Miniproject 3 Report

James Jagielski, Anthony Costarelli, and Natsuki Sacks

October 2022

1 Introduction

For this mini project, we were tasked to create a line-following robot using DC motor control, an Arduino, and two infrared reflectance sensors. These reflectance sensors consist of an infrared light emitting diode and a phototransistor. It outputs a value closer to zero when it's on top of a lighter surface (more light is reflected) than a higher surface. We created a closed-loop control using the reflectance reported from the phototransistor, which communicated to the motors and changed the robot's direction as necessary.

2 Electrical Integration

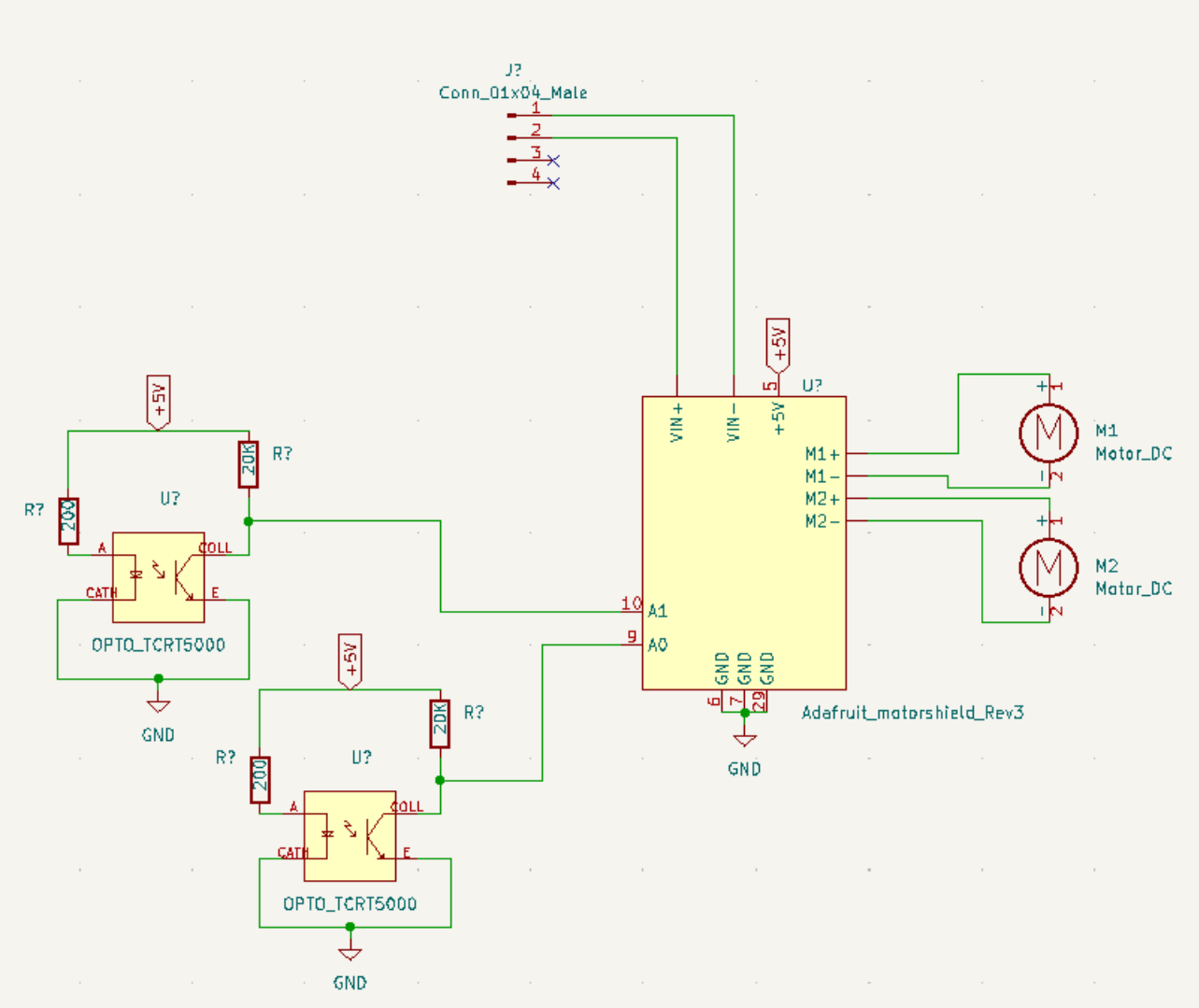

Figure 1: Circuit schematic for the robotic system.

2.1 Sensor Calibration

Our circuit was based on the diagram given to us for this project. We decided on a value of 20k for R_{sense}. We initially started with a resistance value of 5K, but the range of values that were being read in by the Arduino analog pins weren't large enough to be significant, so we increased the resistance value. We found a resistance value of 20K would give a good range of outputs and be sensitive enough. We used a guess and check method to find a good R_{sense} value.

We ran into issues with the spacing of the sensors being too wide. The robot was running into one corner where it couldn't pass through because when the robot would turn into the

corner the opposite sensor would sense the black line and it would get stuck going between the two sensors. We decided to move the sensors closer together, but also keep them as far apart as possible while still rounding the sharpest corner, which was about 80 degrees.

With a higher resistor value, the difference in analog readings between the tape and floor was much greater. Calibrating the sensors was now was very simple: we held the sensor close to the floor above the tape and again above the floor without the tape, examining the analog readings at both positions. Then, we created simple conditional statement from these numbers in our code as a means for checking if the sensor is looking at the tape or the floor.

3 Progression of Code

There were a number of things that we learned along the way as we created our Arduino code. Initially, we thought the sensor was going to return 0 or 1, so we checked if the black tape was being read with the digitalRead() function. After realizing, we changed the code to read from the sensor using analogRead(), which gave us the proper analog values (0 to 1023). With this information read to the Arduino, we created a threshold (see figure 4) to command the robot to move. This threshold was determined by checking values on the Serial Monitor to see where the change in sensor values was.

When it came to the actual driving, we originally would have the robot drive straight and then veer to the side when it would drift off the track. However, this made it so the robot couldn't turn fast enough for most turns, so we changed it so that the robot would quickly stop in place and then turn on the spot for as long as it needed. This turned out to be mostly robust except on turns ¿ 90 degrees. For these turns, due to the placement of the sensors, the robot would often find itself in a trap where it would keep wiggling left and right, without ever driving forward and completing the turn. We then changed to code so that the robot would maintain its velocity (whether that be forward or turning) for at least 100 ms before reading its sensors again and updating its velocity, which then allowed the robot to handle all turn angles because the robot wouldn't turn and then immediately turn back if the other sensor saw even a glimpse of the tape.

4 Mechanical Integration

The mechanical design of our robot was relatively simple. We put the robot together with the components we were provided, and decided to add one component for consistency in our data (Figure 2). To secure our breadboard to the chassis, we used a breadboard glued to cardboard. This prevented the breadboard from falling off while the robot was moving. Additionally, we used tape to secure the Arduino to the chassis.

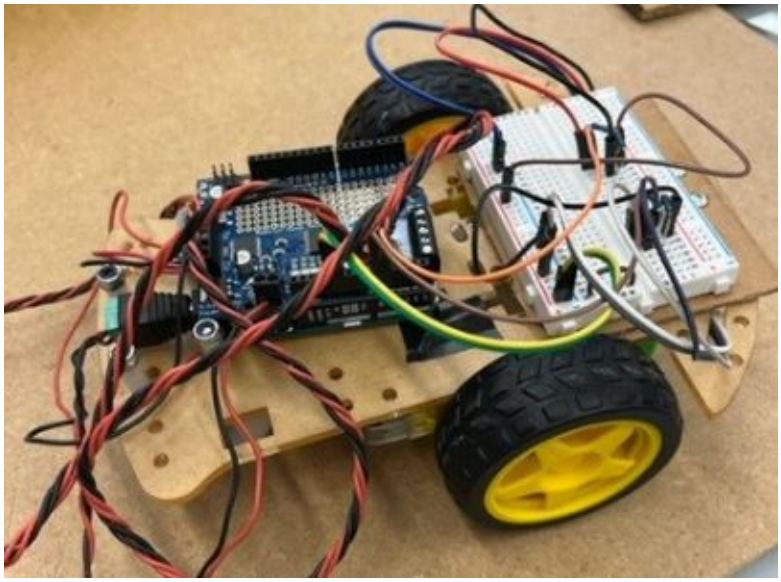

Figure 2: This is the top view of the robot, which displays the circuit on the breadboard.

While testing our sensors, we quickly realized that they needed to be secured somehow to allow for accurate and consistent measurements. To do this, we reused 3D print pieces by securing them to the chassis and then securing our sensors to the pieces. These held the sensors in place at the same height and assisted us in reading accurate data.

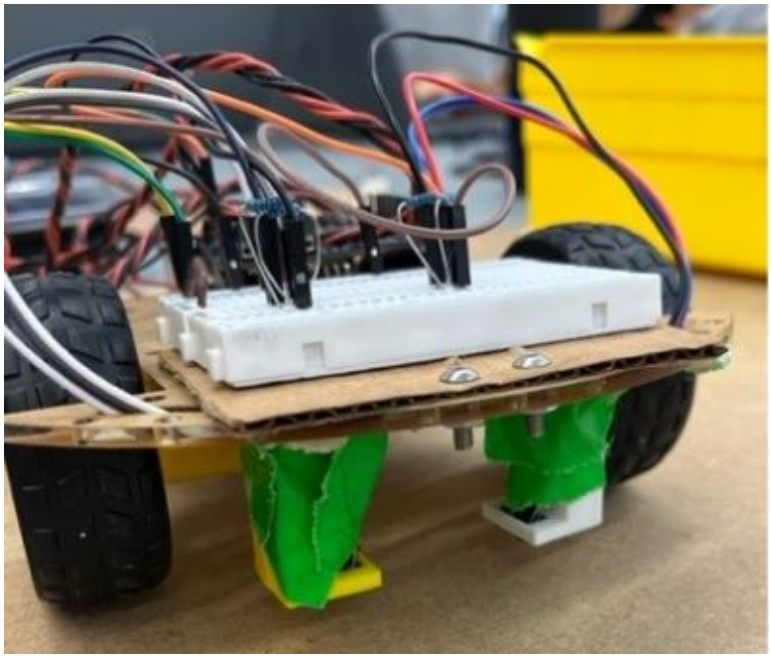

Figure 3: This is the front view of the robot, showing even sensors.

5 Arduino Integration

After calibrating our infrared sensors, we knew to program our motors around a measurement of 500 for the left sensor and 750 for the right sensor (we don't have units for these numbers – they are simply the values returned from analogRead). When the left sensor is over the line (>= 500), the motors are commanded in directions that turn the robot left. When the right sensor is over the line (>= 750), the motors are commanded in directions that turn the robot right. The idea is that if the robot drives over the line to the left (for example), then the right sensor will see the black line, and the robot should turn to the right.

```
// If left sensor over line, turn left
if(analogRead(ir_port) >= 500) {
  motor_port->run(FORWARD);
  motor_stbd->run(BACKWARD);
  delay(100);
}

// If right sensor over line, turn right
else if(analogRead(ir_stbd) >= 750) {
  motor_port->run(BACKWARD);
  motor_stbd->run(FORWARD);
  delay(100);
}
```

Figure 4: The Arduino code that handled the motors when the robot was over the line.

After trial and error, we found that the fastest the robot could go while accurately following the line was a speed of 25 within an analog range of 0 to 255. As a result of the way we put the DC motors on, commanding the motors to go forward was actually backward for us. When commanding the robot to turn, the two options for the speed are analog 25 (backward) and analog -25 (forward). We also found, through trial and error, that forcing the robot to continue turning or going straight for 100ms allowed it to handle sharper turns better – if it changes direction too fast, then it ends up wobbling left and right without going forward.

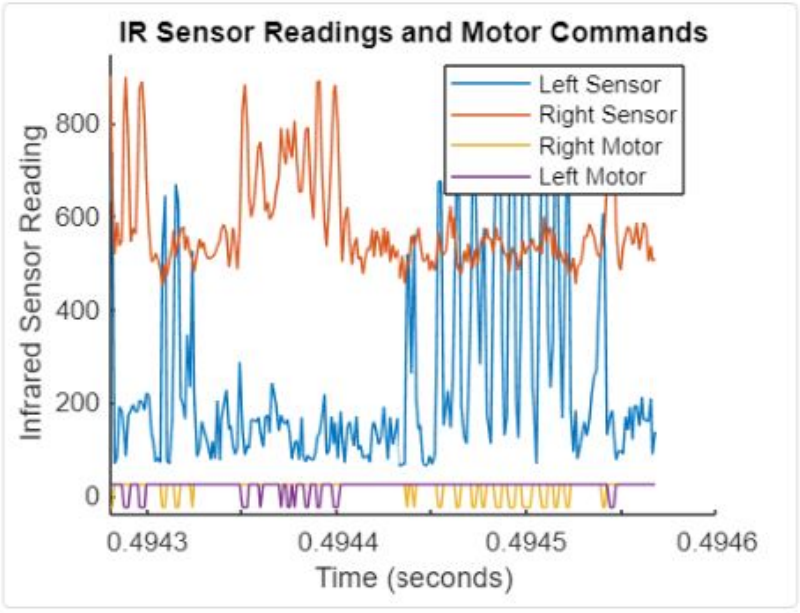

Figure 5: Plot of sensor measurements and motor commands for a short trial run.

In order to control our robot in some form via the serial connection, we utilized a couple of Serial functions. First, we used Serial.available() within an if-statement to see if there was anything to be communicated from the serial port. If there was, we used the Serial.parseInt() function to read the input into the serial monitor. If it was 1, the robot would start line-following. If it was 2, the robot would stop.

```
29  void loop() {
30    if (Serial.available() > 0) {
31      input = Serial.parseInt();
32
33      if(input == 1) {
34        motor_port->setSpeed(25);
35        motor_stbd->setSpeed(25);
36      } else if (input == 2) {
37        motor_port->setSpeed(0);
38        motor_stbd->setSpeed(0);
39      }
40    }
```

Figure 6: Our Serial control conditional statements.

We wanted our behavior to change the speed of the robot between a faster and slower speed. However, given our mechanical design, we found our maximum speed to be 25 – anything higher and the robot will likely drive off the track. Then, unfortunately with our motors, anything noticeably slower is too slow and the robot doesn't drive. So, we settled on a behavior change for toggling the robot between driving and stopping.

6 Reflection

Throughout this mini project, we learned a lot about the Serial communication protocol. For instance, Serial data is buffered. So, when the computer is sending data to the Arduino, it gets stored in a buffer until the Arduino reads it. Serial.available() returns how many bytes of data are in the buffer. Most importantly, when you read from the buffer, you remove the bytes you read from the buffer but you don't empty the buffer unless you read all the bytes. When using Serial.parseInt() to read a number command from the serial port, we found that sending the message with a newline character created inconsistencies. This was because the newline is still in the buffer after the character has been read. Additionally, we found that the sensors were quite difficult to work with. They needed extreme stability. At first, our sensors were not secured beneath the robot. We found that if the sensors moved even a little, our calibration was invalidated – the light thresholds for determining the difference between tape and floor would change. To provide consistency, we recycled sensor holders to secure our sensors to the chassis. After calibrating, we didn't have to re-determine the thresholds.

7 Results

Successful Runs Here is a link to a successful run of our line-following robot: https://www.youtube.com/watch?v=63zhl5NgQYA

Here is a link to a demonstration of our behavior change: https://www.youtube.com/shorts/GguOWYQUGMQ

Here is a link to a demonstration of our behavior change on the track: https://www.youtube.com/shorts/yq6FzR06kXo

7.1 Source Code

MATLAB code for the plot of motor speeds and sensor measurements over time: https://github.com/nsacks2905/PIE2022/blob/main/MP3/graph.mlx

Arduino code for robot: https://github.com/nsacks2905/PIE2022/blob/main/MP3/controller