MEAM 510: Design of Mechatronic Systems Fall 2024

Final Lab: Group 3

Name: Kashish Garg, Oluwagbemiga Oloyede, Dhyey Shah

1 Functionality:

For this project, there was a major focus towards control and hardware design. From experience, we have seen that decent autonomy hinges on tight control and calibrations. On this front, we made sure that the robot's rotations were discrete movements and that the linear velocity speed controller was properly tuned. This strategy seemed to work well for our missions. The robot could make n turns until its heading faced the waypoint and then drive until it was some distance from the goal. For hardware, we used a skid steer design with four motors to ensure high torque and speed. Each motor also held an encoder with 12 ticks per rotation, this would be something to improve upon in the future.

In addition to control, it was clear that the state estimation was subpar. The Vive would return positions with high noise, especially on the edges of the course. One method to mitigate this issue was to use a sliding window to ensure that a series of good measurements is what determines the origin of the robot; from there, all points to go to for the mission were waypoints defined relative to this origin.

On the Architecture side of things, there was a lot of thought done on designing the software for a proper flow of logic. We were able to easily adapt the code from lab 4 by using the motor control logic as the core idea of the code. Essentially, there is an enum holding the different "modes" the robot is in. These modes can only be changed from the web interface. The different modes are ["Manual", "Auto", "WallFollow"]. Once a robot enters one of those modes, we can calculate the linear and angular velocity with which to control the robot, given the Vive reading or the TOF sensor reading. Each mode can be preempted by another.

As far as what worked, we felt that our robot had a very robust manual controller that helped with all the relevant tasks during evaluation. Wall following was performed using two TOF sensors and took two attempts to perform a full circuit. This worked pretty well in our testing, and we were happy with this implementation. We think that the autonomy mode had some very solid ideas, and we were able to hit one of the three buttons, however there was some piece of logic causing the robot to get stuck between two angles and keep rotating clockwise and counter clockwise, rendering it immobile.

On the day of the graded evaluation, the I2C interface was giving us trouble with integration with our system. We ran out of time for implementing it for the graded evaluation, however we did get it working for competition day. In summary, everything we implemented worked pretty well on competition day, except for waypoint missions.

2 Mechanical Design

We designed our robot to be a skid steer robot, we have four motors (JGA25-371), and they were chosen due to their high performance from previous projects. They each have integrated quadrature encoders. All custom components were laser cut, with either acrylic glue or bolts to mate them. These motors were mounted on the baseplate shown in Section 6.7 as "Bulma Base". We included three TOF sensors to look directly in front of the robot and one on either side of it. This sensor array was mounted on the acrylic sheet shown in Section 6.7 as "Bulma TOF Mount".

For localization, we ensured the Vive board was mounted on the top to ensure we got clean readings. We figured making a design that is adaptable and allows for improvising was valuable, so we added mounting holes in remote areas where ever to ensure future proofing for any unforeseen issues.

One method that we tried to take was to reuse old omni-wheels within a skid-steer configuration. This worked for lab 4 where there was a manual operator, however for the final project, due to size of the wheel and its morphology, we ended up accruing more error than we had wished for, requiring us to change the wheels entirely.

One unaccounted component that we should've spent more time thinking about before laser cutting our plates was the Whisker sensor. On the morning of the graded evaluation, we ended up designing a flimsy fixture. This fixture worked, but it could've been designed with more intention and forethought.

3 Electrical Design

3.1 Overall Circuit Design

The electrical circuit design of our robot car was planned to ensure efficient power distribution and communication between components, while ensuring optimal functionality for all subsystems. The major components of our system are noted below:

- Central Microcontroller: One ESP32-S3 acted as the central and sole microcontroller, commanding the robot's functions.
- Motor Control: A single L298N motor driver powered all four JGA25-371 motors with built-in encoders. The OUT and EN pins for the motors on the right side and the left side were shared to minimize redundant use of controller pins.
- Wall-Following Circuit: Three Time-of-Flight (VL53L0X) sensors were connected to the ESP32-S3 via the same I2C line (pins 8 and 9). These sensors were powered by the 3.3V supply from the ESP32-S3.
- Localization Circuit: Two Vive circuits provided localization data, with their logic output relayed to the ESP32-S3.
- Top Hat Integration: The Top Hat module used another set of I2C pins to interface with the NeoPixel ring and whisker switch. An additional ESP32-S3 (master for Top Hat communication) handled Wi-Fi packet transmission from the slave circuit to the main ESP32-S3.
- Attacking Arm: A DSSERVO 20 kgcm servo motor controlled the attacking arm, designed as a Minecraft pickaxe. The servo was powered by the 5V output from the motor driver.
- Power Supply: The circuit was powered by a 2200 mAh Zeee 11.1V 50C LiPo battery. An external power bank provided additional current for the main ESP32-S3 to support the servo motor.

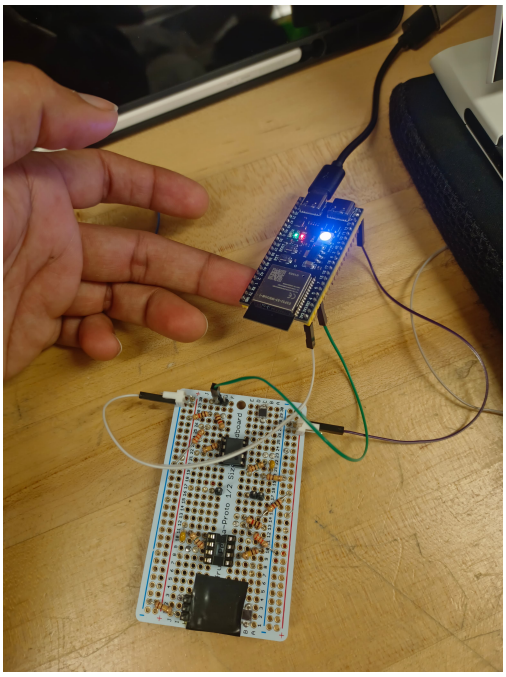
3.2 Motor Driver Circuit

The motor driver circuit featured a single L298N motor driver, powered by a 2200mAh 11.1V LiPo battery, to control the four JGA25-371 motors (rated at 463 RPM at 12V). The two motors on the right side of the chassis were connected to the same output and enable pins of the motor driver, sharing a common PWM signal for synchronized control. Similarly, the left-side motors were connected to the second set of output and enable pins. A SN74HC14N inverter was integrated to manage logic control for the motor direction pins. The direction, enable, and encoder pins were interfaced with the ESP32 S3, which was powered by the 5V output of the L298N (as depicted in circuit diagram 6.5.1).

3.3 Vive Detection Circuit

We implemented two identical Vive circuits soldered onto a single perfboard. Both circuits utilized a single TLV272 OPAMP IC along with the PD70-01 photodiode, following the design outlined

in the lecture slides. One circuit provided the robot's position coordinates (x,y), while the other determined its heading or yaw. This setup was essential for enabling autonomous operation, allowing the robot to accurately target objectives during evaluations and competitions. The circuit was powered by a 3.3V supply directly from the ESP32 S2, with each Vive circuit output being fed into the ESP32 S2 for further data processing.



3.4 Wall Following Circuit

The wall-following circuit incorporated three Time-of-Flight (VL53L0X) sensors connected to the ESP32 S3 via a shared I2C line (pins 8 and 9). The XSHUT pins were connected to GPIOs to enable simultaneous operation of the sensors. The sensors were powered by the 3.3V supply from the main ESP32 S3.

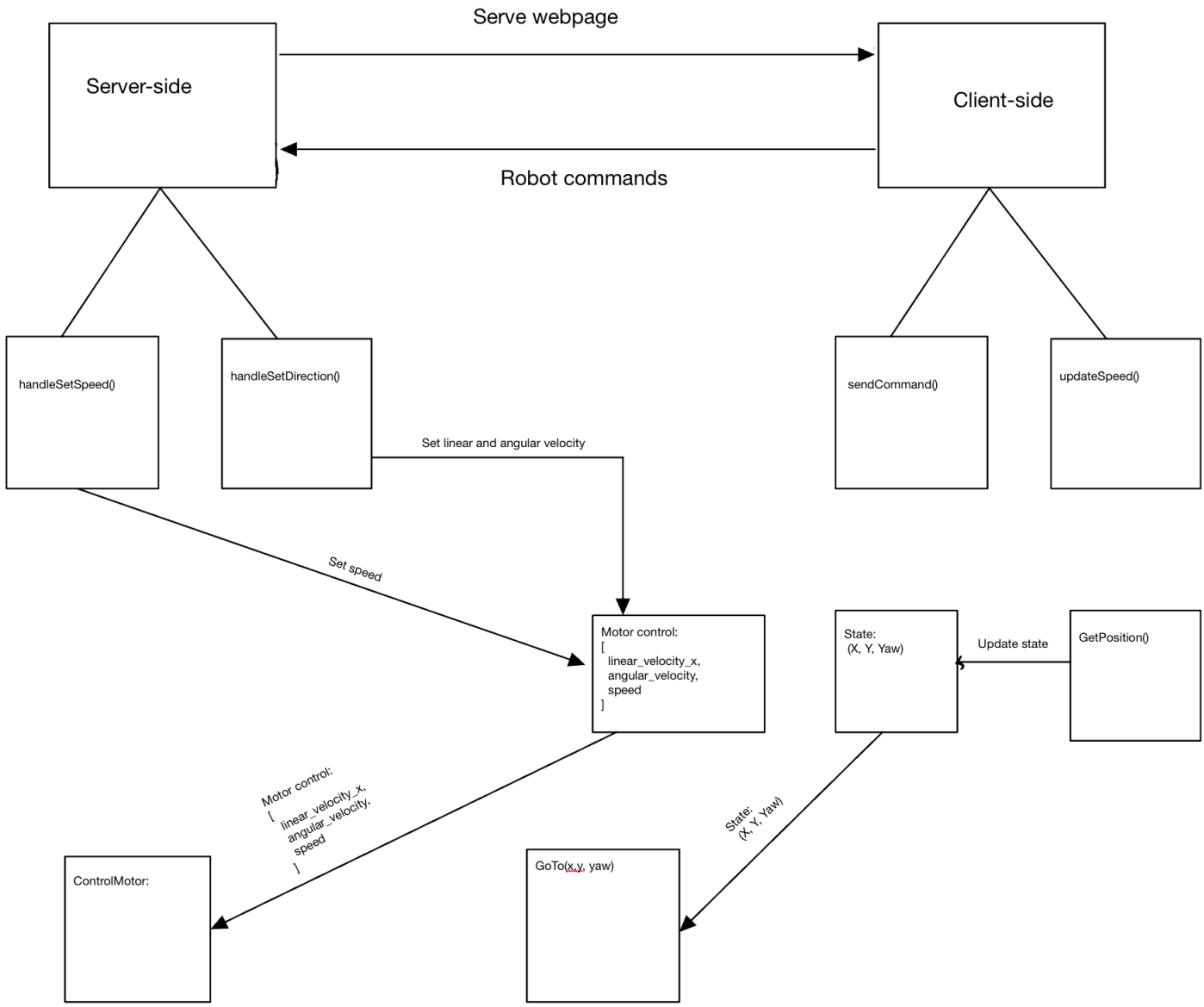
3.5 Attacking Arm and Top Hat Circuit

The attacking arm utilized a DSSERVO 20 kgcm servo motor, powered by the 5V output from the L298N motor driver. The servo's signal pin was connected to the main ESP32 S3, as illustrated in circuit diagram 6.5.4.

For Top Hat communication, an additional ESP32 S3 was employed as a master to interface with the provided slave circuit. The slave circuit included the Neopixel ring and the Whisker Switch. Communication between the master ESP32 and the slave ESP-Wroom was facilitated through I2C lines, while a dedicated GPIO flag allowed the master ESP32 to communicate with the main ESP32 S3, as depicted in circuit diagram 6.5.5.

4 Processor Architecture and Code Architecture

The figure below shows the high-level architecture of our software.



The code architecture is structured into two main parts: the client-side and the server-side. The loop is the central part of the application, where all the necessary functions for proper functionality of the robot are called. Below is a breakdown of each part of the system.

4.1 Client-side:

The client-side is a simple web application built using HTML, CSS, and JavaScript. It communicates with the server-side and provides user input through two primary functions: `sendCommand()` and `updateSpeed()`.

- `sendCommand()`: Sends user commands from the client-side to the server-side for processing.
- `updateSpeed()`: Sends the requested speed from the user to the server-side.

The commands sent from the client are encoded as integers, each corresponding to a specific robot action. These commands range from 0 to 6, as summarized in the table below:

Command Code	**Meaning**
0	Stop
1	Forward
2	Left
3	Right
4	Attack
5	Follow Wall
6	Autonomous Location Targeting

4.2 Server-side:

The server-side handles incoming requests from the client and contains three main request handlers:

- `handleRoot()`: Handles requests for the index page.
- `handleSetSpeed()`: Processes requests to update the robot's speed.
- `handleSetDirection()`: Handles requests to control the robot's movement and switch between modes.

Both `handleSetSpeed()` and `handleSetDirection()` modify a global array, `MotorControl`, which represents the robot's current velocity and speed. The array is structured as follows:

$$\texttt{MotorControl} = [\text{linear velocity}, \text{angular velocity}, \text{speed}]$$

4.3 Control Loop

The core of the robot's control system is the `loop()` function, which continuously calls other necessary functions to ensure the robot moves as per user input and completes the desired task. Below is a breakdown of the primary functions used by `loop()` to control the robot:

1. `ControlMotor()`: This function reads the `MotorControl` array to determine the robot's movement direction. It processes this data and sends appropriate commands to the motors to ensure the robot moves at the desired velocity and speed.

2. `GetPosition()`: Updates a state structure that captures the robot's current location and yaw (orientation) in the Vive coordinate system.

3. `GoTo(x, y, yaw)`: Takes in the target coordinates (x, y) and orientation (yaw) and computes the velocity needed for the robot to reach the target location. It then updates the `MotorControl` array with the necessary movement commands.

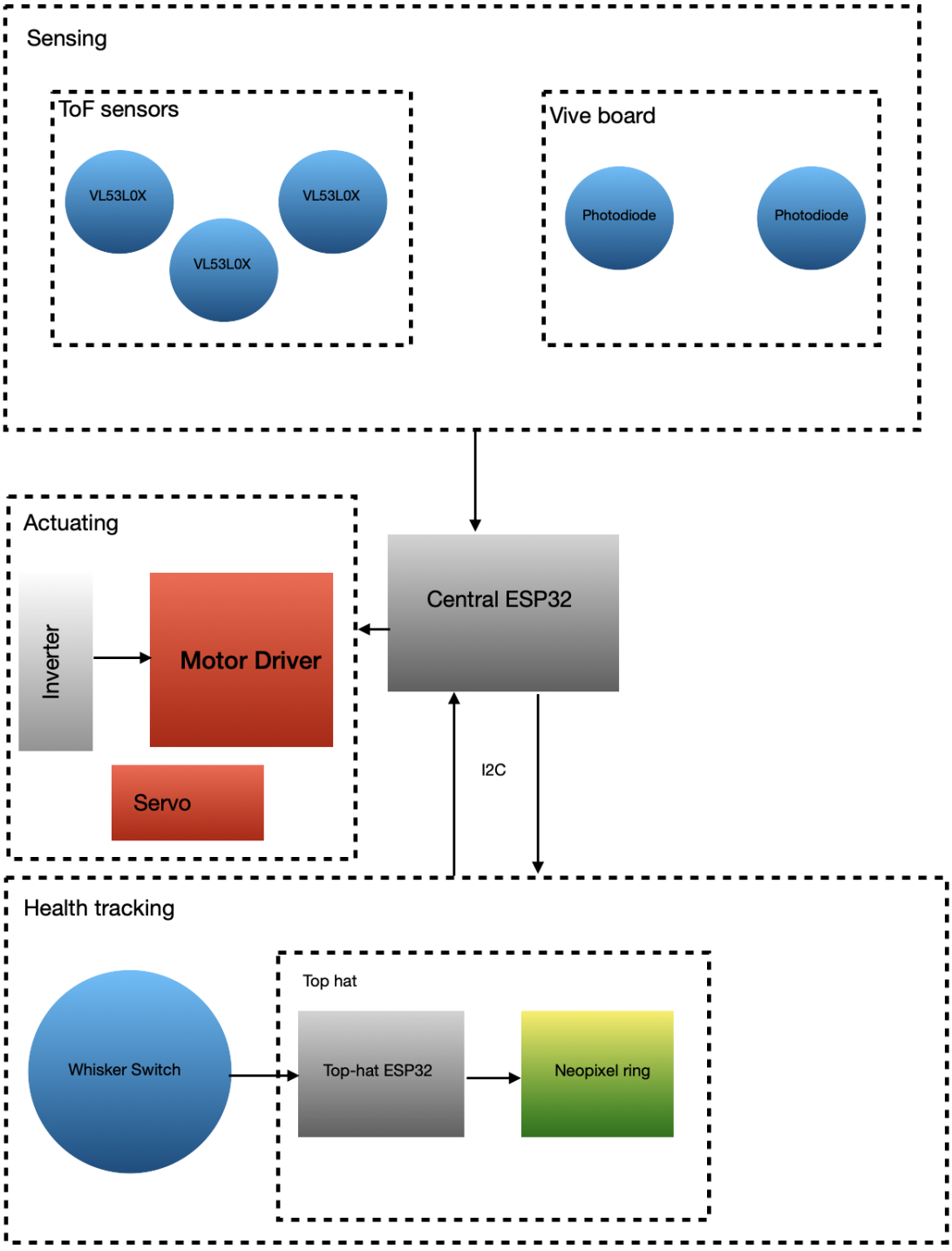
4. `wallfollow()`: This function is activated when the robot enters wall-following mode. The basic algorithm works as follows:

   - If the front `ToF` (Time of Flight) sensor detects the robot is too far from the wall (i.e., it exceeds the threshold), the robot turns left to correct its trajectory.
   - It checks the right `ToF` sensor, and if its value exceeds its threshold, the robot adjusts its path by turning right.

   During wall-following, the `MotorControl` array is updated continuously to guide the robot along the wall.

4.4 MCU Connection Logic

Our project consists of four primary components, with the central unit being the main ESP32. The remaining three components—the sensing unit, actuating unit, and health tracking unit—communicate with the central unit to perform their respective functions. The diagram below illustrates the overall architecture and the connections between these units.



4.5 Challenges

During the design of our code, we initially devised a relatively complex plan to create an occupancy grid that would represent the battle arena for path planning. However, right from the start, we encountered significant difficulties in implementing this grid. From this experience, we learned an

important lesson: **start simple**.

One of the major challenges we faced was **code integration**. While our approach of separating different components and circuits for individual testing seemed promising, it became clear that integrating them later in the process was problematic. Testing the individual parts in isolation made it easier to identify faults, but as the deadline approached, it was critical to integrate and test these components much earlier to ensure they worked together seamlessly.

One specific area where integration caused issues was the **tophat-whisker interface**, which proved to be particularly troublesome towards the end of the project.

4.6 Summary

The control system is centered around the `loop()` function, which integrates various subsystems like motor control, positioning, and sensor-based behaviors (e.g., wall-following). It interacts with both the client-side (via commands) and server-side (via request handlers) to manage robot actions and achieve the desired operation modes.

5 Retrospective

5.1 Kashish Garg:

What you feel was most important that you learned.
I think the importance placed on some major components of embedded systems programming is what I took away from the class. The importance of timers, ADCs specifically. I think also a level of comfort-ability with electronics was valuable to me as well.

What was the best parts of the class.
I think the comradery with my classmates going through frustrations and anger within the final project really brought a lot of joy to me. I remember hugging a classmate when we decided to finish after 34 hours of continous work.

What you had the most trouble with.
I think hardware classes are going to be inherently tough and time-consuming. I think having to work a 9-5 while taking this class proved to be one of the most challenging things I've had to do.

What you wish was improved.
The top hat component of the final felt very rushed. I also think the waldo lab should be stripped, giving an extra week to the final.

Your thoughts on gameplay.
I think the game was alright, but my focus was on the graded work. I'm happy that it translated well between the two of them.

5.2 Oluwagbemiga Oloyede:

What you feel was most important that you learned.
I think the most important thing I learned in the class were important aspects of embedded programming such as timers and how they can be powerful tools in a lot of different applications. I also found learning how to design circuits for different use cases very useful.

What was the best parts of the class.
I think the best parts of this class was the joy of seeing things you designed and spent hours working on come to life. I also enjoyed how everyone supported each other in tacking hard problems.

What you had the most trouble with.
I had the most trouble with the mechanical design aspect of this class. As I have no prior experience in mechanical design. Designing the circuit and writing the code for lab 3 went fairly well but the mechanical design proved to be quite a task for me.

What you wish was improved.
I think figuring out the mechanical design was a nice challenge, however I felt more support and resources could have been provided much earlier in the semester on how to go about mechanical design and using CAD software.

Your thoughts on gameplay.
The gameplay was good, however I felt a lot of the requirements were vague and often needed a lot of clarifying.

5.3 Dhyey Shah:

What you feel was most important that you learned. The most important thing I learnt was a methodical integration of electronic systems to function smoothly for multiple tasks while handling and debugging for signal noise. As an electrical engineer undergrad I gained a lot of insights into mechanical design as a new skill as well.
What was the best parts of the class. For me the best part of the class MEAM510 was probably working on Lab3-Waldo where I let my creative side take over. For the final project, it was fun to see implementing localization and see our robot move autonomously and test it on the course along with other teams late in the night.
What you had the most trouble with. I thinking I had some initial trouble understanding register level programming, Wifi communication and then we faced some issues tuning our PD controller gains for our robot for smooth operation. Apart from that the class required us to dedicate a good amount of time which required a lot of managing but was fun.
What you wish was improved. On a personal level I think a few topics could have used a few more classes as that would have given us some time to explore more on those topics, like communication protocols, logic levels, encoders. For the final project I think, the Top Hats availability was an issue as there was not enough time for us to test them till the end.
Your thoughts on gameplay. The gameplay was well thought, it did fee a little too convoluted in terms of the requirements initially but as we went along it all made sense and was fun to watch and play and we ended up in the winning alliance (1) which made it even sweeter for us.

6 Appendix:

6.1 Code

6.2 main.o

```
#include <WiFi.h>
#include <WiFiClient.h>
#include <WebServer.h>
#include "index.h"
#include "html510.h"
#include "Adafruit_VL53L0X.h"
#include "vive510.h"
#include <math.h>

enum RobotMode {
  MANUAL,
  WALL_FOLLOW,
  TARGET
};

enum RobotMode robot_mode_ = MANUAL;
// ############ VIVE ############ //
#define VIVE_PIN1 37 // pin receiving signal from Vive circuit 1
#define VIVE_PIN2 38 // pin receiving signal from Vive circuit 2

Vive510 vive1(VIVE_PIN1);
Vive510 vive2(VIVE_PIN2);

uint32_t med3filt(uint32_t a, uint32_t b, uint32_t c) {
  uint32_t middle;
  if ((a <= b) && (a <= c))
    middle = (b <= c) ? b : c;
  else if ((b <= a) && (b <= c))
    middle = (a <= c) ? a : c;
  else     middle = (a <= b) ? a : b;
  return middle;
}
struct PositionData {
  int16_t x=0;
  int16_t y=0;
  float yaw=0;
};

struct PositionData state_;
struct PositionData origin_;
bool origin_initialized_ = 0;


void GetPosition() {
  static uint16_t x1, y1, x2, y2;
  static float yaw;

  if (vive1.status() == VIVE_RECEIVING) {
    static uint16_t x0_1, y0_1, oldx1_1, oldx2_1, oldy1_1, oldy2_1;
    oldx2_1 = oldx1_1; oldy2_1 = oldy1_1;
    oldx1_1 = x0_1;      oldy1_1 = y0_1;

    x0_1 = vive1.xCoord();
    y0_1 = vive1.yCoord();
    x1 = med3filt(x0_1, oldx1_1, oldx2_1);
    y1 = med3filt(y0_1, oldy1_1, oldy2_1);

    if (x1 > 8000 || y1 > 8000 || x1 < 1000 || y1 < 1000) {
      x1 = 0; y1 = 0;
    }
  } else {
    x1 = 0;
    y1 = 0;
    vive1.sync(5);
  }
    if (vive2.status() == VIVE_RECEIVING) {
    static uint16_t x0_2, y0_2, oldx1_2, oldx2_2, oldy1_2, oldy2_2;
    oldx2_2 = oldx1_2; oldy2_2 = oldy1_2;
    oldx1_2 = x0_2;      oldy1_2 = y0_2;

    x0_2 = vive2.xCoord();
    y0_2 = vive2.yCoord();
    x2 = med3filt(x0_2, oldx1_2, oldx2_2);
    y2 = med3filt(y0_2, oldy1_2, oldy2_2);

    if (x2 > 8000 || y2 > 8000 || x2 < 1000 || y2 < 1000) {
      x2 = 0; y2 = 0;
    }
  } else {
    x2 = 0;
```

```
    y2 = 0;
    vive2.sync(5);
  }
  if (x1 != 0 && y1 != 0 && x2 != 0 && y2 != 0) { // Ensure both trackers have valid data
    float dx = x2 - x1;
    float dy = y2 - y1;
    yaw = atan2(dy, dx) * 180 / PI; // Yaw in degrees

    // Normalize the angle to 0-360 degrees
    if (yaw < 0) {
      yaw += 360;
    }

    state_.x = (uint16_t)(x1 + x2)/2;
    state_.y = (uint16_t)(y1 + y2)/2;
    if (yaw > 90) {
      yaw = yaw - 90;
    } else {
      yaw = 360 - (90-yaw);
    }
    state_.yaw = yaw;

    Serial.println("POSITION");
    Serial.print("X: ");
    Serial.println(state_.x);
    Serial.print("Y: ");
    Serial.println(state_.y);
    Serial.print("YAW: ");
    Serial.println(state_.yaw);

  } else {
    Serial.println("Invalid data for yaw calculation.");
  }

}
// ############ VIVE ############ //

// ############ TOF ############ //
// address we will assign if dual sensor is present
#define LOX1_ADDRESS 0x30
#define LOX2_ADDRESS 0x31
#define LOX3_ADDRESS 0x32

// set the pins to shutdown
#define SHT_LOX1 41
#define SHT_LOX2 40
#define SHT_LOX3 39

// objects for the vl53l0x
Adafruit_VL53L0X lox1 = Adafruit_VL53L0X();
Adafruit_VL53L0X lox2 = Adafruit_VL53L0X();
Adafruit_VL53L0X lox3 = Adafruit_VL53L0X();

// this holds the measurement
VL53L0X_RangingMeasurementData_t measure1;
VL53L0X_RangingMeasurementData_t measure2;
VL53L0X_RangingMeasurementData_t measure3;
// ############ TOF ############ //

// ##### WALL FOLLOWING ###### //
// TODO: Figure out the thresholds
int front_tof_threshold[4] = {600, 500, 500, 500};

// ##### WALL FOLLOWING ###### //


#define LEDC_RESOLUTION_BITS 14
#define LEDC_RESOLUTION ((1 << LEDC_RESOLUTION_BITS) - 1)
#define LEDC_FREQ_HZ 10

// Meters
#define HALF_WHEEL_BASE_B 0.13
#define WHEEL_RADIUS .035
#define PI 3.14159265358979323846

#define KD -.01
//*******WIFI***********//
// WiFi Credentials
const char *ssid = "Bulma";
const char *password = "12345678";
// Web server on port 80
WebServer server(80);

// Motor control values array
float motorControl[3] = {0.0, 0.0, 0.0};
PositionData waypoints[] = {{-1000, 0, 0}, {-1000, -900, 0},  {-3400, -900, 0}, {-3400, -400, 0}, {-2350, -400, 0}, {-4400,

//*******WIFI***********//

hw_timer_t * timer = NULL;
bool ret = false;
```

```
bool ledsig = false;

const int encoderPIN1[4] = {4, 5, 6, 7}; // channel 1 for 4 encoders
const int motorPINEN[4] = {15, 16, 17, 18}; // motor EN PWM pins
const int motorPINDIR[4] = {20, 3, 46, 21}; // motor DIR pins

volatile long ticksPerInterval[4] = {0, 0, 0, 0};
volatile long motorDir[4] = {1, 1, 1, 1};

volatile float motorVelSetpoint[4] = {0.0, 0.0, 0.0, 0.0};
volatile float kpMotor[4] = {50, 50, 50, 50};
volatile float kpMotorAng[4] = {2, 2, 2, 2};

volatile float currentWheelAngVel[4] = {0.0, 0.0, 0.0, 0.0};
volatile unsigned long currentWheelPwm[4] = {0, 0, 0, 0};

int CPR = 13.75;
float rotPerTick = 360.0 / CPR;
const int timer_interval = 100;

int initializeControl = 0;

void IRAM_ATTR encoderISR0() { ticksPerInterval[0] += motorDir[0];}
void IRAM_ATTR encoderISR1() { ticksPerInterval[1] += motorDir[1];}
void IRAM_ATTR encoderISR2() { ticksPerInterval[2] += motorDir[2];}
void IRAM_ATTR encoderISR3() { ticksPerInterval[3] += motorDir[3];}

// ############ TOF ############ //
void setID() {
  // all reset
  digitalWrite(SHT_LOX1, LOW);
  digitalWrite(SHT_LOX2, LOW);
  digitalWrite(SHT_LOX3, LOW);
  delay(10);
  // all unreset
  digitalWrite(SHT_LOX1, HIGH);
  digitalWrite(SHT_LOX2, HIGH);
  digitalWrite(SHT_LOX3, HIGH);
  delay(10);

  // activating LOX1 and resetting LOX2
  digitalWrite(SHT_LOX1, HIGH);
  digitalWrite(SHT_LOX2, LOW);
  digitalWrite(SHT_LOX3, LOW);

  // initing LOX1
  if(!lox1.begin(LOX1_ADDRESS)) {
    Serial.println(F("Failed to boot first VL53L0X"));

    while(1);
  }
  delay(10);

  // activating LOX2
  digitalWrite(SHT_LOX2, HIGH);
  delay(10);

  //initing LOX2
  if(!lox2.begin(LOX2_ADDRESS)) {
    Serial.println(F("Failed to boot second VL53L0X"));
    while(1);
  }

  // activating LOX3
  digitalWrite(SHT_LOX3, HIGH);
  delay(10);

  if(!lox3.begin(LOX3_ADDRESS)) {
    Serial.println(F("Failed to boot Third VL53L0X"));
    while(1);
  }
}

void read_tof() {

  lox1.rangingTest(&measure1, false); // pass in 'true' to get debug data printout!
  lox2.rangingTest(&measure2, false); // pass in 'true' to get debug data printout!
  lox3.rangingTest(&measure3, false); // pass in 'true' to get debug data printout!

  Serial.print(F("1: "));
  if(measure1.RangeStatus != 4) {     // if not out of range
    Serial.println(measure1.RangeMilliMeter);
  } else {
    Serial.println(F("Out of range"));
  }

  Serial.print(F(" "));

  Serial.print(F("2: "));
  if(measure2.RangeStatus != 4) {
    Serial.println(measure2.RangeMilliMeter);
```

```
  } else {
    Serial.println(F("Out of range"));
  }

  Serial.print(F(" "));

  Serial.print(F("3: "));
  if(measure3.RangeStatus != 4) {
    Serial.println(measure3.RangeMilliMeter);
  } else {
    Serial.println(F("Out of range"));
  }


  Serial.println();
}
// ############ TOF ############# //

void calcMotorVelSetpoint(float lin_vel, float ang_vel) {
  float ang_v_left = (lin_vel - HALF_WHEEL_BASE_B * ang_vel) / WHEEL_RADIUS;
  float ang_v_right = (lin_vel + HALF_WHEEL_BASE_B * ang_vel) / WHEEL_RADIUS;

  motorVelSetpoint[0] = ang_v_right;
  motorVelSetpoint[1] = ang_v_right;
  motorVelSetpoint[2] = ang_v_left;
  motorVelSetpoint[3] = ang_v_left;
}

void controlMotor(float lin_vel, float ang_vel) {

  static unsigned long previous_timestamp;
  unsigned long current_timestamp = millis();
  static int prev_tick_dot[4];

  if (initializeControl == 0) {
    previous_timestamp = current_timestamp;
    initializeControl = 1;
    return;
  }

  double time_diff = current_timestamp - previous_timestamp;

  int u[4];
  float current_wheel_vel[4];
  static int prev_tick[4];
  static int ticks_between_control[4];
  Serial.println("Start");
  for (int i = 0; i < 4; i++) {
    float tick = ticksPerInterval[i] - prev_tick[i];
    tick = tick / (34.0);
    float rad = tick * 30 * PI / 180;
    float ang_vel = rad / (time_diff/1000.0);
    prev_tick[i] = ticksPerInterval[i];
    currentWheelAngVel[i] = ang_vel;
  }

  float error[4];
  float acceleration[4];


  for (int i = 0; i < 4; i++) {
    error[i] =  abs(motorVelSetpoint[i]) - abs(currentWheelAngVel[i]);
    acceleration[i] = 1000 * (abs(currentWheelAngVel[i]) - abs(prev_tick_dot[i] / time_diff));

    float u = kpMotor[i] * error[i] + KD* acceleration[i];
    currentWheelPwm[i] = currentWheelPwm[i] + u;
    // Clamping
    if (currentWheelPwm[i] > 16383){
      currentWheelPwm[i] = 16383;
    } else if(currentWheelPwm[i] < 0) {
      currentWheelPwm[i] = 0;
    }
    Serial.print("U Control Signal: ");
    Serial.println(currentWheelPwm[i]);

    if (motorVelSetpoint[i] > 0) {
      digitalWrite(motorPINDIR[i], HIGH);
      motorDir[i] = 1;
    } else if (motorVelSetpoint[i] < 0) {
      digitalWrite(motorPINDIR[i], LOW);
      motorDir[i] = -1;
    }
    if (abs(lin_vel) < .01) {
      ledcWrite(motorPINEN[i], 4000);

    } else {
      if (abs(ang_vel) < .01) {
        ledcWrite(motorPINEN[i], currentWheelPwm[1]);
      } else {
        if ( i == 0 || i == 1) {
          int val;
```

```
          if (ang_vel > 0) {
            val = currentWheelPwm[1] + 2500;
          } else {
            val = currentWheelPwm[1] - 2500;
          }
          if (val > 16383) {
            val =  16383;
          } else if (val < 0) {
            val = 0;
          }
          ledcWrite(motorPINEN[i], val);
        } else if (i == 2 || i == 3) {
          int val;
          if (ang_vel > 0) {
            val = currentWheelPwm[1] - 2500;
          } else {
            val = currentWheelPwm[1] + 2500;
          }
          if (val > 16383) {
            val =  16383;
          } else if (val < 0) {
            val = 0;
          }
          ledcWrite(motorPINEN[i], val);
        }
      }
    }
  }
  for (int i=0; i < 4; i++) {
    prev_tick_dot[i] = currentWheelAngVel[i];
  }
  previous_timestamp = current_timestamp;

}

void turn_left() {
  static int iter = 0;
  // int steps_for_turn = 10;
  while (measure2.RangeMilliMeter < 600) {
    motorControl[0] = 0;
    motorControl[1] = .1;
    motorControl[2] = 0.0;
    float lin_vel_ = motorControl[0] * motorControl[2]/100.0;
    float ang_vel_ = motorControl[1];
    calcMotorVelSetpoint(lin_vel_, ang_vel_);
    controlMotor(lin_vel_, ang_vel_);
    read_tof();
    delay(50);
    iter++;
  }
  if (measure2.RangeMilliMeter < 700 &&  measure1.RangeMilliMeter > 275) {
    motorControl[0] = 0;
    motorControl[1] = -.1;
    motorControl[2] = 0.0;

  }
  iter = 0;
}

void wall_follow() {
  Serial.println("IM FOLLOWING THE WALL!");
  motorControl[0] = 0;
  motorControl[1] = 0;
  motorControl[2] = 0.0;
  // We are assuming initial position is constant
  static const int left_margin = 250;
  static const int right_margin =200;
  Serial.print("Front Sensor: ");
  Serial.println(measure2.RangeMilliMeter);
  Serial.print("Right Sensor: ");
  Serial.println(measure1.RangeMilliMeter);

  // TODO: Set i based off of iteration;
  int i = 0;
  if (measure2.RangeMilliMeter > front_tof_threshold[i]) {
    motorControl[0] = 1;
    motorControl[1] = 0;
    motorControl[2] = 50.0;
  } else if (measure2.RangeMilliMeter < front_tof_threshold[i]){
    motorControl[0] = 0;
    motorControl[1] = 0;
    motorControl[2] = 0;
    turn_left();

  } else if (measure2.RangeMilliMeter > 600) {
    motorControl[0] = 0;
    motorControl[1] = -.1;
    motorControl[2] = 0;

  }
  delay(50);
```

```
}

bool goTo(int16_t x_goal, int16_t y_goal) {
  static bool turning_mode = 0;
  static int16_t x_temp = 0;
  static int16_t y_temp = 0;
  Serial.print("X goal offset: ");
  Serial.println(x_goal);
  Serial.print("Y goal offset: ");
  Serial.println(y_goal);
  Serial.print("X Origin ");
  Serial.println(origin_.x);
  Serial.print("Y Origin");
  Serial.println(origin_.y);
  x_goal = origin_.x + x_goal;
  y_goal = origin_.y + y_goal;

  Serial.print("Waypoint x: ");
  Serial.println(x_goal);
  Serial.print("Waypoint y: ");
  Serial.println(y_goal);
  // get current position (state_)
  // calculate angle difference
  float dx = x_goal - state_.x;
  float dy = y_goal - state_.y;
  float ang_tan2 = (atan2(dy, dx) * 180 / PI) ;
  if (ang_tan2 < 0) {
    ang_tan2 = 360 + ang_tan2;
  }
  float ang_diff = ang_tan2 - state_.yaw;
  // ang_diff = atan2(sin(ang_diff), cos(ang_diff));
  if (ang_diff > 180.0) {
    ang_diff -= 360.0;  // Turn the angle to the shortest direction
  } else if (ang_diff < -180.0) {
    ang_diff += 360.0;  // Adjust for negative angles beyond -180
  }
  Serial.print("ang_tan2: ");
  Serial.println(ang_tan2);

  Serial.print("ang_diff: ");
  Serial.println(ang_diff);

  float ang_threshold = 12.0;
  float distance_threshold = 150;
  float distance_ = sqrt((dx*dx)+ (dy*dy));
  Serial.print("Distance_: ");
  Serial.println(distance_);
  if (abs(distance_) < distance_threshold) {
    return true;
  }

  if (abs(ang_diff) > ang_threshold) {
    if (turning_mode == 0) {
      x_temp = state_.x;
      y_temp = state_.y;

      turning_mode = 1;
    }
    GetPosition();
    dx = x_goal - x_temp;
    dy = y_goal - y_temp;
    ang_tan2 = (atan2(dy, dx) * 180 / PI) ;
    if (ang_tan2 < 0) {
      ang_tan2 = 360 + ang_tan2;
    }
    ang_diff = ang_tan2 - state_.yaw;


    if (state_.yaw < ang_tan2) {
      motorControl[0] = 1;
      motorControl[1] = .1;
      motorControl[2] = 0;

    } else {
      motorControl[0] = 1;
      motorControl[1] = -.1;
      motorControl[2] = 0;
    }
    return false;
  } else  {
    turning_mode = 0;
    // if (turning_mode == )
    float distance = sqrt((dx*dx)+ (dy*dy));
    Serial.print("Distance: ");
    Serial.println(distance);
    if (abs(distance) > distance_threshold) {
      motorControl[0] = 1;
      motorControl[1] = 0;
      motorControl[2] = 60.0;
      return false;
    } else {
```

```
      return true;
    }
  }
}
int waypoint_iterator = 0;
void target_auto() {
  GetPosition();

  if (!origin_initialized_) {
    int init_count = 0;
    struct PositionData prev_state;
    while (init_count < 5){

      GetPosition();
      if (init_count > 0) {
        struct PositionData current_state;
        current_state = state_;
        // check for the dist between current and prev state
        float dx_origin = current_state.x - prev_state.x;
        float dy_origin = current_state.y - prev_state.y;

        float dist = sqrt((dx_origin*dx_origin) + (dy_origin*dy_origin));
        Serial.print("origin init dist: ");
        Serial.println(dist);
        Serial.print("dx_origin: ");
        Serial.println(dx_origin);
        Serial.print("dy_origin: ");
        Serial.println(dy_origin);
        Serial.print("current x: ");
        Serial.println(current_state.x);
        Serial.print("current y: ");
        Serial.println(current_state.y);
        Serial.print("prev] x: ");
        Serial.println(prev_state.x);
        Serial.print("prev y: ");
        Serial.println(prev_state.y);
        if (dist < 20) {
          init_count ++;
        } else {
          init_count = 0;
        }

      } else {
        init_count ++;
      }

      prev_state = state_;
    }

    origin_ = state_;
    origin_initialized_ = 1;
    Serial.println("Setting origin to: ");
    Serial.println( origin_.x);
    Serial.println( origin_.y);

  }
  Serial.println("IM FOLLOWING MY TARGETS");
  motorControl[0] = 0;
  motorControl[1] = 0;
  motorControl[2] = 0.0;
  Serial.print("Waypoint iterator: ");
  Serial.println(waypoint_iterator);
  bool success = goTo(waypoints[waypoint_iterator].x, waypoints[waypoint_iterator].y);
  if (success) {
    Serial.println("SUCCESS!!!!!");
    Serial.print(" number of waypoints: ");
    Serial.println(sizeof(waypoints)/sizeof(waypoints[0]));
    if (waypoint_iterator + 1 < sizeof(waypoints)/sizeof(waypoints[0])) {
      Serial.println("HERE!!!!!");

      waypoint_iterator++;
      // delay(10000);
    } else {
      Serial.println("Successful Mission!");
      robot_mode_ = MANUAL;
    }
  }


  delay(50);
}

void setup() {
  Serial.begin(115200);
  while (! Serial) { delay(1); }

  pinMode(SHT_LOX1, OUTPUT);
  pinMode(SHT_LOX2, OUTPUT);
  pinMode(SHT_LOX3, OUTPUT);

  Serial.println(F("Shutdown pins inited..."));
```

```
  Serial.println(F("Both in reset mode...(pins are low)"));
  Serial.println(F("Starting..."));
  setID();

  for (int i = 0; i < 4; i++){
    pinMode(encoderPIN1[i], INPUT);
    attachInterrupt(digitalPinToInterrupt(encoderPIN1[i]), i == 0 ? encoderISR0
    :(i == 1 ? encoderISR1 : (i == 2 ? encoderISR2 : encoderISR3)), CHANGE);
  }

 // PWM EN Pins
  ledcAttach(motorPINEN[0], LEDC_FREQ_HZ, LEDC_RESOLUTION_BITS);
  ledcAttach(motorPINEN[1], LEDC_FREQ_HZ, LEDC_RESOLUTION_BITS);
  ledcAttach(motorPINEN[2], LEDC_FREQ_HZ, LEDC_RESOLUTION_BITS);
  ledcAttach(motorPINEN[3], LEDC_FREQ_HZ, LEDC_RESOLUTION_BITS);

  // DIR Pins
  pinMode(motorPINDIR[0], OUTPUT);
  pinMode(motorPINDIR[1], OUTPUT);
  pinMode(motorPINDIR[2], OUTPUT);
  pinMode(motorPINDIR[3], OUTPUT);
  //****************PIN Setup*************//

  //****************WIFI*************//
  // Connectting to WiFi in AP mode
  WiFi.softAP(ssid, password);
  Serial.print("Access Point IP: ");
  Serial.println(WiFi.softAPIP());

  // Defining routes
  server.on("/", HTTP_GET, handleRoot);
  server.on("/setSpeed", HTTP_GET, handleSetSpeed);
  server.on("/setDirection", HTTP_GET, handleSetDirection);
  // server.on("/setMotorState", HTTP_GET, handleSetMotorState);
  server.begin();
  //****************WIFI*************//
  //****************VIVE*************//
  vive1.begin();
  vive2.begin();
  delay(2000);
  Serial.println("Vive trackers started");
  //****************VIVE*************//

}

//****************WIFI*************//
// Function to handle root
void handleRoot() {
  server.send(200, "text/html", htmlPage);
}

// Setting motor speed
void handleSetSpeed() {
  // if (robot_mode_ == MANUAL) {
    if (server.hasArg("value")) {
      motorControl[2] = server.arg("value").toInt();
      Serial.print("Speed set to: ");
      Serial.println(motorControl[2]);
      // delay(1000);
    }
    server.send(204);
   }

}

// Setting motor direction
void handleSetDirection() {
  if (server.hasArg("dir")) {
    int dir = server.arg("dir").toInt();
    if (dir == 2) {
      robot_mode_ = MANUAL;
      motorControl[1] += 0.1;  // Incrementing by 0.25 - Left

      // motorControl[0] = 0;
      // motorControl[2] = 0;

    } else if (dir == 3) {
      robot_mode_ = MANUAL;
      motorControl[1] -= 0.1;  // Decrement by 0.25 - Right
      // motorControl[0] = 0;
      // motorControl[2] = 0;
    }
    else if (dir == 1){
      robot_mode_ = MANUAL;
      motorControl[0] = 1; // Up
    }
    else if (dir == -1){
      robot_mode_ = MANUAL;
      motorControl[0] = -1; // Down
    }
```

```
    else if (dir == 0){
      robot_mode_ = MANUAL;
      motorControl[0] = 0; // Stop
      motorControl[1] = 0;
      // motorControl[2] = 0;
    } else if (dir == 4) {
      robot_mode_ = MANUAL;
    } else if (dir == 5) {
      robot_mode_ = WALL_FOLLOW;
      motorControl[0] = 0;
      motorControl[1] = 0;
      motorControl[2] = 0;
      // delay(50);
    } else if (dir == 6) {
      robot_mode_ = TARGET;
      motorControl[0] = 0;
      motorControl[1] = 0;
      motorControl[2] = 0;
      // delay(50);
    }

  }
  server.send(204);
}
//***************WIFI*************//


void loop() {
  read_tof();
  GetPosition();


  Serial.println(robot_mode_);
  if (robot_mode_ == WALL_FOLLOW) {
    wall_follow();

  } else if (robot_mode_ == TARGET) {
    target_auto();
  }
  server.handleClient();
  float lin_vel = 0.0;
  float ang_vel = 0.0;
  // put your main code here, to run repeatedly:
  // Wifi Servr code to get Velocity command:
  // moveMotor(1, 1);

  //***************WIFI*************//

  Serial.println("################");
  Serial.println("User Input: ");
  for (int i = 0; i < 3; i++) {
    Serial.println(motorControl[i]);
  }
  Serial.println("\n");

  // Set the direction of each motor from the input

  lin_vel = motorControl[0] * motorControl[2]/100.0;
  ang_vel = motorControl[1];
  if (abs(motorControl[0]) < .01 && abs(motorControl[1]) < .01) {


    Serial.println("Received STOP");
    Serial.println(motorControl[1]);

    for (int i = 0; i < 4; i++) {
      ledcWrite(motorPINEN[i], 0);
      motorVelSetpoint[i] = 0.0;
    }
    return;

  }

  Serial.print("Linear Vel: ");
  Serial.println(lin_vel);
  Serial.print("Angular Vel: ");
  Serial.println(ang_vel);
  //***************WIFI*************//

  calcMotorVelSetpoint(lin_vel, ang_vel);
  controlMotor(lin_vel, ang_vel);
  delay(50);
}
```

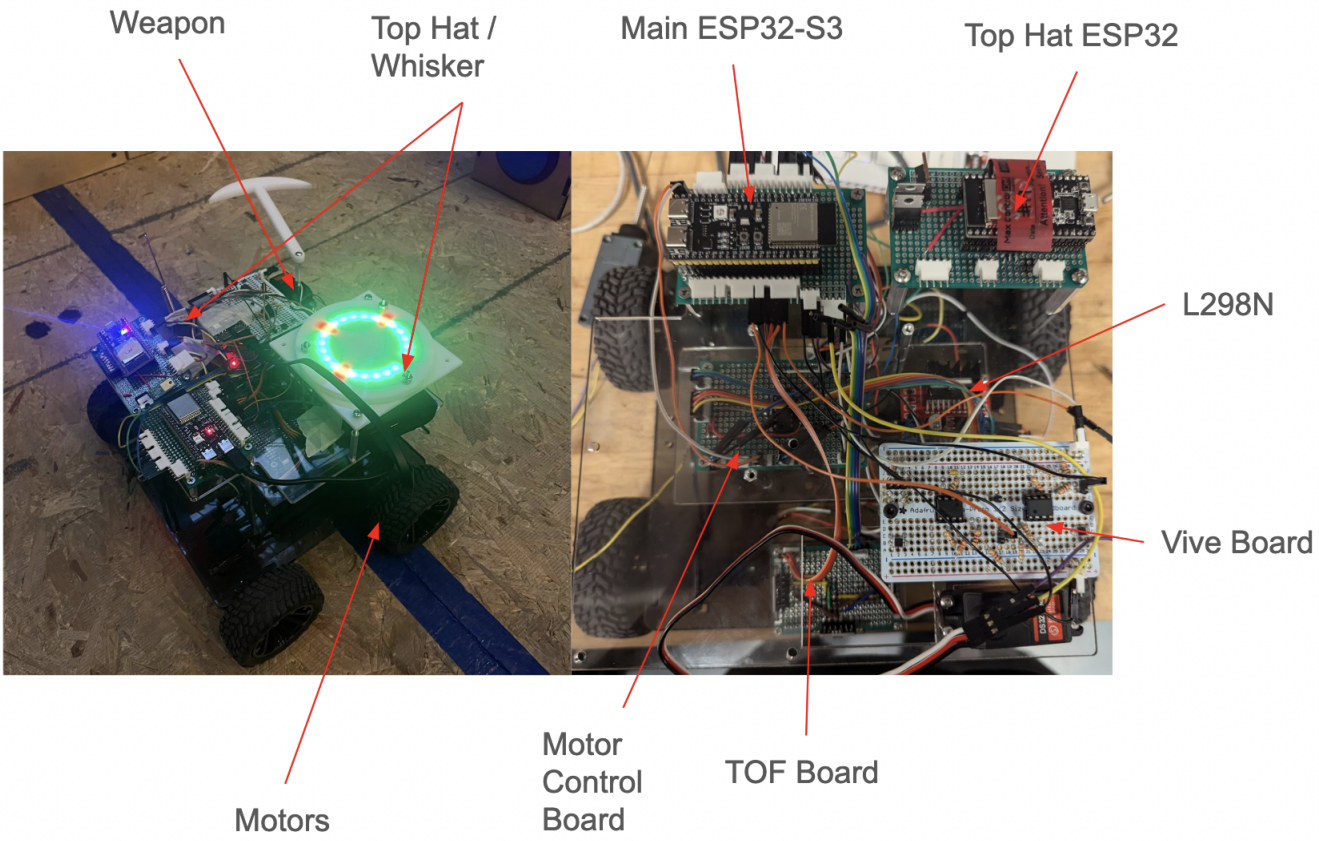
6.3 index.h

```
/* Project Name: Lab 4
 * Author: Oluwagbemiga Oloyede, Kashish Garg, Dhyey Shah
 * License: You may use, distribute and modify this code under the terms of the GNU GPLv3.0 license
 * Name: 4.2
 */

const char htmlPage[] PROGMEM = R"===(
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Robot Car Controller</title>
  <style>
    body {
      display: flex;
      flex-direction: column;
      align-items: center;
      font-family: Arial, sans-serif;
      background-color: #f2f2f2;
      margin: 0;
      padding: 20px;
    }

    .interface {
      display: flex;
      flex-direction: row;
      padding: 20px;
      justify-content: space-between;
    }

    h1 {
      color: #333;
    }

    .controls {
      display: grid;
      grid-template-areas:
        ". up ."
        "left stop right"
        ". down .";
      gap: 20px;
    }

    .controls button {
      width: 80px;
      height: 80px;
      font-size: 18px;
      font-weight: bold;
      cursor: pointer;
      border: none;
      border-radius: 10px;
      color: #fff;
      transition: 0.3s;
    }

    .controls-div {
      display: flex;
      flex-direction: column;
      align-items: center;
      justify-content: center;
      margin-right: 120px;
    }

    .up {
      grid-area: up;
      background-color: #4CAF50;
    }

    .down {
      grid-area: down;
      background-color: #e74c3c;
    }

    .left {
      grid-area: left;
      background-color: #3498db;
    }

    .right {
      grid-area: right;
      background-color: #f39c12;
    }

    .stop {
      grid-area: stop;
      background-color: #555;
    }

    .speed-control {
```

```
      margin-top: 20px;
      display: flex;
      flex-direction: column;
      align-items: center;
      justify-content: center;
    }

    .speed-control label {
      margin-bottom: 10px;
      font-size: 18px;
      color: #333;
    }

    .speed-control input[type="range"] {
      width: 300px;
    }

    .extra-controls {
      margin-top: 20px;
      display: flex;
      flex-direction: column;
      align-items: center;
    }

    .extra-controls button {
      width: 120px;
      height: 40px;
      font-size: 16px;
      font-weight: bold;
      cursor: pointer;
      border: none;
      border-radius: 10px;
      color: #fff;
      transition: 0.3s;
      margin: 10px 0;
    }

    .follow-wall {
      background-color: #8e44ad; /* Purple */
    }

    .attack {
      background-color: #c0392b; /* Red */
    }

    .target {
      background-color: #27ae60; /* Green */
    }

    @media (max-width: 600px) {
      body {
        padding: 20px;
      }

      .interface {
        display: flex;
        flex-direction: row;
        padding: 20;
        justify-content: space-between;
        transform: rotate(-270deg);
        padding-left: 50vh;
      }

      .controls button {
        width: 65px;
        height: 65px;
        font-size: 18px;
      }

      .controls-div {
        display: flex;
        padding: 10px;
        align-items: center;
        justify-content: center;
        margin-right: 20px;
      }

      #header {
        display: none;
      }

      .extra-controls button {
        width: 80px;
        height: 30px;
        font-size: 14px;
      }
    }
  </style>
</head>

<body>
```

```
  <div id="header">
    <h1 id="header-elem">Robot Car Controller</h1>
  </div>
  <div class="interface">
    <div class="controls-div">
      <div class="controls">
        <button class="up" onclick="sendCommand('1')">Up</button>
        <button class="left" onclick="sendCommand('2')">Left</button>
        <button class="stop" onclick="sendCommand('0')">Stop</button>
        <button class="right" onclick="sendCommand('3')">Right</button>
        <button class="down" onclick="sendCommand('-1')">Down</button>
      </div>

      <!-- New extra controls section -->
      <div class="extra-controls">
        <button class="follow-wall" onclick="sendCommand('5')">Follow Wall</button>
        <button class="attack" onclick="sendCommand('4')">Attack</button>
        <button class="target" onclick="sendCommand('6')">Target</button>
      </div>
    </div>

    <div class="speed-control">
      <label for="speed">Speed Control</label>
      <input type="range" id="speed" min="0" max="100" value="50" oninput="updateSpeed(this.value)">
      <span id="speed-display">50</span>%
    </div>
  </div>

  <script>
    function sendCommand(command) {
      console.log(`sending command: ${command}`);
      var xhttp = new XMLHttpRequest();
      var str = "setDirection?dir="
      var res = str.concat(command);
      xhttp.open("GET", res, true);
      xhttp.send();
    }

    function updateSpeed(speed) {
      var xhttp = new XMLHttpRequest();
      var str = "setSpeed?value=";
      var res = str.concat(speed);
      xhttp.open("GET", res, true);
      xhttp.send();
      document.getElementById('speed-display').textContent = speed;
      console.log(`setting speed to: ${speed}`);
    }
  </script>
</body>
</html>
)===";
```

6.4 BOM



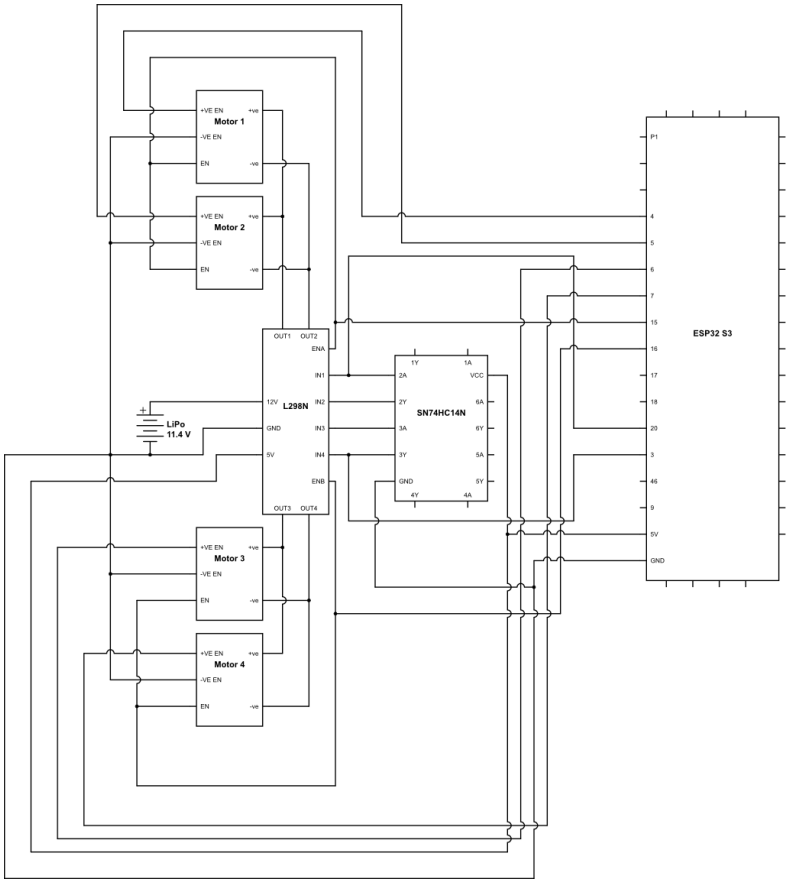
Subcomponent	Component	Qty
Top Hat		
	Top Hat ESP32 Board	1
	Top Hat ESP32	1
	Whisker	1
Main ESP32-S3		
	ESP32-S3	1
	Molex Connectors	
Motor and Control		
	L298N	1
	Perf Board	1
	JGA25-371 Motors and encoders	4
	Hobby Park 2.9 in Wheels	4
	4mm Coupler Shaft	4

Continued on next page

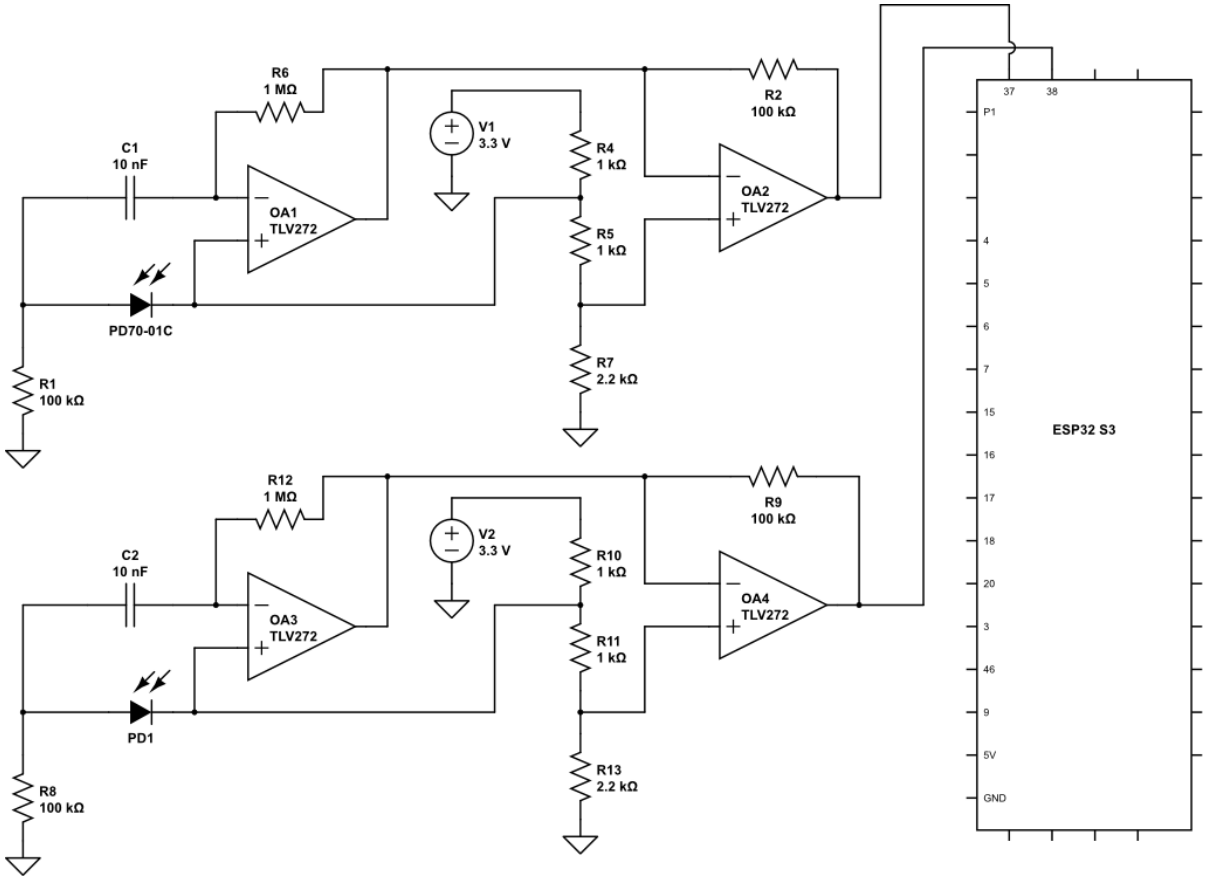
Subcomponent	Component	Qty
TOF Board		
	Perf Board	1
	VL53L0X TOF	3
Vive Board		
	Perf Board	1
	TLV272	2
	PD70-01C	2
Weapon		
	DSSERVO 20 kgcm Servo	1
	Acrylic Minecraft Picaxe	1

6.5 Electrical Schematics

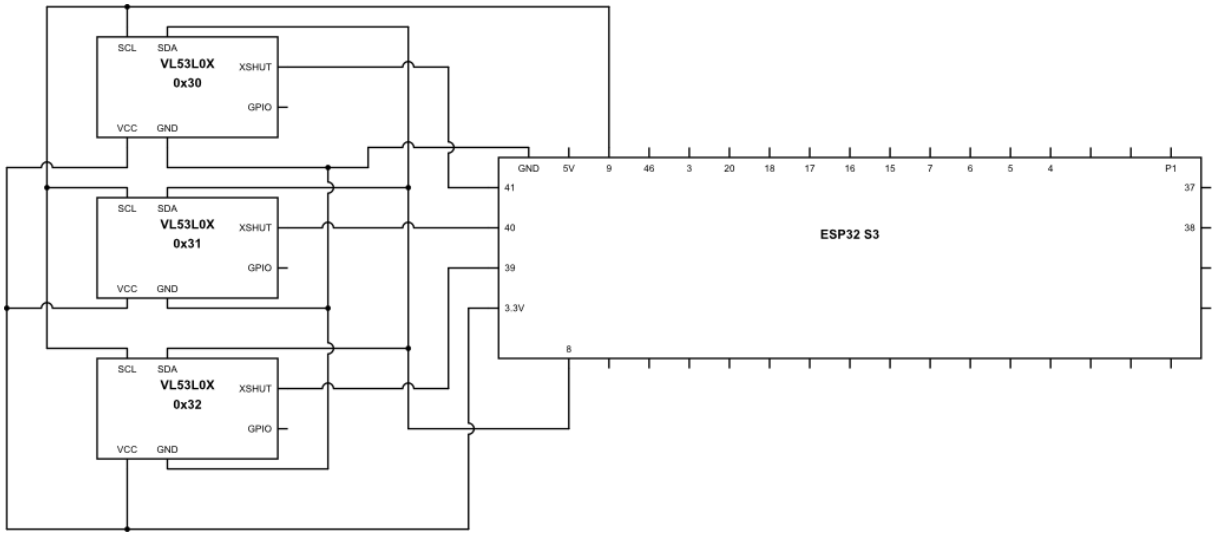
6.5.1 Motor Driver Circuit



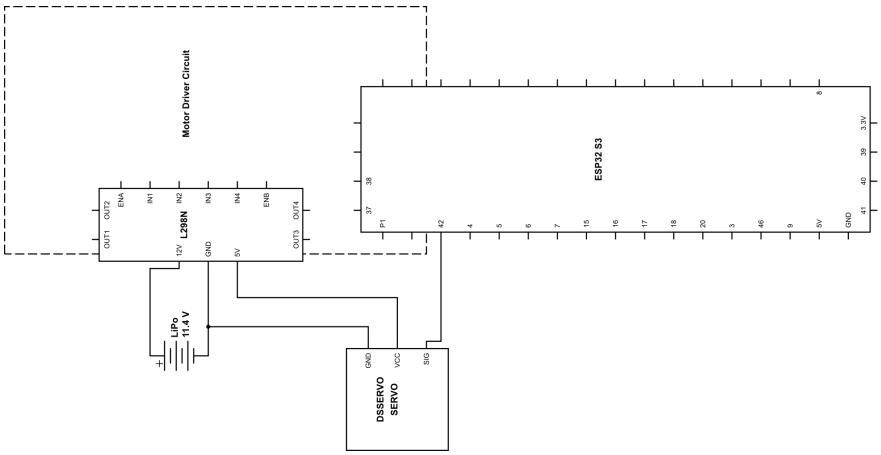
6.5.2 Vive Detection Circuit



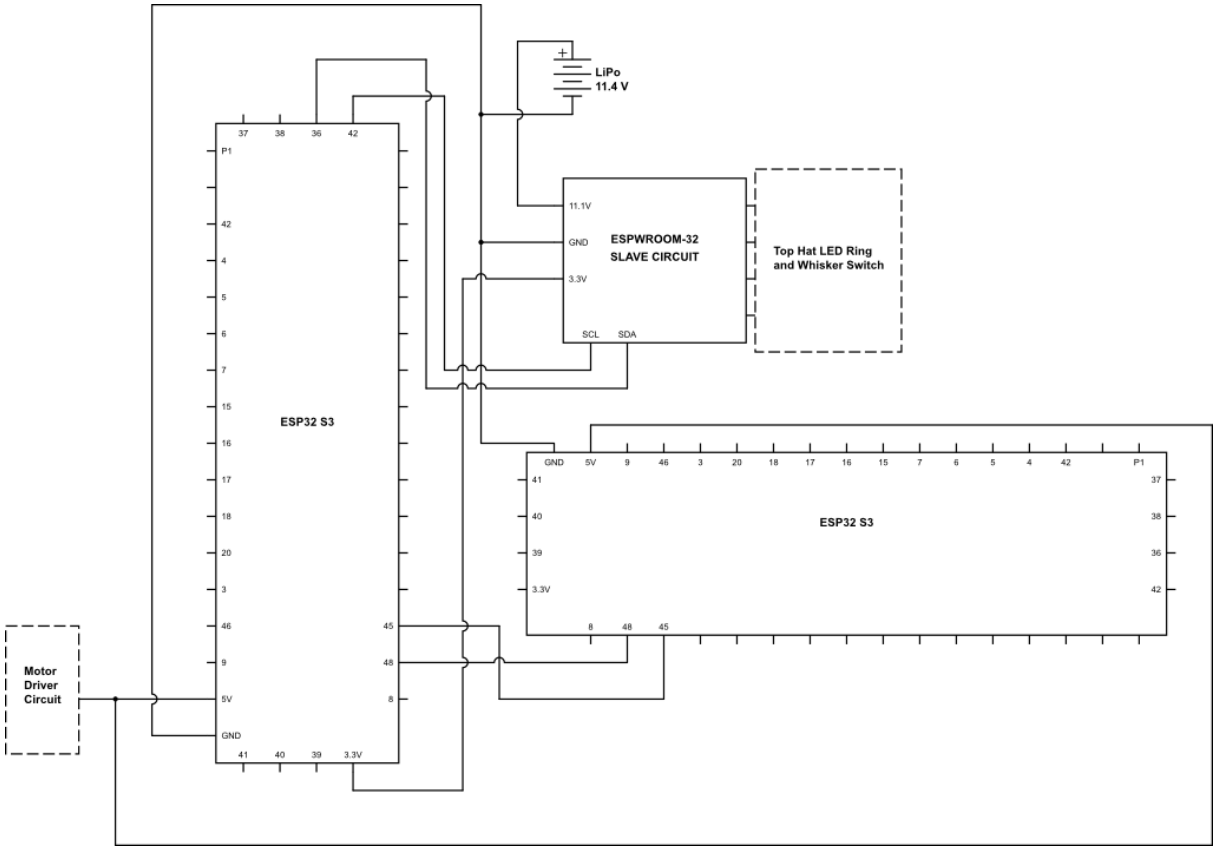
6.5.3 Wall Following with ToFs



6.5.4 Attacking Arm Servo Circuit

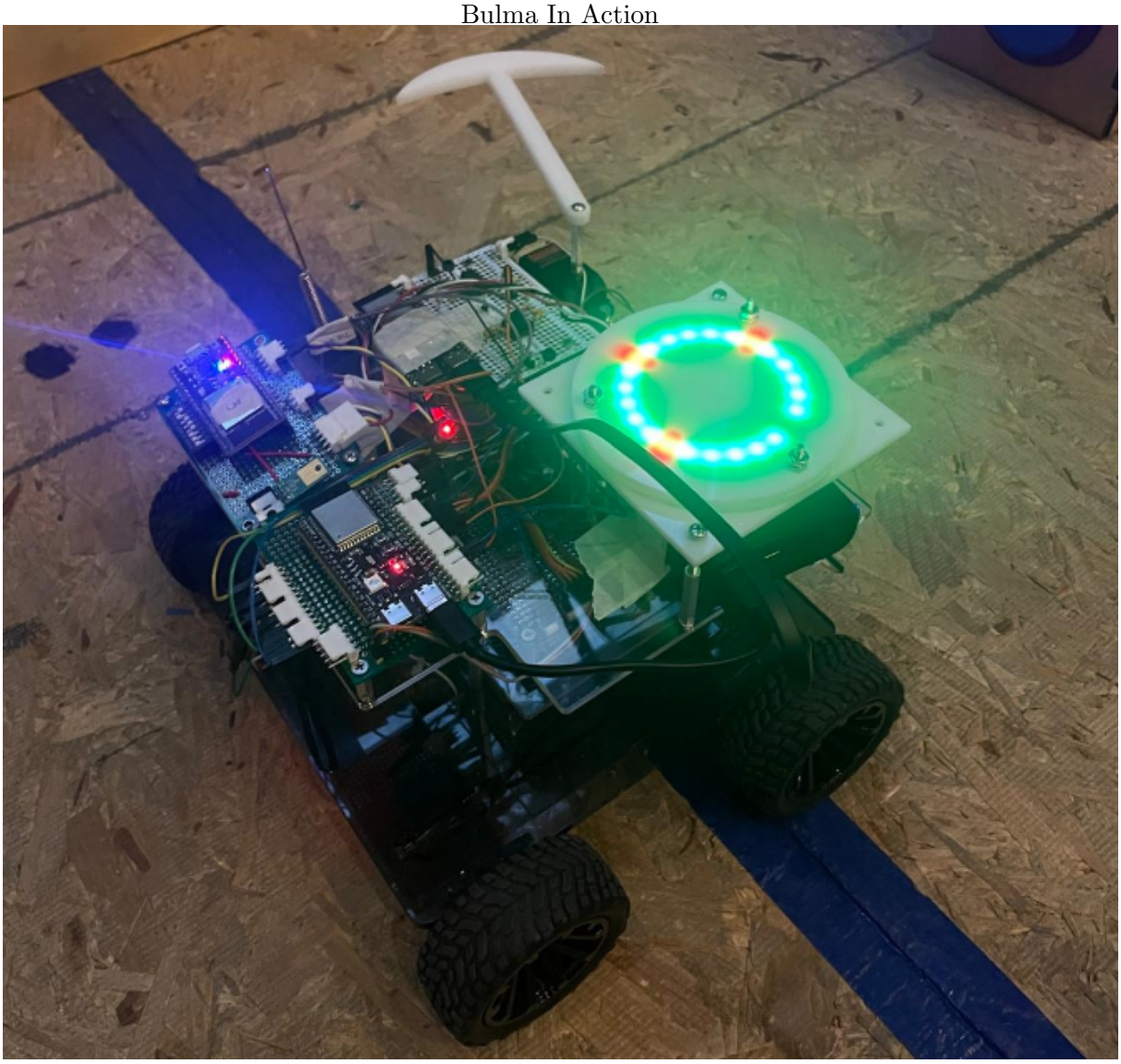


6.5.5 I2C Top Hat Circuit



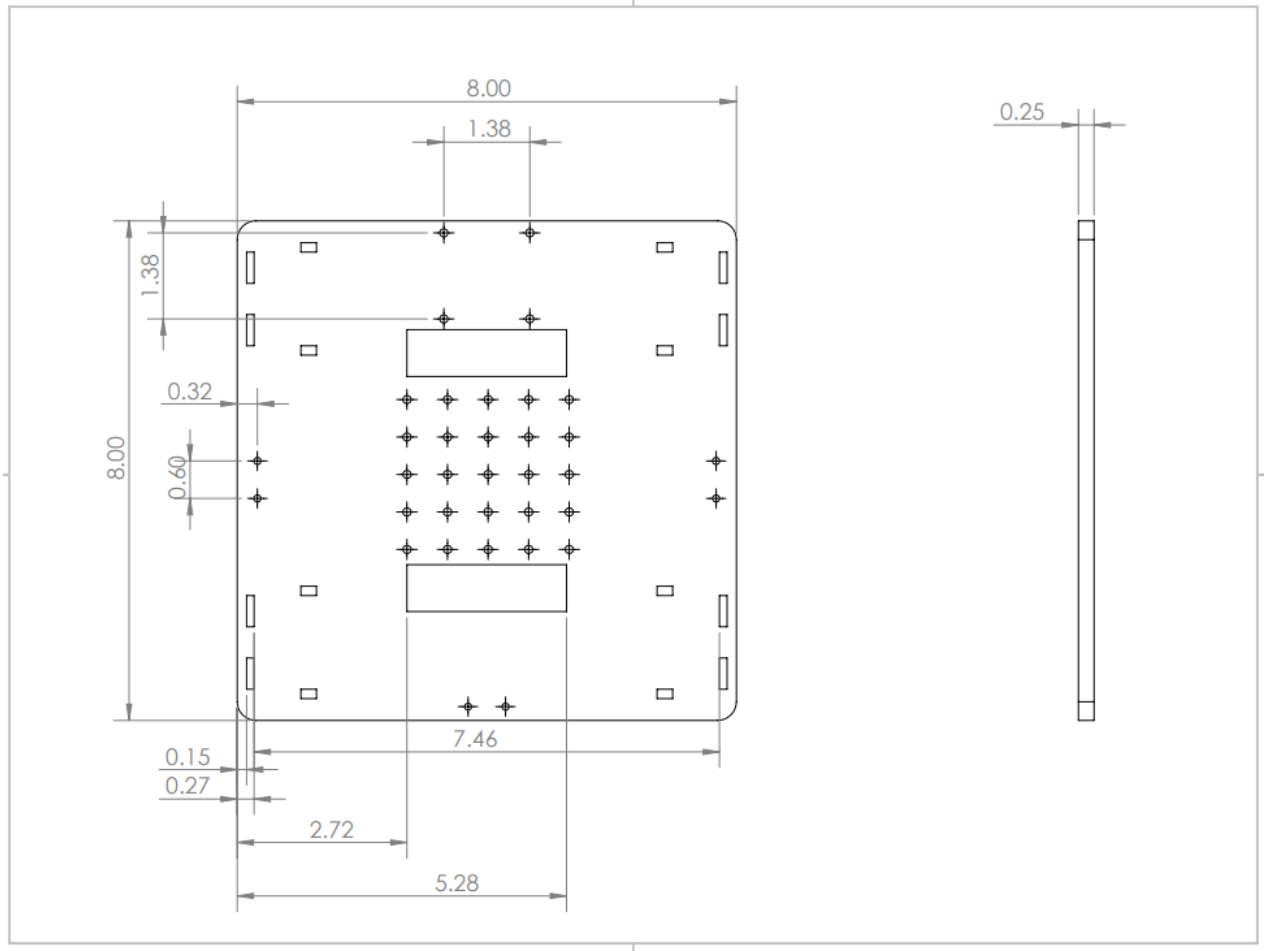
6.6 Bulma Photo

Bulma In Action

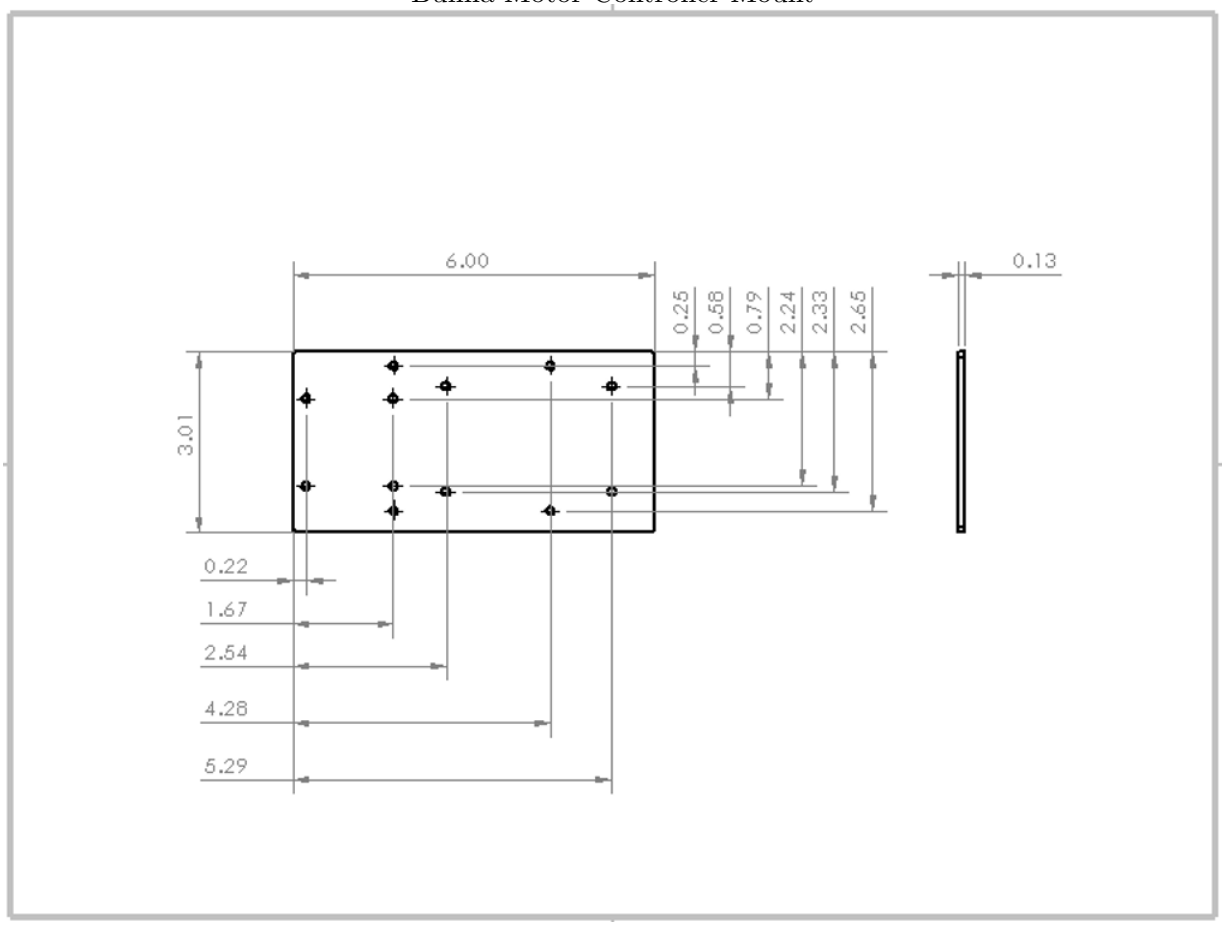


6.7 CAD

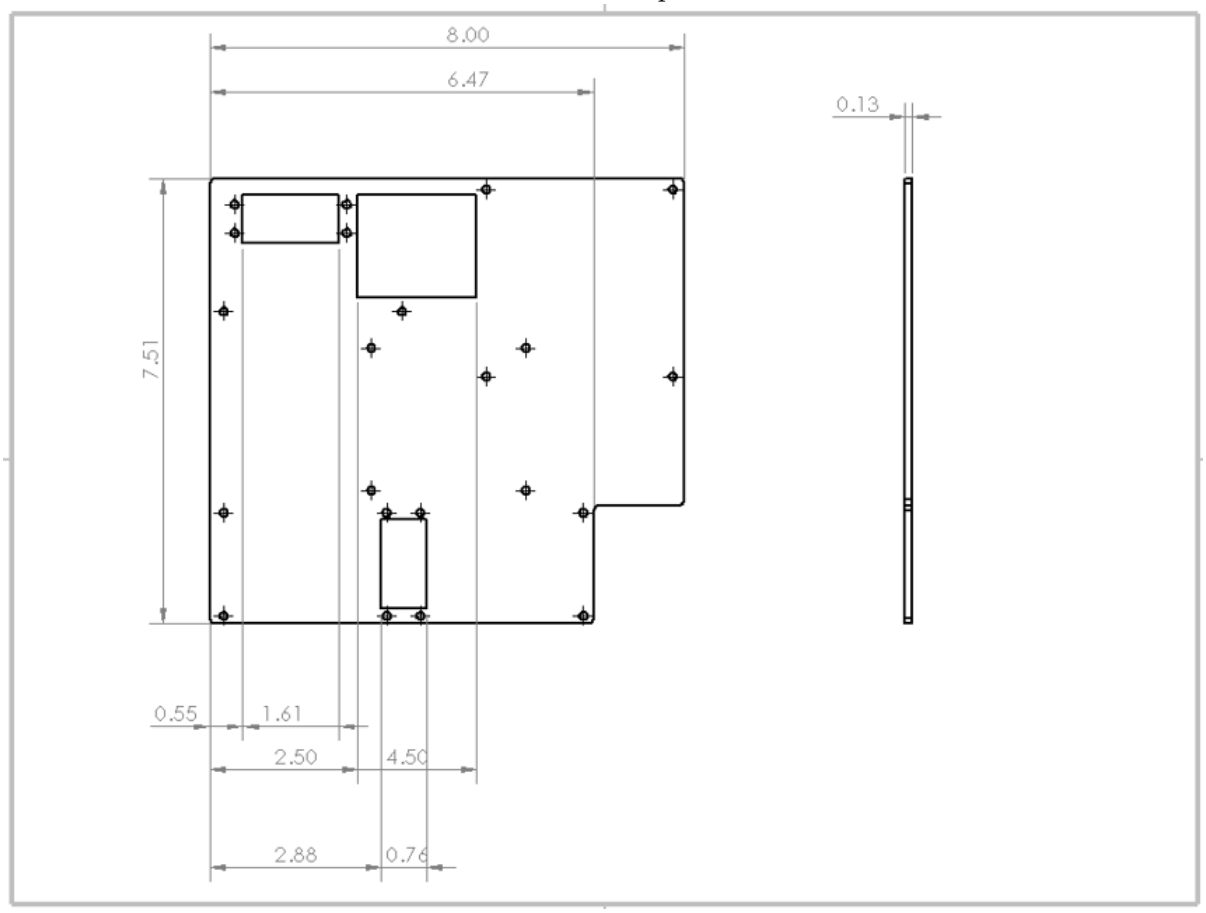
Bulma Base



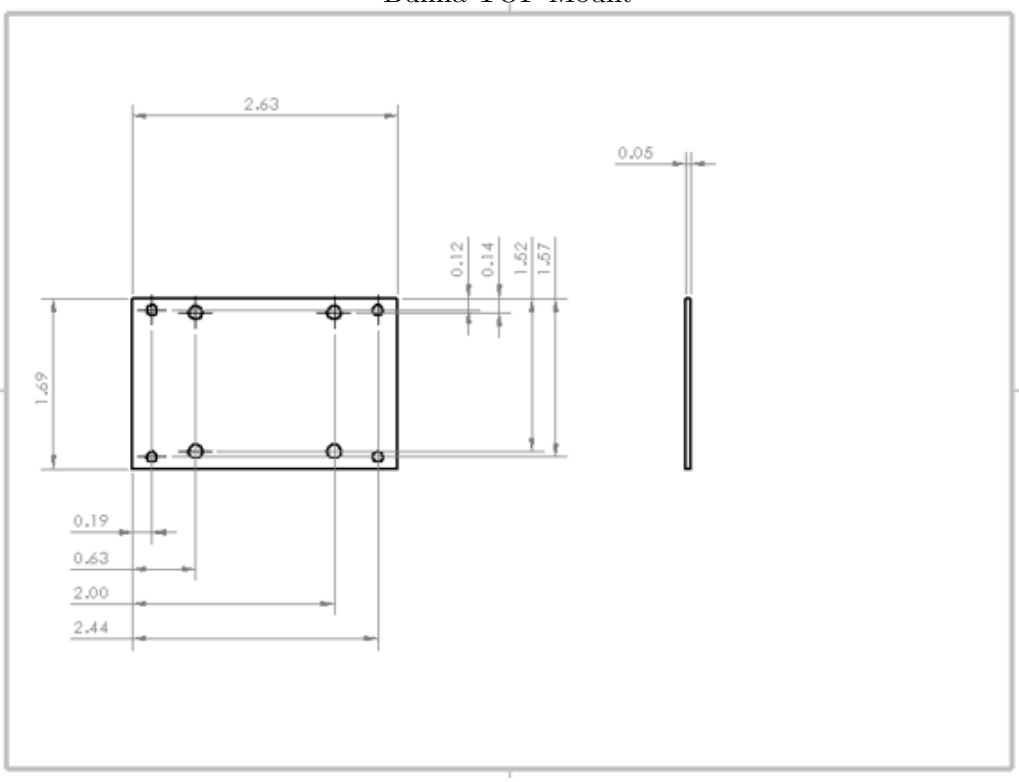
Bulma Motor Controller Mount



Bulma ESP32 and Top Hat Mount



Bulma TOF Mount



6.8 Datasheets

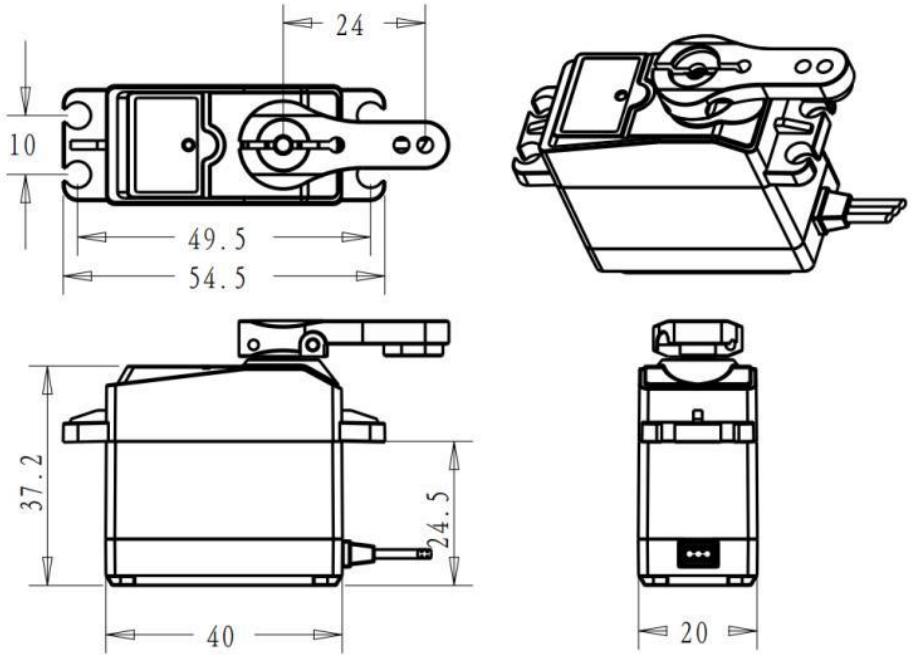
Component	Datasheet
ESP32-S3	https://tinyurl.com/5edsm9zs
L298N	https://tinyurl.com/3a6end5h
JGA25-371 Motors and encoders	https://tinyurl.com/fcyszzvh
VL53L0X TOF	https://www.pololu.com/file/0J1187/vl53l0x.pdf
TLV272	https://tinyurl.com/mry74k8h
PD70-01C	https://tinyurl.com/3c47pf88
DSSERVO 20 kgcm Servo	Attached Below

ANNIMOS DS-Model Servo

(Product Name): DS3218MG

(Product Description): 6.8V 20kg Digital Servo

(Drawing)



1. Apply Environmental Condition

No.	Item	Specification
1-1	Storage Temperature Range	-20℃～60℃
1-2	Operating Temperature Range	-10℃～50℃
1-3	Operating Voltage Range	4.8-6.8V

2.Mechanical Specification

No.	Item	Specification
2-1	Size	40*20*40.5mm
2-2	Weight	60g
2-3	Gear ratio	250
2-4	Bearing	Double bearing
2-5	Connector wire	400±5mm
2-6	Motor	3-pole(s)
2-7	Waterproof performance	IP66

Any quesiton please contact with us: servomaker@hotmail.com

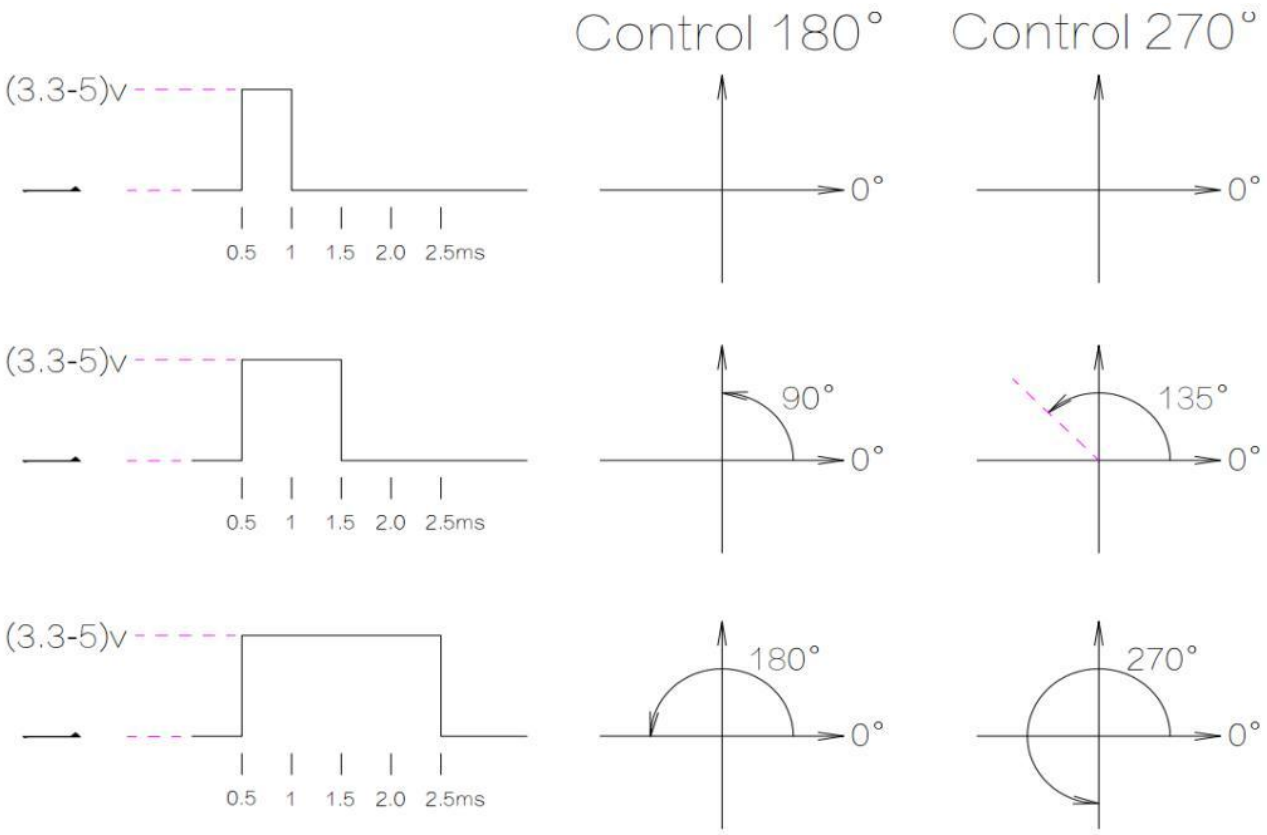
3. Electrical Specification

No.	Operating Voltage	5V	6.8V	
3-1	Idle current(at stopped)	4mA	5mA	
3-2	Operating speed (at no load)	0.16 sec/60°	0.14 sec/60°	°
3-3	Stall torque (at locked)	19 kg-cm	21.5 kg-cm	
3-4	Stall current (at locked)	1.5A	1.8 A	

4. Control Specification

No.	Item	Specification
4-1	Control System	PWM(Pulse width modification)
4-2	Pulse width range	500～2500μsec
4-3	Neutral position	1500μsec
4-4	Running degree	270° (when 500～2500 μ sec)
4-5	Dead band width	3 μsec
4-6	Operating frequency	50-330Hz
4-7	Rotating direction	Counterclockwise (when 500～2500 μsec)

5. About PWM Control



Any question please contact with us: servomaker@hotmail.com

7 Links to Videos

By the competition day, our Robot was capable of demonstrating all the required criteria, including manual and autonomous control modes, which we already showed and got verified from the TA in the check off. Here are some of the videos we took of our robots' performance on a few of the tasks:

- Wall Follow: Video
- Autonomous Target 1: Video
- Top Hat Working: Video
- BULMA in Action (Competition): Video1 Video2

We Ranked 1 in the competition and had a lot of fun competing!