, AsyncWebSocketClient * client, AwsEventType
type, void * arg, uint8_t * data, size_t len) {
  if (type == WS_EVT_CONNECT) {
    // check if client connected
    Serial.println("Client connected");
```

```
    } else if (type == WS_EVT_DISCONNECT) {
      Serial.println("Client disconnected");
      // check if client disconnected
    } else if (type == WS_EVT_DATA) {
      // if data is received
      if (len > 0) {
        char command = (char) data[0]; // Read only the first character --> for buffer management
        Serial.println("Received command: " + String(command));
        executeCommand();
        lastCommandTime = millis(); // Update last command timestamp
        currentCommand = command;
      }
    }
}
void executeCommand() {
   static char lastCommand = 'S'; // Store the last executed command
   if (currentCommand != lastCommand) {
      lastCommand = currentCommand; // Update the last executed command
      // Execute the function corresponding to the current command
      if (currentCommand == 'U') {
        accelerateForward();
        Serial.println(currentDutyCycle);
      } else if (currentCommand == 'D') {
        accelerateBackward();
        Serial.println(currentDutyCycle);
      } else if (currentCommand == 'L') {
        turnLeft();
      } else if (currentCommand == 'R') {
        turnRight();
      } else if (currentCommand == 'X') {
        decelerate();
      } else if (currentCommand == 'S') {
        immediateStop();
      }
   }
}
void failsafeCheck() {
   if (millis() - lastCommandTime > commandTimeout && currentCommand != 'S') {
      Serial.println("Failsafe triggered: Stopping motors");
      currentCommand = 'S'; // Set to stop
      currentDutyCycle = 0;
      // stop motors
      digitalWrite(enable1Pin, 0);
      digitalWrite(enable2Pin, 0);
```

```
    }
}
void loop() {
    ws.cleanupClients();
    failsafeCheck();
    // Non-blocking WiFi reconnection
    static unsigned long lastWiFiCheck = 0;
    if (millis() - lastWiFiCheck > 10000) { // Check every 10 seconds
        lastWiFiCheck = millis();
        if (WiFi.status() != WL_CONNECTED) {
            connectToWiFi();
        }
    }
}
```

Website Code

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>D.I.M BOT CONTROLLER</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 0;
            min-height: 100vh;
            display: flex;
            flex-direction: column;
            align-items: center;
            background: linear-gradient(to bottom right, #e9d5ff, #fbcfe8,
#fecaca);
        }
        .container {
            text-align: center;
            padding: 2rem;
        }
        h1 {
            font-size: 2.25rem;
            font-weight: bold;
            margin-bottom: 0.5rem;
            animation: color-change 5s infinite;
        }
        .names {
            color: white;
            font-size: 1.125rem;
        }
        .controls {
            display: grid;
            grid-template-columns: repeat(3, 1fr);
            gap: 0.75rem;
            width: 14rem;
```

```
        }
        .button {
            width: 100%;
            padding: 1.25rem;
            border-radius: 0.5rem;
            font-size: 1.5rem;
            border: none;
            cursor: pointer;
            transition: background-color 0.2s;
        }
        .arrow {
            background-color: #6366f1;
            color: white;
        }
        .arrow:hover {
            background-color: #4f46e5;
        }
        .arrow.active {
            background-color: #fbbf24;
            color: #1f2937;
        }
        .stop {
            grid-column: 2;
            grid-row: 2;
            padding: 0.5rem;
            background-color: #ef4444;
        }
        .stop:hover {
            background-color: #dc2626;
        }
        @keyframes color-change {
            0% { color: #f472b6; }
            25% { color: #60a5fa; }
            50% { color: #34d399; }
            75% { color: #fbbf24; }
            100% { color: #f472b6; }
        }
    </style>
</head>
<body>
```

```
    <div class="container">
        <h1>D.I.M BOT CONTROLLER</h1>
        <p class="names">Ishanvi Lakhani, Megha Esturi, Deepika
Agrawal</p>
    </div>

    <div class="controls">
        <button class="button arrow" id="up">▲</button>
        <button class="button arrow" id="left">◀</button>
        <button class="button stop" id="stop">STOP</button>
        <button class="button arrow" id="right">▶</button>
        <button class="button arrow" id="down">▼</button>
    </div>

    <script>
        let ws;

        // Function to connect to the WebSocket
        function connectWebSocket() {
            // ws = new WebSocket('ws://192.168.88.165/ws');     // DEV
BOARD
            ws = new WebSocket('ws://192.168.88.12/ws');         // ESP
CHIP 1

            ws.onopen = () => {
                console.log('Connected to WebSocket');
            };

            ws.onmessage = (event) => {
                console.log("Received message from ESP32: " + event.data);
            };

            ws.onerror = (error) => {
                console.error("WebSocket error:", error);
            };

            ws.onclose = (event) => {
                console.log("WebSocket closed, attempting to
reconnect...");
```

```
                setTimeout(connectWebSocket, 1000); // Retry connection
after 1 second
            };
        }

        // Initialize WebSocket connection
        connectWebSocket();

        // Function to send commands via WebSocket
        function sendCommand(command) {
            if (ws.readyState === WebSocket.OPEN) {
                ws.send(command);
            } else {
                console.log("WebSocket not connected. Command not sent:",
command);
            }
        }

        // Button click handlers
        const buttons = document.querySelectorAll('.button');
        buttons.forEach(button => {
            button.addEventListener('mousedown', () => {
                button.classList.add('active');
                const commandMap = {
                    up: 'U',
                    down: 'D',
                    left: 'L',
                    right: 'R',
                    stop: 'S'
                };
                sendCommand(commandMap[button.id]);
            });
            button.addEventListener('mouseup', () =>
button.classList.remove('active'));
            button.addEventListener('mouseleave', () =>
button.classList.remove('active'));
        });

        // Keyboard event listeners in the desired format
        document.addEventListener('keydown', (event) => {
```

```
            console.log('Key pressed:', event.key);
            if (event.key === 'ArrowUp') {
                sendCommand('U');
            } else if (event.key === 'ArrowDown') {
                sendCommand('D');
            } else if (event.key === 'ArrowLeft') {
                sendCommand('L');
            } else if (event.key === 'ArrowRight') {
                sendCommand('R');
            } else if (event.code === 'KeyS') {
                sendCommand('S');
            }
        });

        document.addEventListener('keyup', () => {
            buttons.forEach(button => button.classList.remove('active'));
        });
    </script>
</body>
</html>
```

Requirements and Verification Tables

DriveTrain Subsystem

Table 3. The requirements and verification table for our drivetrain subsystem

Requirements	Verification
The motors and motor drivers must operate efficiently with a 9V battery.	Measure the voltage supplied to the motor driver and the motors. Verify that the voltage is between 6 - 8.5V range during operation. 1. Attach a battery to the system 2. For every 5 minutes, check the voltage difference for the motors by using multimeter on the ends of the motors 3. For every 5 minutes, check the voltage between the motor driver by using multimeter on the ends of the motors 4. Confirm that the voltage across motors is greater than 6V 5. Confirm that the voltage across the motor driver is at least 5V
The wheels must enable the robot to move at a sufficient speed, achieving an RPM of approximately 50 - 460 .	Calculate the wheel's circumference using its diameter, then find the theoretical speed by multiplying the circumference by the motor's RPM and converting to meters per second. Finally, divide the desired travel distance by the speed to estimate the time required. 1. Find distance travelled by the wheels in one rotation (circumference) 2. Divide distance needed to be travelled (10m) with the distance covered by one rotation to find number of rotations required to cover distance 3. Record a video to find how many seconds it takes to get that many rotations. Find the speed of the wheels without any weight being added 4. Add the body of the bot which is about 1.2lbs. 5. Run the bot for a distance of 10m and find out how many seconds it takes to cover the distance

	6. Find the speed and RPM of the bot

Defense Subsystem

Table 4. The requirements and verification table for our defense subsystem

Requirements	**Verification**
Destabilizing Force: The should generate enough force to destabilize the opponent robot.	Place a weight equivalent to the opponent robot on one side of the battle bot. Make the battle bot run into the opponent and see if it gets destabilized or flipped. 1. Gather items that can roll, such as duct tape, water bottle, or another car 2. Make sure at least one item is around 2 lbs 3. Roll items at different speeds and check if item is destabilized or gets flipped

Power Subsystem

Table 5. The requirements and verification table for our power subsystem

Requirements	Verification
The subsystem must include a 9V battery to power the bot for about 3-5 minutes which is what we expect the duration of the battle to be.	Test the battlebot until it gets to 6V (voltage at which circuit is not useful) by keeping the power on and making the bot run around for the duration of the time and make sure it doesn't power down. 1. Get a new battery 2. Connect battery to the circuitry 3. Run the bot constantly 4. Stop every 3 minutes 5. Use multimeter to record the voltage across the battery 6. Graph the points

Control Subsystem

Table 6. The requirements and verification table for our control subsystem

Requirements	Verification
The latency should be between 50-100 tenths of a second for responsive control.	Measure the time between a button being pressed and the command being relayed. Test multiple times with multiple commands to get average latency across operation. 1. Use stopwatch on phone 2. Click on a singular command (up, down, left, right) 3. Start timer when clicked 4. When movement of the wheels is noticed, immediately stop timer 5. Mark time taken to move (use lap feature)
The battlebot should stop operating if WIFI connection is lost as a safety precaution (kill switch)	Turn off wifi, and observe if the battle bot stops operating 1. Turn on bot 2. Constantly run motors 3. Remove wifi connection 4. Visually check if the motors have stopped
The battlebot should be able to receive commands and execute them up to a distance of at least 8.48ft	Put the battlebot at incremental distances up to 10ft from the PC and check the operational condition. 1. Connect the setup of the battlebot 2. Move bot around 9 feet away a. If bot still runs, the requirement is met b. Else, check incremental distances to see the farthest the signal goes