MEAM 510 Group 34 Lab 4 Report

4.2.2

The video is sped up for viewer's convenience.

https://youtu.be/G-gv486vKTs

4.2.2.1

No PID:

https://youtu.be/H_RXuE_E-K8

With PID:

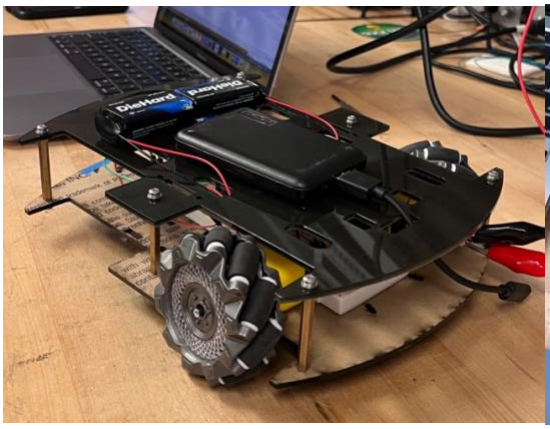
https://youtu.be/oWnLqM4T6kw

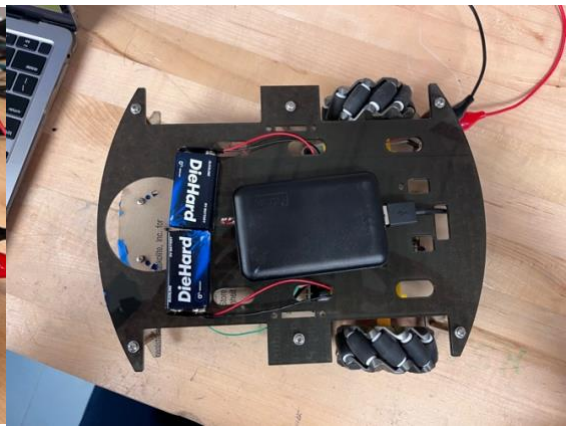
The robot drifted at the beginning of the video due to the caster wheel. After this motion is corrected, the motors will try to align its trajectory to keep going straight. Our PID control adjusts the duty cycle of each motor to reach a certain velocity (ticks per 100ms) set in the code. We had weight balance issues, so the car was moving rather slowly. Otherwise, the motors and code work as intended to keep the motors going at the same speed. The video is sped up for the viewer's convenience the actual time it took for the robot to complete 1 meter with our PID controls is 2 minutes.

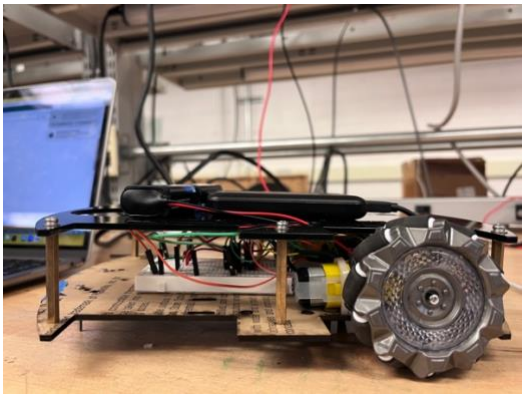
4.2.3

1. Our mobile base was intended for a four-motor drive car. The body is wide enough to house multiple components on each level. For the first iteration, the first level contains the power bank and motors, while the second level contains of our breadboard, microcontroller, h bridges, and batteries for motors. We knew that we were going to perform modifications on our circuitry, so we placed the breadboard and the motor batteries on the top surface.

   In the end, we got better results using two motors with a caster wheel instead of four wheels. We simply removed the two motors that we were not using but kept the base. We also switched the placements of the breadboard and batteries during our race we saw that our wires were vulnerable. Below you can see our final design for this lab.







2. Our race time was about three minutes. Its performance was unpredictable. For example, when we command the car to go straight sometimes it would only sometimes move straight. Each side of the motor would stop working suddenly, but when picked up it would run as intended. Sometimes the motors would need a slight nudge to start rotating. The motors would not work as intended when placed on the ground too. Both motors work fine when it is not externally loaded. They become responsive to the directions we input them and are fast at rotating when picked up. At the end the race had to be completed by always adjusting its trajectory in every step. It was almost as if we were squiggling around to get to the end.

   The first improvement we made was to secure the wiring of the car. The motors would suddenly stop and sometimes it would be due to wires popping out. Hence why we switched the placement of the breadboard and the batteries. The second improvement was driving two motors with just one h bridge. Somehow using two h bridges caused noise in each of the 0V direction line that messed up the motor's movement logic. The third improvement we made was to supply the h bridge with two 9V batteries in parallel. It added necessary capacitance to drive the two motors. We connected the grounds of the motor to the ground of the h bridge. This further reduced the noise of the directional inputs on our motors and made movements more precise on the track.

3. Codes

Code for race:

```
#include "html510.h"
HTML510Server h(80);

#define M1_PWM 1
#define M1_DIR1 4
#define M1_DIR2 5

#define M2_PWM 10
#define M2_DIR1 18
#define M2_DIR2 19

#define LEDC_CHANNEL0 0
#define LEDC_CHANNEL1 1
#define LEDC_RESOLUTION_BITS 14
#define LEDC_RESOLUTION ((1 << LEDC_RESOLUTION_BITS) - 1)
#define LEDC_FREQ_HZ 100

int duty = 0;
int mval = 0;

const char* ssid     = "daudi";
const char* password = "123456789";
IPAddress ip(192, 168, 1, 1);

// Your HTML body
const char body[] PROGMEM = R"===(
<!DOCTYPE html>
<html>

<head>
  <style>
    .container {
      text-align: center;
    }
```

```
    .arrow-btn {
      background-color: #4CAF50;
      border: none;
      color: white;
      text-align: center;
      text-decoration: none;
      display: inline-block;
      font-size: 16px;
      margin: 0px;
      cursor: pointer;
    }

    .stop-btn {
      background-color: #f44336;
      border: none;
      color: white;
      text-align: center;
      text-decoration: none;
      display: inline-block;
      font-size: 16px;
      margin: 0px;
      cursor: pointer;
    }

    .btn-group {
      margin-top: 0px;
    }

    .arrow-btn:active,
    .stop-btn:active {
      background-color: #45a049;
    }
  </style>
</head>

<body>
  <div class="container">
```

```
    <h1>
      <br>
      <div class="btn-group">
        <button class="arrow-btn up-btn" style="padding: 39px 28px; margin-bottom:-100px"
onmousedown="startCommand(4)" onmouseup="stopCommand()">W</button>
      </div><br>
      <div class="btn-group">
        <button class="arrow-btn left-btn" style="padding: 18px 43px;" onmousedown="startCommand(2)"
onmouseup="stopCommand()">A</button>
        <button class="stop-btn" style="padding: 20px 30px;" onclick="sendCommand(5)">STOP</button>
        <button class="arrow-btn right-btn" style="padding: 18px 43px;" onmousedown="startCommand(3)"
onmouseup="stopCommand()">D</button>
      </div><br>
      <div class="btn-group">
        <button class="arrow-btn down-btn" style="padding: 45px 30px; margin-top: -100px;"
onmousedown="startCommand(1)" onmouseup="stopCommand()">S</button>
      </div>
      <br>
      <span id="mValue"></span><br>
    </h1>
  </div>
  <script>
    var isButtonPressed = false;

    function startCommand(direction) {
      if (!isButtonPressed) {
        isButtonPressed = true;
        sendCommand(direction);
      }
    }

    function stopCommand() {
      isButtonPressed = false;
    }

    function sendCommand(direction) {
      var xhttp = new XMLHttpRequest();
```

```
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("mValue").innerHTML = this.responseText;
      if (isButtonPressed) {
        sendCommand(direction);
      }
    }
  };
  var str = "mSlider?val="; // Update this to match your slider handler
  var res = str.concat(direction);
  xhttp.open("GET", res, true);
  xhttp.send();
}

document.addEventListener("keydown", function(event) {
  switch (event.key) {
    case 'w':
      startCommand(4); // Forward
      break;
    case 'a':
      startCommand(2); // Left
      break;
    case 'd':
      startCommand(3); // Right
      break;
    case 's':
      startCommand(1); // Backward
      break;
  }
});



document.addEventListener("keyup", function(event) {
  switch (event.key) {
    case 'w':
      stopCommand(); // Stop when the key is released
      break;
```

```
      case 'a':
        stopCommand(); // Stop when the key is released
        break;
      case 'd':
        stopCommand(); // Stop when the key is released
        break;
      case 's':
        stopCommand(); // Stop when the key is released
        break;
    }
  });
 </script>
</body>

</html>

)===";


void handleRoot() {
 h.sendhtml(body);
}

void handleSliderM() {
  mval = h.getVal();

  if (mval == 1) { // BACK
   duty = 16300;
   ledcAttachPin(M1_PWM, LEDC_CHANNEL0); // M1 BACK
   ledcSetup(LEDC_CHANNEL0, LEDC_FREQ_HZ, LEDC_RESOLUTION_BITS);
   ledcWrite(LEDC_CHANNEL0, duty);
   digitalWrite(M1_DIR1, LOW);
   digitalWrite(M1_DIR2, HIGH);

   ledcAttachPin(M2_PWM, LEDC_CHANNEL1); // M2 BACK
   ledcSetup(LEDC_CHANNEL1, LEDC_FREQ_HZ, LEDC_RESOLUTION_BITS);
   ledcWrite(LEDC_CHANNEL1, duty);
```

```
  digitalWrite(M2_DIR1, HIGH);
  digitalWrite(M2_DIR2, LOW);


} else if (mval == 2) { // LEFT
  duty = 16300;
  ledcAttachPin(M1_PWM, LEDC_CHANNEL0); // M1 BACK
  ledcSetup(LEDC_CHANNEL0, LEDC_FREQ_HZ, LEDC_RESOLUTION_BITS);
  ledcWrite(LEDC_CHANNEL0, duty);
  digitalWrite(M1_DIR1, LOW);
  digitalWrite(M1_DIR2, HIGH);

  ledcAttachPin(M2_PWM, LEDC_CHANNEL1); // M2 FORW
  ledcSetup(LEDC_CHANNEL1, LEDC_FREQ_HZ, LEDC_RESOLUTION_BITS);
  ledcWrite(LEDC_CHANNEL1, duty);
  digitalWrite(M2_DIR1, LOW);
  digitalWrite(M2_DIR2, HIGH);

} else if (mval == 3) { // RIGHT
  duty = 16300;
  ledcAttachPin(M1_PWM, LEDC_CHANNEL0); // M1 FORW
  ledcSetup(LEDC_CHANNEL0, LEDC_FREQ_HZ, LEDC_RESOLUTION_BITS);
  ledcWrite(LEDC_CHANNEL0, duty);
  digitalWrite(M1_DIR1, HIGH);
  digitalWrite(M1_DIR2, LOW);

  ledcAttachPin(M2_PWM, LEDC_CHANNEL1); // M2 BACK
  ledcSetup(LEDC_CHANNEL1, LEDC_FREQ_HZ, LEDC_RESOLUTION_BITS);
  ledcWrite(LEDC_CHANNEL1, duty);
  digitalWrite(M2_DIR1, HIGH);
  digitalWrite(M2_DIR2, LOW);

} else if (mval == 4) { // FORWARD
  duty = 16300;
  ledcAttachPin(M1_PWM, LEDC_CHANNEL0); // M1 FORW
  ledcSetup(LEDC_CHANNEL0, LEDC_FREQ_HZ, LEDC_RESOLUTION_BITS);
  ledcWrite(LEDC_CHANNEL0, duty);
```

```
    digitalWrite(M1_DIR1, HIGH);
    digitalWrite(M1_DIR2, LOW);

    ledcAttachPin(M2_PWM, LEDC_CHANNEL1); // M2 FORW
    ledcSetup(LEDC_CHANNEL1, LEDC_FREQ_HZ, LEDC_RESOLUTION_BITS);
    ledcWrite(LEDC_CHANNEL1, duty);
    digitalWrite(M2_DIR1, LOW);
    digitalWrite(M2_DIR2, HIGH);

   } else if (mval == 5){ // STOP
    duty = 0;
    ledcAttachPin(M1_PWM, LEDC_CHANNEL0); // M1 BACK
    ledcSetup(LEDC_CHANNEL0, LEDC_FREQ_HZ, LEDC_RESOLUTION_BITS);
    ledcWrite(LEDC_CHANNEL0, duty);
    digitalWrite(M1_DIR1, LOW);
    digitalWrite(M1_DIR2, LOW);

    ledcAttachPin(M2_PWM, LEDC_CHANNEL1); // M2 BACK
    ledcSetup(LEDC_CHANNEL1, LEDC_FREQ_HZ, LEDC_RESOLUTION_BITS);
    ledcWrite(LEDC_CHANNEL1, duty);
    digitalWrite(M2_DIR1, LOW);
    digitalWrite(M2_DIR2, LOW);
   }
}

void setup() {
  IPAddress myIP(192, 168, 1, 214);
  Serial.begin(115200);

  Serial.print("Access point "); Serial.print(ssid);
  WiFi.softAP(ssid);
  WiFi.softAPConfig(myIP, IPAddress(192, 168, 1, 1), IPAddress(255, 255, 255, 0));
  Serial.print(" AP IP address"); Serial.println(myIP);

  h.begin();
  h.attachHandler("/ ", handleRoot);
  h.attachHandler("/mSlider?val=", handleSliderM);
```

```
  // Initialize pin modes
  pinMode(M1_PWM, OUTPUT);
  pinMode(M1_DIR1, OUTPUT);
  pinMode(M1_DIR2, OUTPUT);

  pinMode(M2_PWM, OUTPUT);
  pinMode(M2_DIR1, OUTPUT);
  pinMode(M2_DIR2, OUTPUT);
}

void loop() {
  h.serve();
  delay(10);
}
```

PID Code:

```
#include <PID_v1.h> // using a PID library


const int motor1PWM = 7;  // PWM pin for motor 1
const int motor2PWM = 1; // PWM pin for motor 2
const int encoder1PinA = 4; // encoder pin for motor 1
const int encoder2PinA = 19; // encoder pin for motor 2
const int motor1channel = 1;
const int motor2channel = 0;
const int pin1 = 6;     //direction pin1
const int pin2 = 5; //direction pin2
```

```
int count1=0; // counter for ticks for encoder 1
int count2=0; // counter for ticks for encoder 2
int totcount =0; // total ticks to measure distance

// timer to find ticks per 0.1 second
unsigned long startTime = 0;
unsigned long endTime = 0;
unsigned long elapsedTime;



// PID parameters found by gruelling hit and trial
double Kp = 5;
double Ki = 2;
double Kd = 0.002;

double setpoint =8;  // Setpoint ticks

double input1, output1;
double input2, output2;

PID pid1(&input1, &output1, &setpoint, Kp, Ki, Kd, DIRECT);
PID pid2(&input2, &output2, &setpoint, Kp, Ki, Kd, DIRECT);

void setup() {
  pinMode(motor1PWM, OUTPUT);pinMode(motor2PWM, OUTPUT);
  pinMode(pin1, OUTPUT);pinMode(pin2, OUTPUT);

  Serial.begin(115200);
  pinMode(encoder1PinA, INPUT);
  pinMode(encoder2PinA, INPUT);


  // Attach interrupts for encoder counting
  attachInterrupt(digitalPinToInterrupt(encoder1PinA), updateEncoder1, CHANGE);
  attachInterrupt(digitalPinToInterrupt(encoder2PinA), updateEncoder2, CHANGE);
```

```
  // Setting up PWM pins for motor control
  pinMode(motor1PWM, OUTPUT);
  ledcSetup(motor1channel, 30, 14);
  ledcAttachPin(motor1PWM, motor1channel);
  pinMode(motor2PWM, OUTPUT);
  ledcSetup(motor2channel, 30, 14);
  ledcAttachPin(motor2PWM, motor2channel);


  // Initialize PID controllers
  pid1.SetMode(AUTOMATIC);
  pid2.SetMode(AUTOMATIC);
}

void loop() {
 while(totcout<=320){ // to limit running to 1meter
 // calculation = (ticks per rotation)*(1m/circumfrence of wheel in meters)

  unsigned long currentTime = millis();
  if (currentTime-startTime>=100){ // checking if 0.1 seconds have elapsed
    input1 = count1; // ticks from motor 1 in 100 ms
    count1=0; //resetting counter
    input2 = count2; // ticks from motor 2 in 100 ms
    count2=0; //resetting counter

    startTime=currentTime; //resetting time


  }
  // setting a direction (we dont need it to change for the PID test)
  digitalWrite(pin1, HIGH);
  digitalWrite(pin2, LOW);

  // Update PID inputs with encoder values
  pid1.Compute();
  pid2.Compute();
```

```
  // Apply PID outputs to motor control
  int dc1 = output1;
  int dutyCycle1 = map(dc1, 9, 20, 0, 16384); // the 9 and 20 values were got from observing the motors running at 6 volts and 9 volts
  ledcWrite(motor1channel, dutyCycle1);
  int dc2 = output2;
  int dutyCycle2 = map(dc2, 9, 20, 0, 16384);
  ledcWrite(motor2channel, dutyCycle2);

  // Printing ticks per 0.1 seconds
  Serial.print("Motor 1 Speed: ");
  Serial.print(input1);

  Serial.print(" | Motor 2 Speed: ");
  Serial.print(input2);

  Serial.println();
  Serial.print(totcount);
  Serial.println();
  }
  //stop motors after a meter

  ledcWrite(motor1channel, 0);
  ledcWrite(motor2channel, 0);


}
//updating ticks
void updateEncoder1() {

    count1++;
    totcount++;

}

void updateEncoder2() {
```

```
  count2++;

}
```

4. Bill of Materials

Item	Cost per count ($)	Count	Total for item	Grand Total
Mecanum Wheels	30.99	1	30.99	
9V batteries (2 pack)	4.99	1	4.99	
Powerbank	8.00	1	8.00	
Caster Wheels	7.99	1	7.99	51.97

Item Links:

Mecanum Wheels:

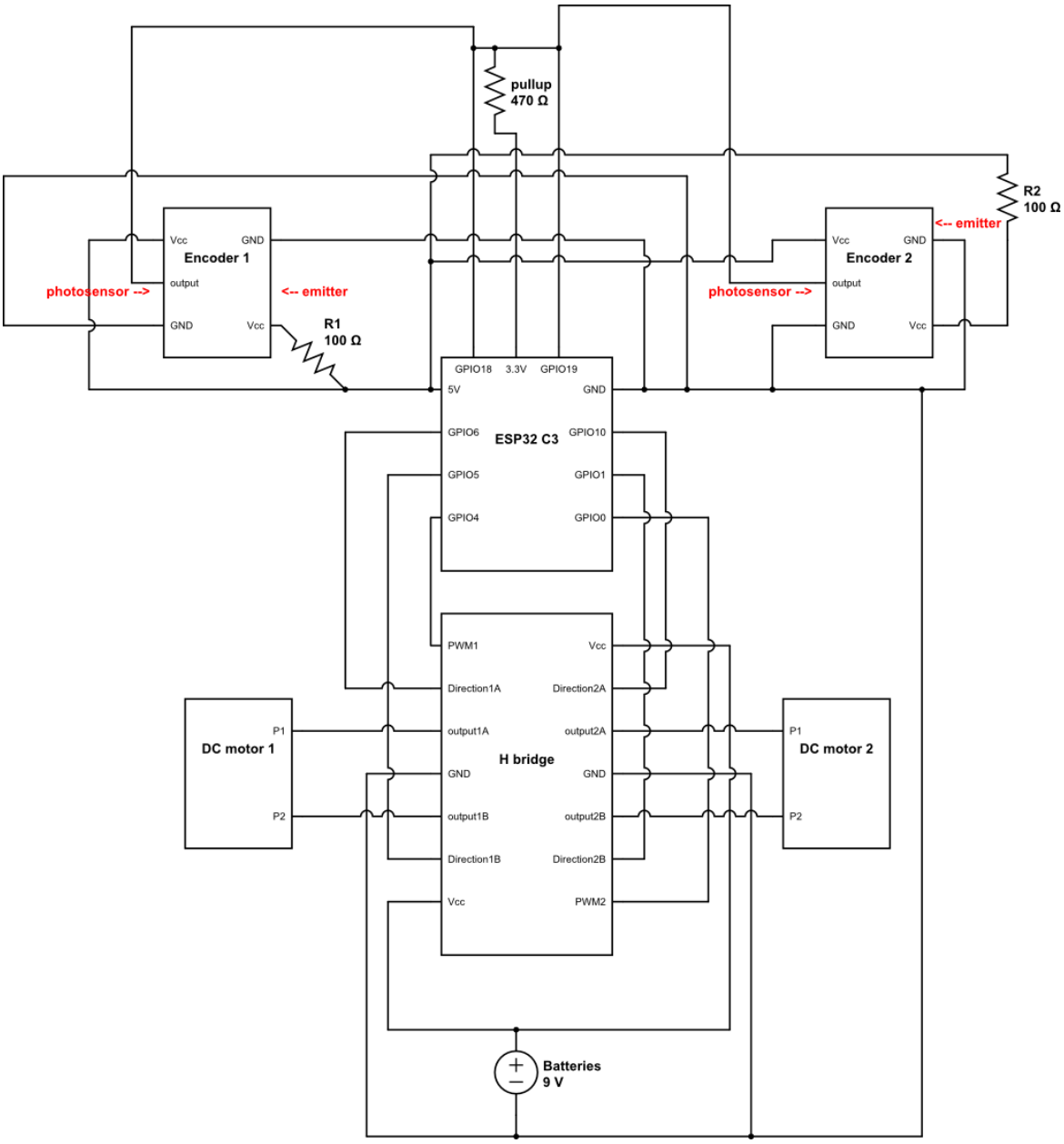
https://www.amazon.com/Professional-Mecanum-Chassis-Raspberry-Omnidirectional/dp/B09KL7NRBK/ref=sr_1_14_sspa?crid=379QTL5PM00S2&keywords=mecanum%2Bwheels&qid=1700077627&sprefix=mecanum%2Bwheels%2B%2Caps%2C79&sr=8-14-spons&sp_csd=d2lkZ2V0TmFtZT1zcF9tdGY&th=1

Caster Wheels:

https://www.amazon.com/Adhesive-Stainless-Universal-Rotation-Furniture/dp/B08YYQ6TGK/ref=sr_1_7_sspa?crid=1IH7SBGO74VQ8&keywords=small%2Bc

aster%2Bwheels%2B4&qid=1700078064&sprefix=small%2Bcaster%2Bwheels%2B4%2B%2Caps%2C141&sr=8-7-spons&sp_csd=d2lkZ2V0TmFtZT1zcF9tdGY&th=1

5. Circuit Diagram



6. Work Division:

We did everything together equally.