Mini-Project 3: DC Motor Control

Melody Chiu, Simrun Mutha, Maya Lum
Principles of Integrated Engineering Fall 2021: Section 3

Overview

In this project, we created a robot that uses infrared sensing and motor speed control to follow a line of electrical tape on the floor. We made the robot out of two DC motors, a pre-designed chassis, infrared sensors, an Arduino Uno, and a motor shield for the Arduino. Because the electrical tape is more matte than the floor and thus less reflective, the robot can use the infrared sensors to tell whether it's on the line, and by using two sensors, it can determine whether it's to the left or right of the line. Once the robot determines which direction it needs to go to follow the line, it can turn in that direction by spinning one motor more quickly than the other.

When creating the line following robot, we first designed a circuit using voltage dividers to send data from the infrared sensors to an Arduino Uno, and tested the sensors at various heights off of the floor to find the ideal sensor position. Next, we assembled the chassis kit and created a mount for the IR sensors so they would be at the correct height above the floor. After that, we got the motors running, making sure the two motors ran at the same speed and adjusting the motor mounts so the mounting bolts wouldn't interfere with the wheels. We then wrote a control loop algorithm that would detect whether the IR sensors were on the line of tape based on whether the voltage values met a certain threshold, and change the motor speeds accordingly. The first working version of the control loop used proportional controls to determine how quickly to spin the motors when the robot was turning, but we also explored using PID control for the robot as one of our stretch goals. Finally, we implemented a graphical user interface that allowed us to make changes to the robot's behavior using Serial inputs.

Methods

Materials

- Arduino Uno
- USB cable
- 2X IR reflectance sensor
- 1X Adafruit v2.3 Motor Shield
- 1X 12V power supply
- 1X male barrel jack to screw terminal connector
- 1X Four pin Molex wire-to-wire connector
- 4X genderless crimp terminals
- 24AWG stranded, silicone-insulated wire
- 2X gear motor and wheel
- 1X acrylic chassis platform with caster wheel
- 2X Aluminum motor mount
- Custom IR Sensor Mount
- Custom Caster wheel spacer

IR Sensor Calibration

For our team, the main goal for calibrating the sensors involved finding a threshold value which could be used to determine whether the sensor was on the tape or just on the floor. We tried out different resistor values and different distances from the ground to find a setup that would give us the highest difference between the tape and floor. We tried out 50Ω, 1KΩ, 20KΩ and 100KΩ. We found that the 100KΩ resistor resulted in the most drastic difference. We also tried out different distances from the floor from 0.5 inches to 4 inches. We realised that the sensors work better closer to the ground and settled on trying to keep them at least 0.7 from the ground. As we only needed to detect either tape or not, we didn't go through the process of calibrating the sensors at many different distances as we only needed the optimal case for this robot. Because we knew that we needed to run the robot on the floor for this project, we did all our testing on the floor so the values would be consistent.

Circuit Diagram

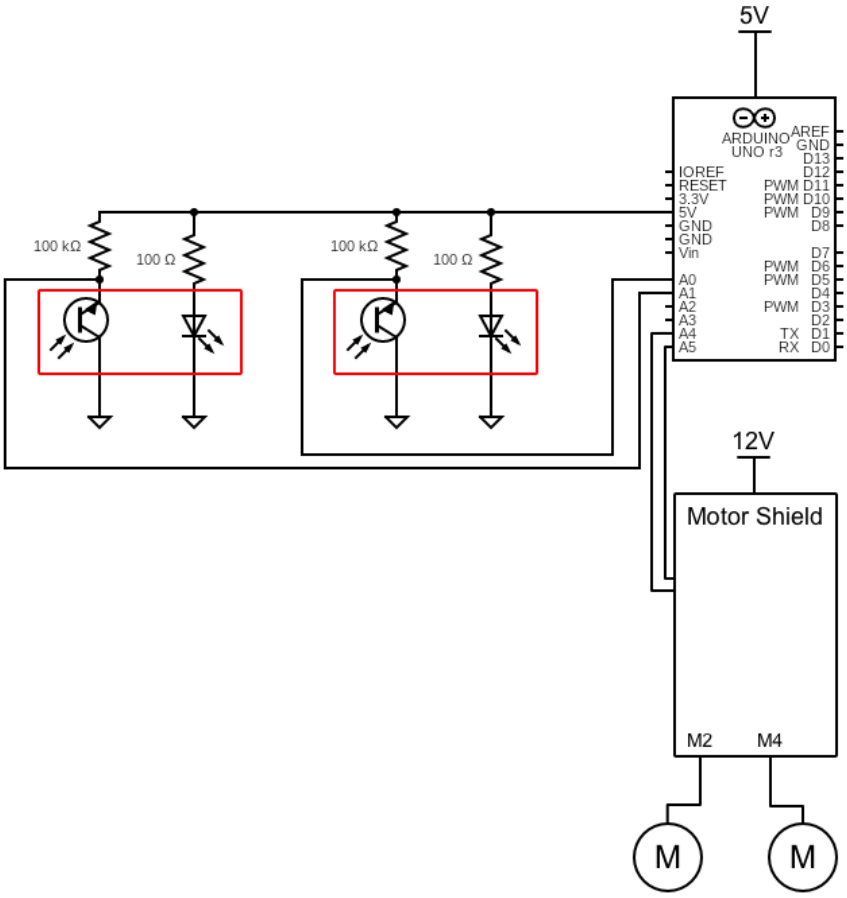

Figure 1. Diagram showing the circuit we used for our robot. The red boxes on the diagram indicate the sensor units, which contain both a phototransistor and an LED. The LEDs emit infrared light.

Mechanical Integration

When working on mechanical integration of the robot, we prioritized simplicity and ease of implementation so we could have more time to test out the control loop. As a result, we used the default chassis kit provided and attached the electronics (breadboard, Arduino Uno, and motor shield) using simple fasteners such as zip ties and masking tape. We slightly modified the chassis by adding a laser cut spacer to the caster wheel, as the original wheel positions resulted in the chassis being slightly tilted.

We also wanted to add an IR sensor mount to the front of the chassis to keep them in the optimal height range for good sensor performance. We designed a part to hold four sensors that are each spaced 0.71” apart (since we measured the tape to be 0.73”) and centered around the chassis’s axis of symmetry. The sensors attached to the mount using the snap pins on the sides, and the mount could be bolted into three holes on the chassis. The mount was designed for the tip of the LED and phototransistors to be 0.4” off the ground, as we measured during the initial sensor calibration, but after re-attaching the mounted sensors to the circuit and testing the code, we found that the sensors were too close to the ground to get meaningful IR data. We re-tested the sensors at increased heights, found that they needed to be raised 0.5”, and modified the mount accordingly. However, during our final tests and line following attempts, we realized that the tape on the actual path was thinner than the tape we measured when designing the mounts: the tape we designed the mount for was 0.73” wide, so our sensors were spaced 0.71” apart, but the actual tape was 0.70” wide. As a result, our control loop would not work when we used the mount to position the sensors, so we removed the mount from the robot and put the sensors back on the breadboard. With the final mechanical integration system shown below in Figure 2, our robot was able to complete a loop of the track.

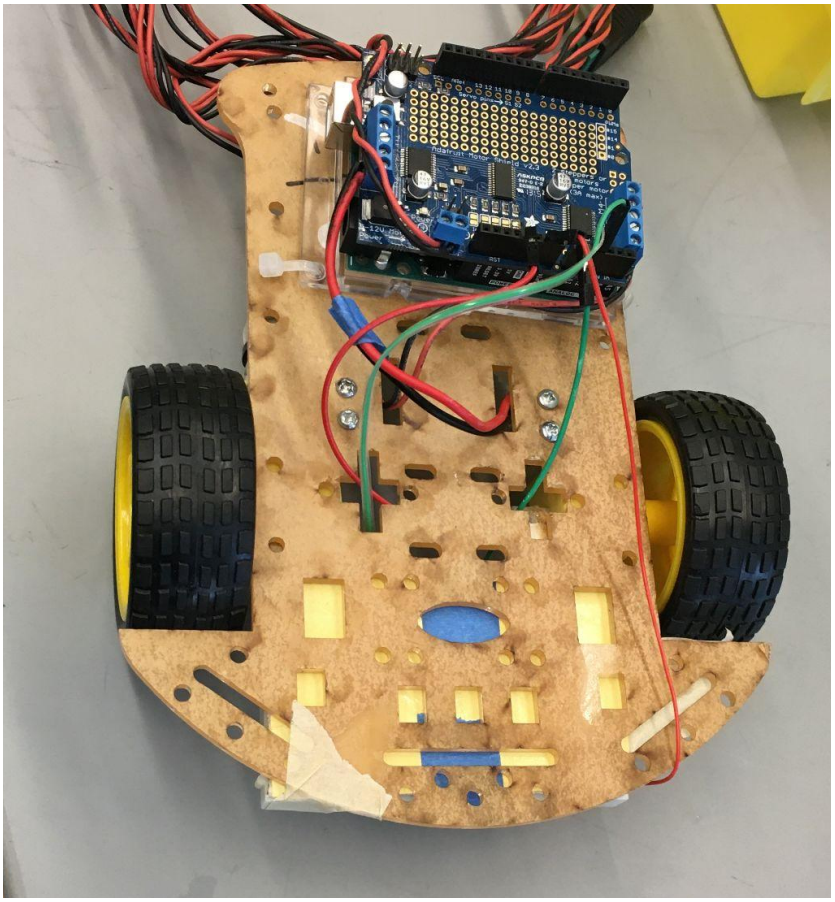

Figure 2. Photo showing the top view of our robot and our system for attaching electrical components to the chassis. As shown, the breadboard has been taped onto the underside of the chassis, and the Arduino and motor shield have been zip tied to holes in the chassis.

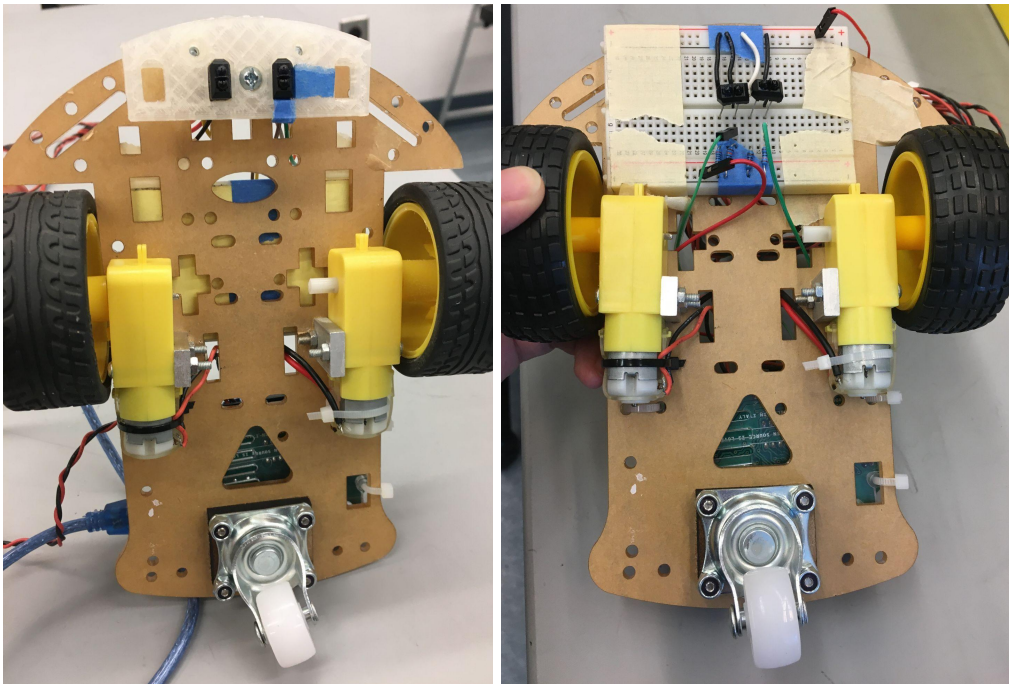

Figure 3. Two photos showing the bottom of our robot with our systems for integrating the motors and caster wheels, while the IR sensor mount was still attached (left), and in its final configuration (right). These components were all bolted to different holes in the chassis. The caster wheel spacer is also visible. As mentioned in the main paragraphs on mechanical integration, the IR sensor mount was removed from the final design because it spaced the sensors too far apart.

Controller

From testing our sensors, we found a threshold value that could determine whether each sensor was on the tape or off the tape. We started out by defining some basic functions for telling the car to go straight, turn left and turn right. When both sensors are on the tape, we tell the car to go straight, when the right sensor is on the tape but the left is off, we tell the robot to turn right and vice versa for turning left. These functions are shown in the code below.

```
53  void goStraight(){
54    // go straight
55    leftMotor->run(BACKWARD);
56    rightMotor-> run(FORWARD);
57    leftMotor->setSpeed(fastSpeed);
58    rightMotor->setSpeed(fastSpeed);
59  }
```

Figure 4. Straight motor control function

In our left turn function, we command the left wheel to rotate at a set speed, while the right wheel stays still. The right turn function follows the same logic, commanding the right wheel to rotate at set speed, while the left wheel stays still. We implemented these functions to take in a parameter that specifies the speed the wheel should rotate at. This is so that we can control how steeply the car will turn, which we will go over in more detail below.

```
79  void leftTurn(int leftDeg){
80    // turn left with the steepness controlled by the input leftDeg
81    rightMotor-> run(BACKWARD);
82    rightMotor->setSpeed(leftDeg);
83  }
```

Figure 5. Left turn motor control function

```
93  void rightTurn(int rightDeg){
94    // turn left with the steepness controlled by the input leftDeg
95    leftMotor-> run(FORWARD);
96    leftMotor->setSpeed(rightDeg);
97  }
```

Figure 6. Right turn motor control function

In our void loop, we get sensor readings from the two IR sensors and call on the appropriate motor control function based on those values. If the reading indicates that the right sensor is sensing the tape, while the left sensor is not, that means the car is steering to the left of the line and should turn slightly right to adjust itself back on track. If both sensors are on the tape, that means the car is on track and should drive forward. If the left sensor is on the tape, while the right is not, the car should adjust by turning slightly left. This logic alone works for a track with slightly curving. However, to make sure our car can make the steeper turns, we adjust the variables leftDeg and rightDef with each loop, to increase the steepness of the turns. The variables leftTurn and rightTurn are initialized at zero and increased by 1 each time the sensors indicate that the car needs to turn in their direction, setting the other direction back to zero. We also have a counter variable that is updated each time the loop function runs. We use the variable modValue to control the rate at which the turn speeds are increased. The variable modValue is the number of times we enter the left turn state until the leftDeg is increased, similarly with the right turn state and rightDeg.

```
129       loopCount += 1;
130       // LEFT TURN: Left sensor on tape, right sensor not
131       if (sensorLeft >= thresholdTape && sensorRight <= thresholdTape){
132         Serial.println("Left");
133         rightDeg = 0; // making the right wheel stop
134         // if the robot continues to enter this statement, then increase the sharpness of the left turn
135         if (loop_count % mod_value == 0){
136           leftDeg += 1;
137         }
138         leftTurn(left_deg);
139       }
140       // STRAIGHT
141       // both sensors are on the tape
142       else if (sensorLeft >= thresholdTape && sensorRight >= thresholdTape){
143         Serial.println("going straight");
144         rightDeg = 0;
145         leftDeg = 0;
146         goStraight();
147        }
148        // RIGHT TURN: Right sensor on tape, left sensor not
149        else if (sensorRight >= thresholdTape && sensorLeft <= thresholdTape){
150         Serial.println("Right");
151         leftDeg = 0; // making the left wheel stop
152          // if the robot continues to enter this statement, then increase the sharpness of the right turn
153         if (loop_count % mod_value == 0){
154            rightDeg += 1;
155          }
156         rightTurn(rightDeg);
157        }
158     }
```

Figure 7. Control loop

To illustrate how the logic of our program works, we can think about the scenario where the car is driving towards a steep left turn. As it approaches the curve, both sensors are on the tape so the car drives forward along the line. Once the line starts curving, the right sensor is no longer on the tape so we enter the left turn state of the loop function. The variable leftDeg is updated and the leftTurn function is called with leftDeg as the parameter. Since the curve is steep, the car remains off the line and the right IR sensor remains off the tape even after turning left slightly. Now, we enter the left turn state of the loop function again, updating leftDeg again. When the leftTurn function is called with the updated leftDeg, the car turns faster. Eventually, the car turns enough that it's back on track and continues down the line.

Next, we programmed a GUI in Python using the Tkinter library. From this interface, we can command the car to start and stop driving, set the modValue, and quit the GUI program. We begin by initializing and setting the properties of the GUI window. Next we define the properties of our widgets: the three control buttons, the modValue label and the modValue text input. Then, we specify the positioning of the widgets on the GUI window.

```
42      # Defining all the widgets
43      start_b = tk.Button(window, text = "START", command = starting, bg= "pale green", fg="dark slate gray", font=("Lucida Console", 20))
44      stop_b = tk.Button(window, text = "STOP", command = stopping, bg= "tomato", fg="dark slate gray", font=("Lucida Console", 20))
45      quit_b = tk.Button(window, text = "QUIT", command = quit, bg= "orchid", fg="dark slate gray", font=("Lucida Console", 20))
46      mod_label = tk.Label(window, text='Mod value:', font=("Lucida Console", 13))
47      mod_value = tk.Text(window, height = 1, width = 10)
48      send_b = tk.Button(window, text="Send", command = set_mod_val, bg= "gray", fg="dark slate gray", font=("Lucida Console", 13))
49
50      # Positioning all the buttons
51      start_b.grid(row=1, column=0, padx=5, pady=10)
52      stop_b.grid(row=1, column=1, padx=5, pady=10)
53      quit_b.grid(row=1, column=2, padx=5, pady=10)
54      mod_label.grid(row=2, column=0, padx=5)
55      mod_value.grid(row=2, column=1)
56      send_b.grid(row=2, column=2)
57
58      window.mainloop()
```

Figure 8. Section of Python GUI program that creates and positions widgets

Each of these widgets are linked to a command function that prints to the command line and sends a specific character to the Arduino to indicate the desired action. In the quit function, the serial connection to the Arduino and Tkinter window are destroyed. In the set_mod_val function, a character is sent to the Arduino followed by the mod value that the user input.

```
19      def starting():
20          # Sending an R to start the motor
21          print("Start driving")
22          megaBoard.write(b'R')
23
24      def stopping():
25          # Sending an S to stop the motor
26          print("Stop driving")
27          megaBoard.write(b'S')
28
29      def quit():
30          # Sending a Q to quit the program
31          print("\n** END OF PROGRAM**")
32          megaBoard.write(b'Q')
33          megaBoard.close()
34          window.destroy()
35
36      def set_mod_val():
37          # Sending a mod value to the Arduino
38          print("Set mod value")
39          megaBoard.write(b'M')
40          megaBoard.write(mod_val)
```

Figure 9. Section of Python GUI program that defines command functions

On the Arduino's side, at each control loop, we read a character from the serial monitor to determine the state of the car and see whether the user wants to change the mod value. As shown in Figure 10, in the case that the received character is R, then the isReady variable is set to 1 which indicates that the car should actively run. If it is S, then that is switched to 0 and the car should stop. In the case that the character is M, the recvWithEndMarker function (shown in Figure 11) should be called which reads the user input value and updates modValue based on that input.

```
109     recvOneChar();
110     // if R is received from the serial monitor, then the robot will run
111     if (receivedChar == 'R'){
112       isReady = 1;
113     }
114     // if S is received from the serial monitor, then the robot will stop
115     if (receivedChar == 'S'){
116       isReady = 0;
117       stopMotors();
118     }
119     // if M is received from the serial monitor, then read the input mod value
120     if (receivedChar == 'M'){
121       recvWithEndMarker();
122       modValue = receivedModValue;
123     }
```

Figure 10. Section of Arduino program that shows serial communication portion of loop function

Other functions that help serial communication are recvOneChar which receive a single character from the serial monitor. The function showNewNumber receives the characters that the user inputs as the new mod value and converts that array of chars into an integer. The function showNewData checks whether new data has been received from the serial monitor.

```
34  void recvOneChar() {
35    // Receiving a character from serial
36      if (Serial.available() > 0) {
37          receivedChar = Serial.read();
38          newData = true;
39      }
40  }
41
42  void recvWithEndMarker() {
43      static byte ndx = 0;
44      char endMarker = '\n';
45      char rc;
46      if (Serial.available() > 0) {
47          rc = Serial.read();
48          if (rc != endMarker) {
49              receivedChars[ndx] = rc;
50              ndx++;
51              if (ndx >= numChars) {
52                  ndx = numChars - 1;
53              }
54          }
55          else {
56              receivedChars[ndx] = '\0'; // terminate the string
57              ndx = 0;
58              newData = true;
59          }
60      }
61  }
```

```
63  void showNewNumber() {
64      if (newData == true) {
65          receivedModValue = 0;
66          receivedModValue = atoi(receivedChars);
67          newData = false;
68      }
69  }
70
71  void showNewData() {
72    // Checking if new data has been received
73      if (newData == true) {
74          Serial.println(receivedChar);
75          newData = false;
76      }
77  }
```

Figure 11. Serial communication functions in Arduino program

Debugging

We ran into a lot of electrical issues over the course of this project. First, we had the IR sensor oriented incorrectly so that the LED inside the sensor was reversed. It took us a while to realize that the issue was with the LED, through which current can only flow through one direction, embedded in the sensor.

Similarly, when we reconnected the IR sensors to the circuit after attaching them to the sensor mount, we went to test the sensor values and found that while one seemed to be working fine, returning analog values that would differ by over a hundred depending on what we were pointing it at, the other sensor always returned a number in the 990-1023 range. After checking that the working IR sensor and the broken IR sensor were wired the same way, we tried switching out many circuit components to fix this error. We started by replacing the IR sensor in case the components had gotten damaged, which didn't resolve the issue, then the jumper wires we used. Neither of those changes solved the error, so we tried switching the buggy circuit to a different port on the Arduino and switching the resistors between the two circuits. Finally, we tried moving the buggy circuit to a different row of the breadboard, and realized that the wire connecting the phototransistor to ground had broken. Replacing the wire fixed the issue.

Results

Input and Output Data

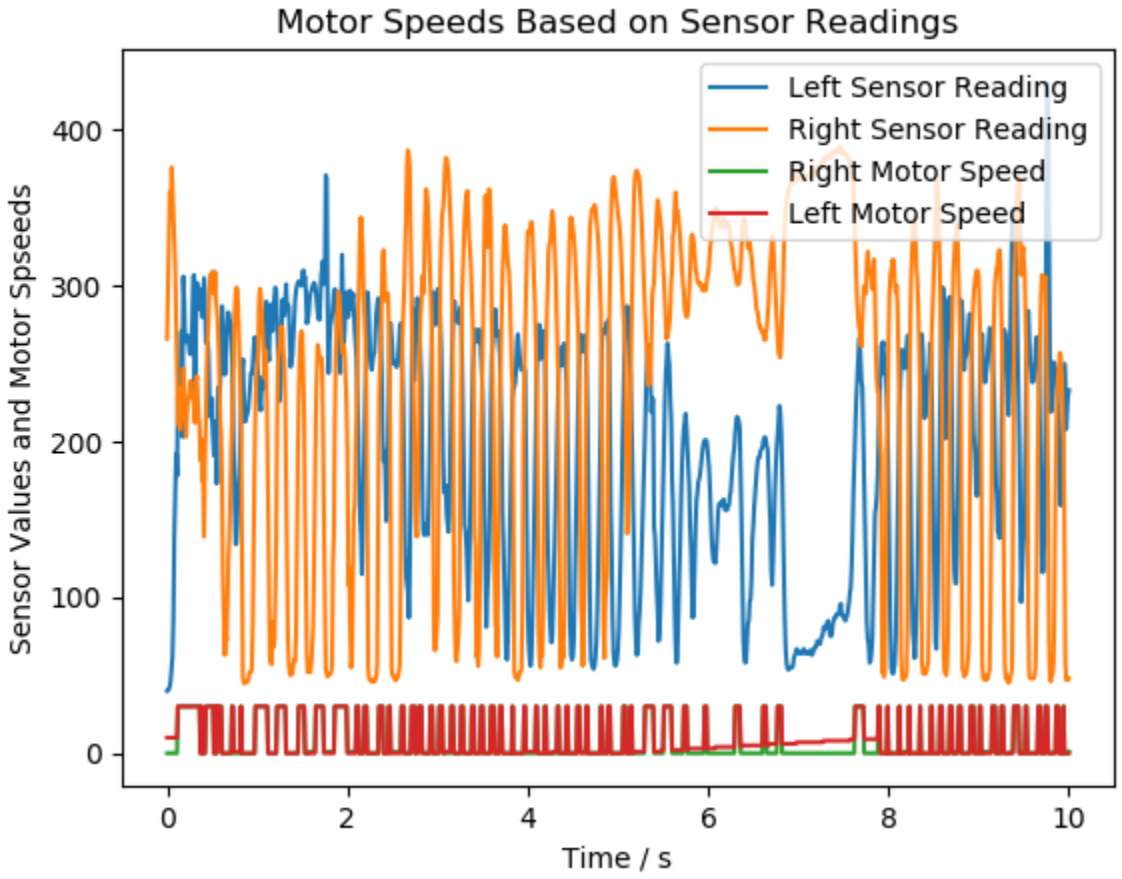

Figure . Plot superimposing sensor data from two IRs and commanded motor speeds

The figure above shows the data collected from the sensors and the motor speeds as the motor is making a turn. The robot is going straight for the first 6 seconds of the turn. At around 7 seconds, the left sensor value is much smaller than the right sensor value which means that only the right sensor is on the tape and the robot needs to turn right. In order to turn the motor right, the right motor stops and the left motor speeds up. The longer the robot needs to turn, the steeper the turn gets. This helps the robot go across steep turns.

Track Completion Video

Reflection

Melody

I feel very comfortable taking on the software portion of these PIE projects, so I contributed to this project in that way as usual. I also got to work with the electronics of the line-following robot which was very fun and useful to learn. I got to practice methodically testing out different resistors until we found a suitable one for our circuit. I've become very comfortable with the workflow of uploading a program onto the Arduino and looking at the Serial monitor to see what's happening. I enjoyed collaborating with my teammates on this project and felt like the software, electrical and mechanical integration was smoother. The way that we had written our program directly determined how the IR sensors should be placed relative to one another and the ground, which affected the physical mount that held them. For this project, I didn't have to wait for the mechanical portion of the project to be complete before I could start working on the code which made it a lot easier for our team to manage our time. Because a chassis was provided, we could simply tape the Arduino and the breadboard to the car and get started with testing out our code. Although our team achieved the goal of completing a run through of the track, we didn't implement a PID system in our controller. From reading the resources provided, I think I have an idea of how a general PID system works, but I'm not confident I know how to apply those concepts.

Simrun

I worked on software and electrical tasks in this project. At the very start of this project, I worked on looking through the IR datasheet and working on creating the circuit with it. I think I still struggle to understand datasheets and figure out what information is important so going through this process was a little challenging but ultimately helpful for me to learn. Once we had figured out a resistor value, I was very happy with how well the sensors were able to detect tape and how drastic the difference between the values was. The other big task I worked on was implementing the control loop. While I read through the PID resources, I did not actually implement it. I think I would need more time to implement that and I was confused by how to select a lot of the values necessary for that. As a team, we decided to start with MVP and then improve the control loop as necessary. I think that was a good idea because we were able to get a pretty good working first draft just by creating some simple conditions. I enjoyed the iterative process for figuring out how to improve the control loop. I had some issues with creating the GUI because I was not sure how to debug it when it was not working because I couldn't use the serial monitor. I relied on my teammates to help me with this and eventually I learned that it was because I was treating a char as a string which was causing problems. Overall, I think I was able to gain valuable electrical and software experience with the lab.

Maya

I was the team member who was most comfortable with mechanical work, so I did a lot of the mechanical integration and assembly. I wanted to do a minimum viable prototype of the mechanical systems, since they were mostly pre-made and fairly simple, so I could also help with the programming aspect of the project. I wanted to help with programming because I felt like I could contribute more than usual to the software side of this project, as I had some familiarity with line following from high school projects. Additionally, after not being able to help with the code for Mini-Project 2, I wanted to make sure I understood the basics of the code for the project. However, due to poor time management on my part and having to quarantine for some of the week, my teammates had already developed some of the code for the robot.

I instead worked on a mount for the IR sensors, since while the robot was already working with the sensors in the breadboard, it was hard to get the sensors to be centered relative to the robot. The IR sensor mount didn't end up helping much with getting the robot working, but designing it taught me some important lessons. Firstly, I should do more prototyping before 3D printing or laser cutting anything. I assumed that spacing the sensors 0.02" less than the width of the tape would be sufficient, but I should have checked that assumption by making a low-fidelity sensor mount out of cardboard and testing the heights and distances. This prototyping process would have also enabled me to catch the discrepancy with the tape widths. Secondly, I need to learn how much to tolerance 3D printed parts from the nominal dimensions so they fit together as intended. I designed a part with through holes so I could put bolts through them, and knew I had to increase the diameter to make sure the bolt could fit cleanly through the holes, but didn't increase the diameter enough and had to file them down anyway.

Appendix: Source Code