Individual Progress Report

Autonomous Featherweight (30lb) Battlebot

Group# 43, Name: Qinghuai Yao

1. Introduction

1.1 Project Overview

Our project aims to enhance a previously manually controlled battlebot called “CRACK?” by integrating an autonomous control system with pure pursuit functionality. This team project focuses on transforming the battlebot into an intelligent system that can detect, track, and pursue an opponent robot without human intervention. The core objective is to implement a reliable "pure pursuing" functionality using computer vision and a pure pursuit algorithm. The subsystems of this project includes power monitoring, PWM I/O, IMU (movement detection), remote camera, and autonomy subsystem.

1.2 Individual responsibilities

As a member of the team, my primary responsibilities include designing the custom PCB housing the ESP32-S3 microcontroller, integrating the MPU6050 sensor for motion tracking, and developing the software interface that enables wireless communication between the robot and the overhead vision system. My contributions directly support the robot’s ability to receive positional data, interpret commands, and execute movements with precision.

2. Individual Design Work

2.1 Design considerations

My individual work focused on designing and implementing the core electronics and communication system for the autonomous BattleBot. Above is the schematic I design based on the example of the course website. [1] Jason modified the schematics and PCB design and got the latest version. We later realized that the chip used for signal processing in the programming circuit was extra and decided to get rid of it.

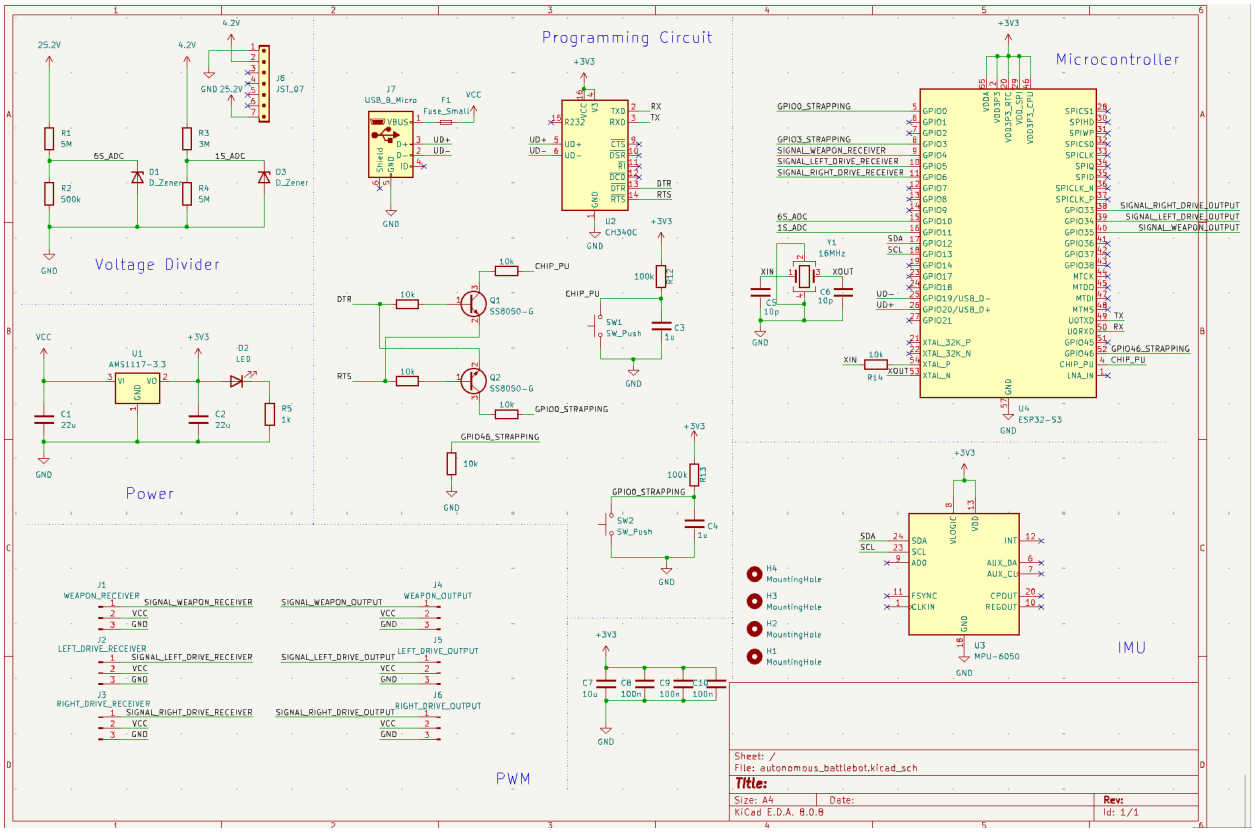

Figure 1: Schematics Design

Figure 1 shows the schematics design of our project. The voltage divider is the power monitoring system. The programming circuit is the circuit we will make changes. Also, we will add a voltage divider at the PWM circuit, because the output voltage of the receivers will be larger than 3.3 V which is the largest voltage ESP32 can hold. For the IMU, we are currently using MPU-6050. However, it's hard to buy a MPU6050 chip, so we decided to use LIS331, which is a great motion sensor for our project.

Another part of our project I focused on was communication systems and data transferring. One of the key design challenges in this project was selecting an efficient communication protocol between the overhead computer vision system and the ESP32-S3 on the robot. Initially, I implemented Bluetooth due to its simplicity and built-in support on the ESP32-S3. However, during early testing, it became evident that Bluetooth introduced significant latency, which was unacceptable for real-time control in a fast-paced battle environment. We then switched to a WiFi-based connection using the WebSocket protocol, hoping for improved speed and stability. While WebSocket offered better performance, it still introduced noticeable delay, especially under network congestion or unstable connections. After further testing and research, Jason discovered the ESP-NOW protocol, a low-latency, peer-to-peer wireless communication method specifically optimized for ESP32 devices. We tested ESP-NOW and found it to be significantly faster and more reliable for our needs, with near-instant data transfer and minimal packet loss. As

a result, we adopted ESP-NOW as our final communication protocol, which enabled smooth and responsive control of the BattleBot during autonomous operation.

2.2 Test and Verification

2.2.1. Data Detection (Voltage Divider and IMU Data)

To ensure reliable sensor data on the BattleBot, I conducted separate tests for the voltage divider and the MPU6050 IMU.

- **Voltage Divider Testing**:
  I validated the voltage divider circuit by applying known voltages through a variable power supply and measuring the ADC readings on the ESP32-S3. These readings were compared against theoretical values calculated using the resistor ratio. The measured values are accurate within +/- 0.1 V. The largest voltage input to the ESP32-S3 is not larger than 3.3 V, which protects the ESP32 chip.
- **IMU Testing**:
  The MPU6050 was tested to ensure accurate detection of motion. I recorded the accelerometer and gyroscope data while manually tilting and shaking the robot. Figure 2 shows the accelerations when the IMU is placed flat, and we can see that the z-axis acceleration is about 9.62 m/s^2, which is reasonable, because the IMU is not absolutely placed flat. Figure 3 shows the accelerations when the IMU is placed vertically. The y-axis acceleration is now about 9.6 m/s^2. This confirmed the IMU was suitable for motion verification and could potentially assist with future stability control.

```
Battery Level: 0.00V
Acceleration X: 0.25, Y: -0.45, Z: -9.62 m/s^2
Rotation X: -0.01, Y: -0.02, Z: 0.01 rad/s
Temperature: 24.99 degC

Battery Level: 0.01V
Acceleration X: 0.29, Y: -0.41, Z: -9.62 m/s^2
Rotation X: -0.01, Y: -0.02, Z: 0.01 rad/s
Temperature: 24.98 degC

Battery Level: 0.01V
Acceleration X: 0.27, Y: -0.42, Z: -9.64 m/s^2
Rotation X: -0.00, Y: -0.02, Z: 0.01 rad/s
Temperature: 25.02 degC
```

Figure 2: Voltage and IMU values output by ESP32-S3 (placed flat)

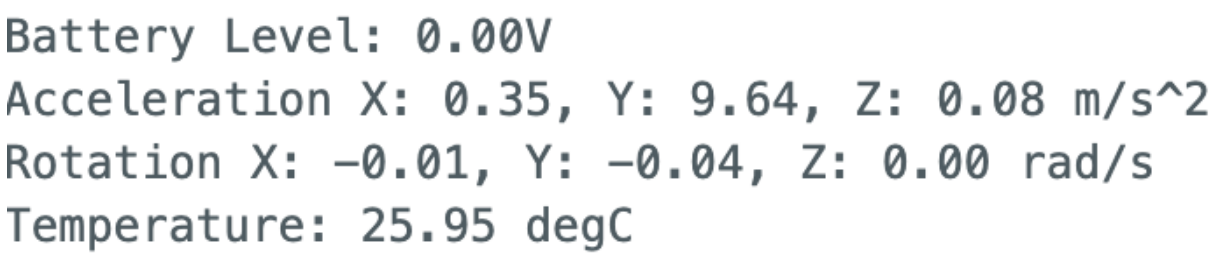

Figure 3: Voltage and IMU values output by ESP32-S3 (placed vertically)

2.2.2. Communication Speed Testing

Reliable and low-latency communication is critical for responsive robot control. We need to transfer 6 bytes data within 16 ms to fulfill real-time control in a battlebot environment.

- **Bluetooth** [2] showed high latency and inconsistent packet delivery, especially under load. It was quickly deemed unsuitable for real-time control.

- **WiFi with WebSocket** [3]

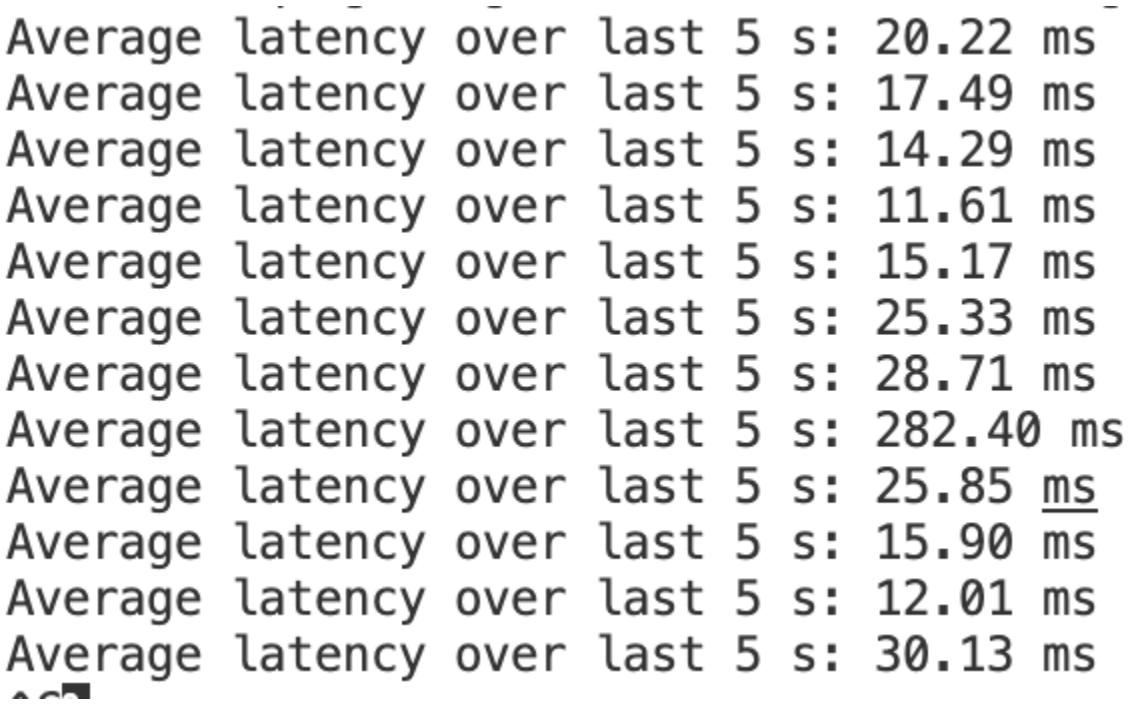

Figure 4: average latency measurement over 5s

Figure 4 shows the average latency of WiFi communication over 5s. We are sending 6 bytes data from esp32 to the computer via WiFi over 1 min and calculating the average

latency per 5 seconds. As Figure 4 shows, it is very unstable with WiFi connection. Most of the latencies are larger than 16 ms which is required by real-time control.

- **ESP-NOW** provided consistently low latency (~5–20ms) with minimal jitter and near-instant communication, even under repeated stress tests.

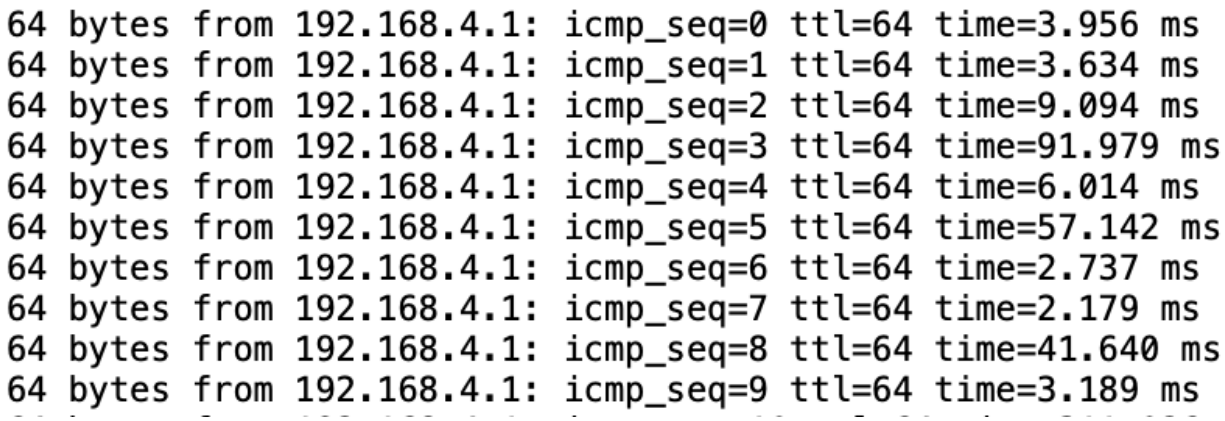

Figure 5: latency with ESP-NOW (derived by Jason)

In figure 5, we are getting 64-byte data from ESP32. The average latency is much lower than 16ms. Since we are sending 64 bytes which are larger than 6 bytes as needed, the cases that latency is larger than 16ms are acceptable.

Each protocol was tested by sending a timestamped command from the ESP32-S3 to the PC and logging the time difference upon reception. We found ESP-NOW to be both the fastest and most stable solution, making it the final choice for our autonomous system.

3 Conclusion

3.1 self-assessment

Throughout the project, I have contributed a fair portion of the workload, particularly in areas related to hardware design and communication protocols. My responsibilities included designing and testing the custom PCB, implementing and debugging sensor data collection (MPU6050 and voltage monitoring), and setting up both Bluetooth and WiFi communication systems between the robot and the PC control. I estimate that my individual workload accounts for approximately 30% of the overall project effort.
Currently, I am mostly on schedule. The remaining work involves PCB soldering and testing.

3.2 Plans for remaining work

For remaining work, Michael and I will finish soldering and test the PCB functionality.

3/31	Modify the PCB design by Friday
4/7	Get the PCB, solder and test, Finalize the PCB design
4/14	Solder the final version of PCB and test

Reference:

[1] “ESP32_Example” Accessed Apr 2, Available: https://courses.grainger.illinois.edu/ece445/wiki/#/esp32_example/index

[2] “Getting Started with ESP32 Bluetooth Low Energy (BLE) on Arduino IDE” Accessed Apr 2, Available: https://randomnerdtutorials.com/esp32-bluetooth-low-energy-ble-arduino-ide/

[3] “ESP32 WiFi Tutorial & Library Examples” Accessed Apr 2, Available: https://deepbluembedded.com/esp32-wifi-library-examples-tutorial-arduino/