PIE Mini Project 3

Jonas Kazlauskas, Dianna Sims,
Jackie Zeng

October 21, 2021

1 Introduction

Mini Project 3 was focused on building a line-following robot, where we were given the opportunity to utilize and develop skills in multiple aspects of engineering. The main deliverables of this assignment were to

1. Integrate sensing and control with a two-wheeled, mechanical robotic platform
2. Write a closed-loop controller that runs on Arduino and enables the robot to follow a track consisting of electrical tape laid out on the floor in a loop
3. Tune the controller or simply change the performance of the robot (without recompiling and reloading code) to enable the robot to complete a lap around the course as quickly as possible

2 Materials

- Arduino
- 3X IR reflectance sensor
- 1X Adafruit v2.3 Motor Shield
- 1X 12V power supply
- 1X male barrel jack to screw terminal connector
- 6X Two pin Molex wire-to-wire connector
- 12X genderless crimp terminals (for crimping to wires and inserting in the Molex connector)
- Some 24AWG stranded, silicone-insulated wire
- 2X gearmotor and wheel
- 1X acrylic chassis platform with caster

3 Procedure

3.1 Calibration

We began by calibrating the IR reflectance sensors to be able to distinguish the tile floor from the black tape line our robot will be following. To do this, we found the maximum current for the sensor

collector using the datasheet, then calculated the smallest resistor value we can safely use in series. The calculation is shown below.

$$V = I * R$$
$$5v = 0.1A * R$$
$$R = 50\Omega$$

After calculating this value we could safely experiment with any resistor in series above 50Ω. Next, we setup a small line of tape on the tile floor and wrote a simple script to display the current sensor reading. Using this script, we monitored the sensor value on and off the tape while increasing the resistor value used in series with the collector. Our goal was to use a resistor value that gave us a clear distinction between the sensor on and off the tape. We also adjusted the height of the sensor to get a sense of where we would like it to be mounted on the robot. After some experimentation we arrived at a resistor value of 50kΩ and a height of roughly 1/2 cm. This gave us readings of around 40 while over the tape and 350 while on the tile.

3.2 Schematic

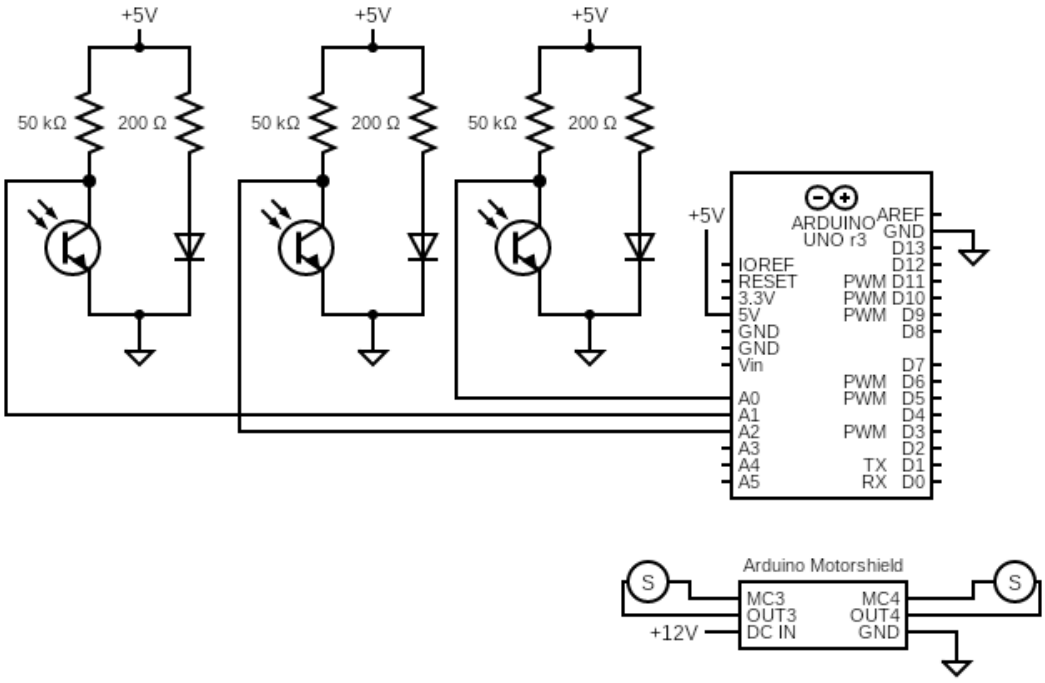

Figure 1: The upper schematic shows how the IR sensors are connected to the Arduino. The lower section shows how the wheel motors connect to the Arduino Motor Shield.

3.3 Mechanical Design

After determining the ideal sensor height, we worked on designing our sensor mount. The mechanical mount structure consists of a single 3D-printed part to hold the sensors in place along with the wires. The part is mounted to the car chassis, which we did not modify, and the chassis allows us to integrate all of the mechanical and electrical components together in a stable manner.

We used the same 3D-printed design to mount each of the three IR sensors the same way. There is a base for a single sensor and a support for the wires in each sensor mount. Each part is mounted via a screw and nut through holes on the chassis highlighted below in Figure 2.

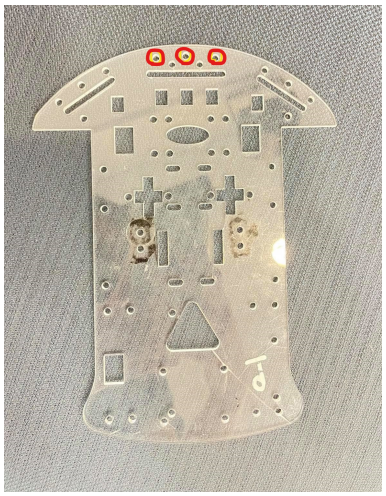

Figure 2: Chassis image with mount holes annotated

Our final line-following vehicle has 3 IR sensors which is shown in Figure 3 below.

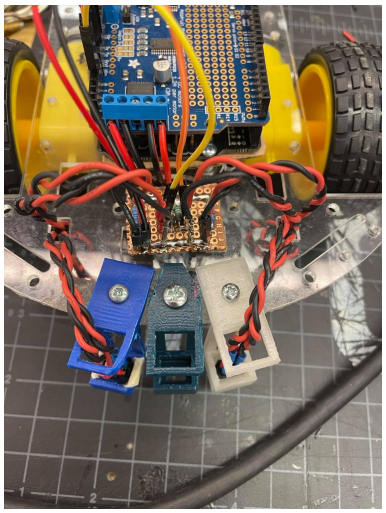

Figure 3: Close up image of three mounts with sensors (middle sensor removed for clarity and visibility)

Each part was designed such that the mounting face had a high surface area to improve mounting stability. It is important to note that the single mounting screw allows the mount to rotate. This helped us fine tune the horizontal positioning of each sensor. We also adjusted the sensor height by adding a nut in between the mounting surface and the chassis after initial testing. This design also allowed us to reduce the sensor height from the ground by mounting it below the chassis if needed.

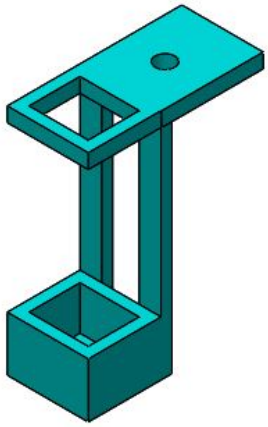

Figure 4: CAD of the sensor-chassis mount

3.4 Controls

The goal of our control code was to adjust motor speeds based on sensor readings to navigate the robot along black tape. We began by writing the simplest version of this controller using the "bang bang" technique. Writing this allowed us to get familiar with interacting with the motor controller library as well as reading and processing sensor readings. This approach resulted in jerky motion of the robot. After writing this version we began researching PID controls. After writing a basic PID control system we decided to add a third sensor to aid with I. This final version is described below.

In our PID control, we define error as the difference in our right and left sensors, with the value of the middle sensor added or subtracted to match the current sign of the error. Middle sensor value is 0 unless it is below a given threshold of 100 (unless it is off the tape). Error offset is set to 25 and is included to account for inconsistencies in the sensor outputs.

```
int getMidSensorReading() {
    int val = getReadings(SENSORPIN3);
    if (val>middleSensorThresh){
        return 0;
    }
    return val;
}

void setErrorDirection(int outsideError) {
    if (outsideError >= errorOffset) {
        errorDirection = 1;
    }
    else if (outsideError <= -1*errorOffset) {
```

```
        errorDirection = -1;
    }
}

int getError() {
    int outsideError = getReadings(SENSORPIN1) - getReadings(SENSORPIN2);
    setErrorDirection(outsideError);
    int error = (abs(outsideError) + getMidSensorReading()) * errorDirection;
    if (abs(error)<errorOffset) {
        return 0;
    }
    return error;
}
```

If our robot drives off on the right side of the tape, the error will increase quickly to a positive value and hold a positive error as long as the center sensor is not on the tape. Conversely, if our robot drives off on the left side a negative error will be returned.

We use this definition of error to adjust the speed of each wheel on our robot. Below is the function for calculating PID as well as adjusting motor speed. P is defined as the value of our error. P is responsible for most of our control, and has the highest coefficient (kP). I is defined as the summation of our error over time. I helps to increasingly turn the car back onto the tape when it has veered off significantly, and has the smallest coefficient kI because it increases quickly. D is defined as the difference between the last error and the current error. D helps to turn the car quickly when a tight turn is encountered. Each of these values are multiplied by their coefficients and summed, then added to the right motor speed and subtracted from the left motor speed to control the robot.

```
void updatePID(int error) {
    P = error;
    I += error;
    D = error-lastError;
    lastError=error;
}

void updateMotorSpeed() {
    pidSpeed = kP*P + kI*I + kD*D;
    pidSpeed = constrain(pidSpeed, -255+motorSpeed, 255-motorSpeed);
    motorRight->setSpeed(motorSpeed + pidSpeed);
    motorLeft->setSpeed(motorSpeed - pidSpeed);
}
```

We tuned our PID coefficients by experimentation. We started with only P, and adjusted this value until the car drove consistently. Next we added D to aid with tight turns, and lastly added I. Each coefficient began at a very small value (0.00001) and was slowly increased. The final values for the coefficients are shown below.

- kP: 0.07
- kI: 0.000001
- kD: 0.03

3.5 Serial Parameter Updating

To help in tuning various parameters we added live serial parameter updating. This was written in python and works by sending a key and a value separated by a ":" character. This message is parsed by the arduino, and a corresponding array is updated. This array then updates all the relevant constants that are used throughout the code.

```
CONST_MAP = {
    "speed": 0,
    "error_offset": 1,
    "kP": 2,
    "kI": 3,
    "kD": 4,
}

def send_updates(data):
    ser = serial.Serial(PORT, BAUD)
    for k, v in data.items():
        if v:
            message = f"{CONST_MAP[k]}:{v},"
            ser.write(message.encode('utf-8'))
    ser.close()
```

```
void setConsts() {
    motorSpeed = consts[0];
    errorOffset = consts[1];
    kP = consts[2];
    kI = consts[3];
    kD = consts[4];
}

void updateFromSerial() {
    if (Serial.available() > 3) {
        int key = Serial.parseInt();
        float value = Serial.parseFloat();
        consts[key] = value;
    }
    setConsts();
}
```

The Python code that sends messages is also wrapped in a flask web application that allows us to fill out a webform to update parameters. The form is shown below and the code that provides this functionality is included in the appendix.

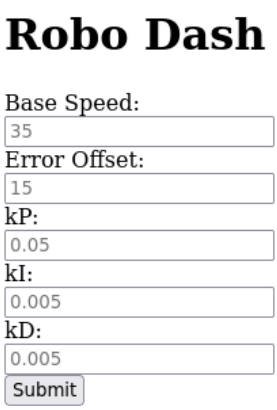

Figure 5: Dashboard to update parameters without recompiling the arduino

3.6 Video Completing the Course

`https://www.youtube.com/watch?v=fA2t12I_eR0`

3.7 Plot of Controls

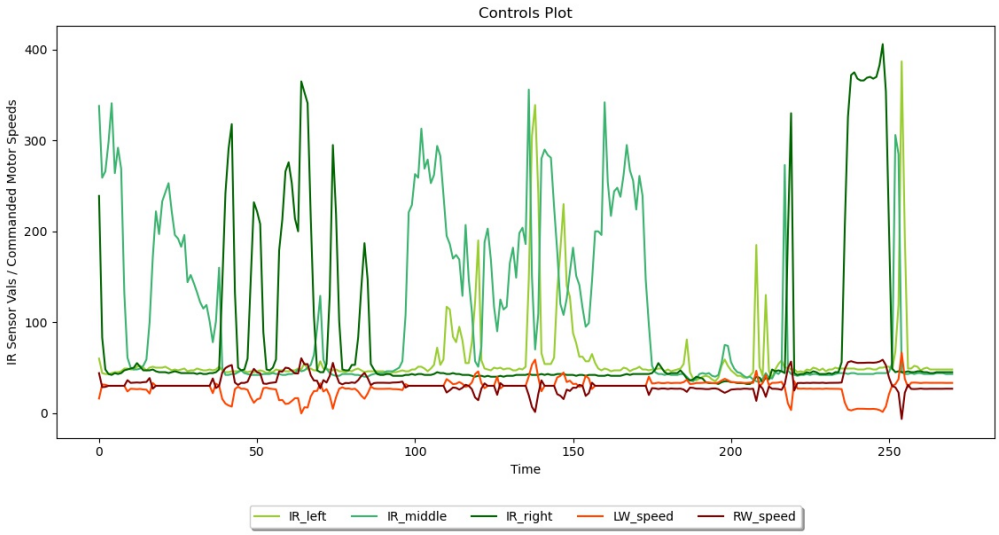

Figure 6: Plot superimposing sensor data from IR sensors and commanded motor speed for a short trial run.

From this figure, we can see that the wheels spin at the base speed of 30 when the middle IR sensor detect the black tape (high IR values). This is the expected behavior since we want to continue straight when the robot is on the line. When the robot veers off course, shown by the spikes in IR_left and IR_right, we adjust the wheels speed based on the direction we're off course. We adjusted both wheel speeds by the same value in opposite directions to have achieve smoother control.

4 Mechanical Design Issues

We experienced some issues early on with the mechanical design of the sensor mounts for this project. Our first iteration was an attempt at an adjustable mount, that would allow for easy addition of sensors and ease of adjustability in the height off the ground and horizontal position about the black tape line. We decided to instead simplify the mounting design to hold one sensor only, but focus on its stability, making sure it was secured to the chassis, would not move in place, and would not cause the wires to have any pull or strain on them. Knowing that it is being designed with a single support for each one, we used the degree of freedom to allow for adjustability about the horizontal position of the sensors, and we were able to adjust the height anyway by choosing to mount the surface from above or below the chassis and offsetting it with nuts. In addition, we learned that 3D printing such a small part with small holes or openings can be inconsistent, and it can be difficult to remove the support material.This required slowing down the print and adjusting the settings to be on the more careful side to ensure that the print was successful. We also encountered issues with the print "sticking" to the print bed, but this was mostly resolved when we added a brim to the base of the part.

5 Reflection

Overall, learning to design our line-following vehicle to travel as fast as possible in this project went well. Through this project we were able to explore how a sensor could be used to control the performance of our vehicle, depending on the conditions it was experiencing. We did not have an ECE on the team, so there was a learning curve in regards to optimizing the space to place all of the electrical components in an an organized manner. Along the way, there was a lot of debugging and troubleshooting involved to figure out why the vehicle was not traveling along the path as expected, especially after integrating our third, middle IR sensor. One of the main problems we faced was that the stable positioning of the height of the sensor mattered, keeping in consideration the conical shape of the sensors. So, we experimented with different heights until we were able to get useful and consistent sensor value readings. On the software side of things, we learned to optimize our PID controller such that our vehicle could complete the path quickly, which consisted of including serial data and troubleshooting our code until our positive (+) and negative (-) signs were correct. This issue became more prominent when we integrated our third sensor. From a nontechnical standpoint, we learned how to work asynchronously on our individual tasks while avoiding inhibiting progress towards the project.

6 Appendix

Link to repository containing full source code:

`https://github.com/jonaskaz/Line-Robot`

6.1 Arduino Controls Code

```
#include <ArduinoJson.h>
#include <Adafruit_MotorShield.h>
#include <SPI.h>


#define SENSORPIN1 A0 // Left
#define SENSORPIN2 A1 // Right
#define SENSORPIN3 A2 // Middle
#define NUMREADINGS 3 // Number of readings to average
```

```
10
11  float motorSpeed = 30;
12  float errorOffset = 30;
13  float middleSensorThresh = 100;
14  int errorDirection = 1;
15  float kP = 0.07;
16  float kI = 0.000001;
17  float kD = 0.03;
18
19  float consts[] = {motorSpeed, errorOffset, kP, kI, kD};
20
21  float P = 0;
22  float I = 0;
23  float D = 0;
24  float lastError = 0;
25  float pidSpeed = 0;
26
27
28  Adafruit_MotorShield AFMS = Adafruit_MotorShield();
29
30  Adafruit_DCMotor *motorRight = AFMS.getMotor(3);
31  Adafruit_DCMotor *motorLeft = AFMS.getMotor(4);
32
33
34  void driveForward() {
35      motorRight->setSpeed(motorSpeed);
36      motorLeft->setSpeed(motorSpeed);
37      motorRight->run(BACKWARD);
38      motorLeft->run(FORWARD);
39  }
40
41  void setup() {
42      Serial.begin(9600);
43      AFMS.begin();
44      driveForward();
45  }
46
47  int getReadings(int sensorPin) {
48      int senseTotal = 0;
49      for (int i=0; i < NUMREADINGS; i++) {
50          senseTotal+= analogRead(sensorPin);
51      }
52      return int(senseTotal/NUMREADINGS);
53  }
54
55  int getMidSensorReading() {
56      int val = getReadings(SENSORPIN3);
57      if (val>middleSensorThresh){
58          return 0;
59      }
60      return val;
```

```
}

void setErrorDirection(int outsideError) {
    if (outsideError >= errorOffset) {
        errorDirection = 1;
    }
    else if (outsideError <= -1*errorOffset) {
        errorDirection = -1;
    }
}

int getError() {
    int outsideError = getReadings(SENSORPIN1) - getReadings(SENSORPIN2);
    setErrorDirection(outsideError);
    int error = (abs(outsideError) + getMidSensorReading()) * errorDirection;
    if (abs(error)<errorOffset) {
        return 0;
    }
    return error;
}

void updatePID(int error) {
    P = error;
    I += error;
    D = error-lastError;
    lastError=error;
}

void updateMotorSpeed() {
    pidSpeed = kP*P + kI*I + kD*D;
    pidSpeed = constrain(pidSpeed, -200+motorSpeed, 200-motorSpeed);
    motorRight->setSpeed(motorSpeed + pidSpeed);
    motorLeft->setSpeed(motorSpeed - pidSpeed);
}

void setConsts() {
    motorSpeed = consts[0];
    errorOffset = consts[1];
    kP = consts[2];
    kI = consts[3];
    kD = consts[4];
}

void updateFromSerial() {
    if (Serial.available() > 3) {
        int key = Serial.parseInt();
        float value = Serial.parseFloat();
        consts[key] = value;
    }
    setConsts();
}
```

```
112
113 void loop() {
114     updatePID(getError());
115     updateMotorSpeed();
116     updateFromSerial();
117 }
```

6.2 Serial Updating Python Code

```
1  from flask import Flask, render_template, request
2  import serial
3
4
5  app = Flask(__name__)
6  PORT = '/dev/ttyACM0'
7  BAUD = 9600
8
9  CONST_MAP = {
10     "speed": 0,
11     "error_offset": 1,
12     "kP": 2,
13     "kI": 3,
14     "kD": 4,
15 }
16
17
18 @app.route('/', methods = ["POST", "GET"])
19 def home():
20     if request.method == 'POST':
21         data = request.form
22         send_updates(data)
23     return render_template("dash.html")
24
25
26 def send_updates(data):
27     ser = serial.Serial(PORT, BAUD)
28     for k, v in data.items():
29         if v:
30             message = f"{CONST_MAP[k]}:{v},"
31             ser.write(message.encode('utf-8'))
32     ser.close()
33
34
35 if __name__ == '__main__':
36     app.run()
```

6.3 HTML Web Form dash.html

```
1  <!DOCTYPE html>
2  <html>
3  <body>
```

```
<h1>Robo Dash</h1>
<form action = "/" method = "POST" enctype='multipart/form-data'>
    <label for="speed">Base Speed:</label><br>
    <input type="text" id="speed" name="speed" placeholder="35"><br>
    <label for="error_offset">Error Offset:</label><br>
    <input type="text" id="error_offset" name="error_offset" placeholder="15"><br>
    <label for="kP">kP:</label><br>
    <input type="text" id="kP" name="kP" placeholder="0.05"><br>
    <label for="kI">kI:</label><br>
    <input type="text" id="kI" name="kI" placeholder="0.005" ><br>
    <label for="kD">kD:</label><br>
    <input type="text" id="kD" name="kD" placeholder="0.005" ><br>
    <button type="submit" id="sub" name="sub">Submit</button><br>
</form>
</div>
</body>
</html>
```