Principles of Engineering Lab 3
Line Following Robot

Katerina Soltan, Regina Walker, Bryan Werth

October 13, 2016

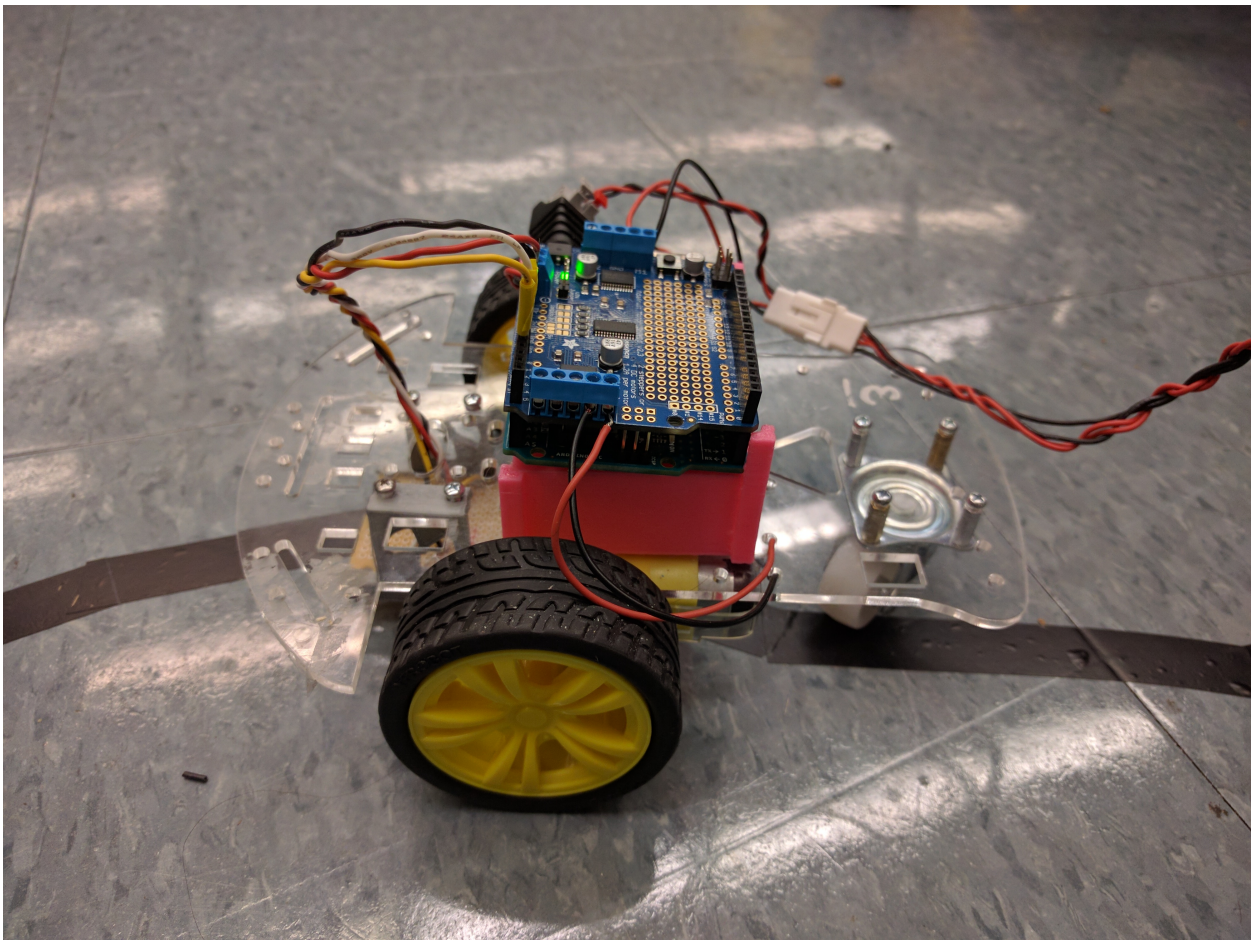

Figure 1: The above image shows our final chassis, lovingly dubbed "Derpbot," on the tape track it had to navigate.

1 Introduction

In this lab, we created a robot, named Derpbot, that can follow a line of electrical tape on the floor. Using two IR sensors to detect the reflectance of the surface below, Derpbot self-corrects its path to always stay on the taped circuit.

2 System Overview

An Arduino interfaces between two IR sensors, mounted underneath the chassis, and two motors connected to wheels. Using the analog output of the sensors, the Arduino determines the position of Derpbot relative to the tape: if the port sensor is on the tape, the robot is veering to the right, and if the starboard sensor crosses the tape, the robot is veering to the left. Depending on the position of the robot, the Arduino reverses one of the motors to compensate for the veering.

2.1 Design Process

We developed Derpbot gradually by building little "unit tests" and making sure they all worked before putting them together into a robot. We created the IR sensor circuit first and made sure we could tell the difference between electrical tape and the floor. Next, we connected the motors to the Arduino and learned to control them.

The code was structured to be very versatile and easily modifiable. We created a sketch of what we thought the robot would be doing without using any real outputs and creating a separate function which we could fill in later that provided the decisions of whether we were on tape or not.

2.2 IR Sensor

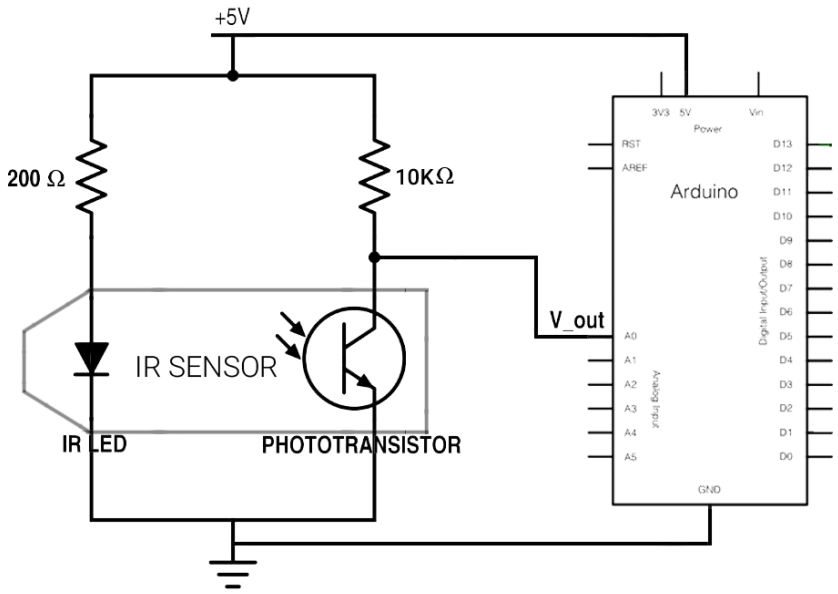

Figure 2: TCRT5000 IR Sensor Wiring Diagram

The IR Sensor is composed of two parts: an infrared LED and a phototransistor. The recommended 200Ω in front of the LED portion creates a $25mA$ current through the LED, well under the maximum $60mA$ forward current. For the phototransistor, the maximum current is $100mA$. Using Ohm's Law, $V = IR$, where the total voltage drop is $5V$, the resistor value must be at least 200Ω. The typical current through the phototransistor is $1mA$, which is a $5K\Omega$ resistor.

The output of the IR sensor is equal to $V_{out} = 5 - i_c R$, where i_c is the current through the resistor. To have a discernible V_{out}, the resistor should be fairly high. The current flowing is pretty small, and the higher the resistance, the higher the voltage. We chose a $10K\Omega$ resistor to both amplify the V_{out} and make sure that a smaller amount of current is flowing through the circuit.

The original circuit was constructed on a breadboard and can be seen in Figure 3.

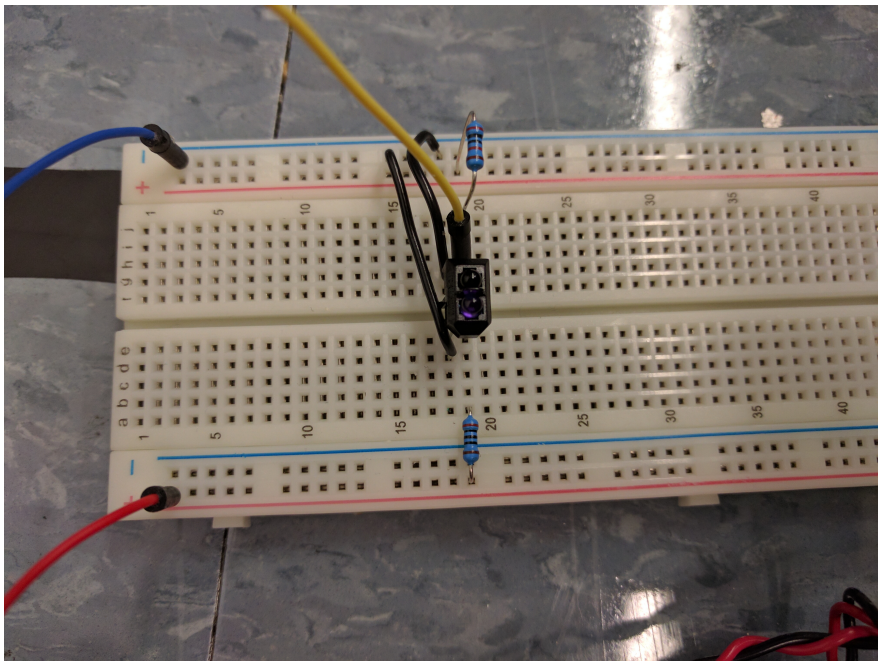

Figure 3: This figure shows the initial circuit we built while testing our IR sensors. We used the same circuit for our final circuit board, duplicating it to account for our second IR sensor. The resistor on the bottom half of the image is the 200Ω resistor connected to 5V and the led, while the resistor on the top half of the image is the 10KΩ resistor connected to 5V and the phototransistor. The yellow wire between the 10KΩ resistor and the phototransistor is where we measured output voltage.

2.3 Calibration

With a working circuit, we tested the sensor to understand the difference between the amount of received light when the sensor is over the tape line versus the linoleum floor. First, we tested the sensor in a closed system at our workstation. Because the sensor can be either on or off the tape but can have fluctuating values on the same material, we were searching for a threshold between the light received over the linoleum versus over the tape. To do this, we connected the IR sensors to our Arduino and found the approximate scaled voltage values to the left, right, and over the tape for each sensor. Three separate trials of this nature were performed for each side, the results of which can be seen in Tables 1 and 2.

Initially we ended up with a threshold value of 500 on a scale from 0 to 1023[1] - if the sensor was returning a value of 500 or higher, it was over the tape because tape is more reflective than the floor. However, there was a large amount of environmental variability between our closed system and the actual course. As a result, we re-calibrated a number of times, eventually ending up with a threshold of 300 on the port side and 500 on the starboard side (this final test is the one reflected in the two tables below). We had different readings for the IR sensors because we could not guarantee perfect alignment. Because the sensors are highly sensitive, the smallest change of angle to the ground returns different values.

[1]The Arduino's analog output it a mapping of voltage from 0 to 5V to a scale from 0 to 1023.

Table 1: Port Side IR Sensor Calibration Data

Test Number	Port of Tape	Over Tape	Starboard of Tape
1	40	371	42
2	42	351	41
3	42	360	42

Table 2: Starboard Side IR Sensor Calibration Data

Test Number	Port of Tape	Over Tape	Starboard of Tape
1	121	646	270
2	230	642	155
3	236	655	229

2.4 Electronics and Hardware Design

With a circuit design proven to work, we moved our circuit design from a breadboard to a protoboard to make it more compact and easily maneuverable, shown in Figure 4. All of our connection nodes are on the side of the protoboard not pictured below: we were unfortunately not careful to place the components in need of soldering on the copper coated side of the board.

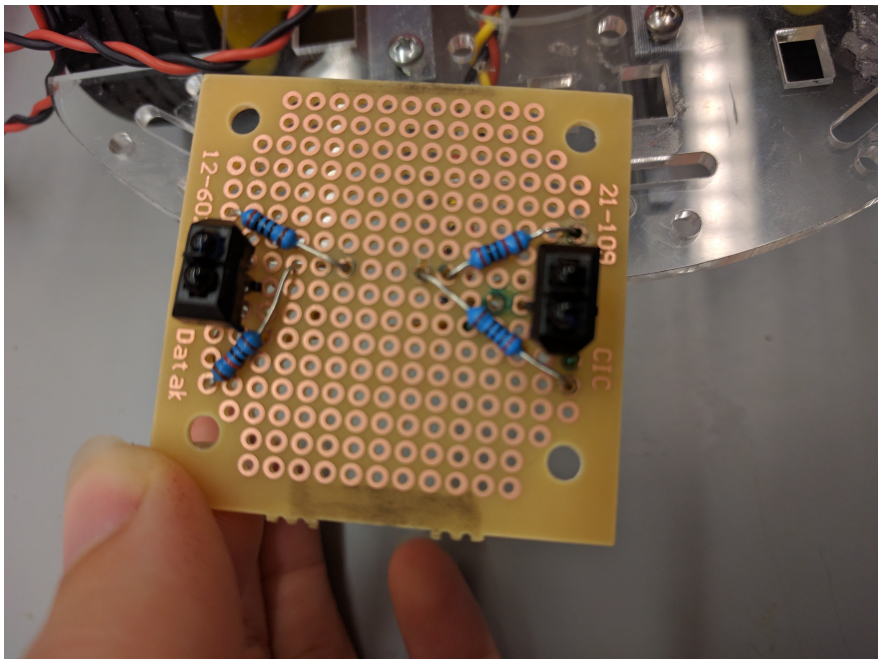

Figure 4: This image shows the protoboard circuit we used to connect our chassis' IR sensors to our Arduino. The top right and lower left resistors are the 10KΩ resistors, while the top left and lower right have values of 200Ω

Because we needed two sensors to determine our position relative to the tape, we mirrored the sensor diagram on both sides of the protoboard, with a space between them greater than the width of the tape. Ground and power were shared by the two sides, making a total of 4 wires come out of the board to interface with the Arduino. The circuit diagram can be seen below:

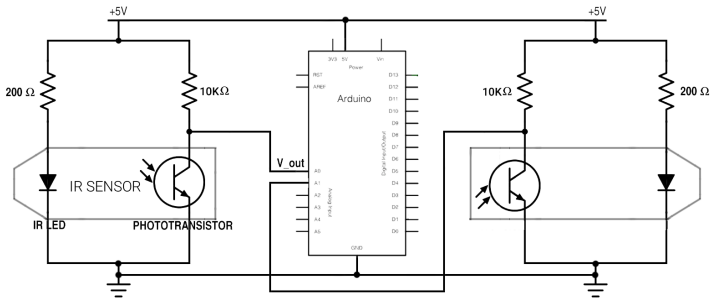

Figure 5: The superposition of motor outputs and voltage readings from the sensors explains how Derpbot operated. When one of the sensors spikes, or encounters tape, the corresponding motor is powered backwards to turn the robot quickly. Such corrections are made as many times as necessary to right Derpbot.

The peak operation distance for the IR sensors is about 2.5mm, and therefore we designed a mount for our robot chassis to suspend the protoboard with the sensors about 3mm from the ground. For easy debugging, we wanted to be able to remove the protoboard quickly without having to unscrew the whole mount. We achieved this by creating a bracket in which we could slide in a side of the protoboard. With a second such bracket on the other side, the protoboard was fully supported underneath the chassis and easily removable. The CAD version can be seen in Figure 6, while the printed version can be seen in Figure 7 already mounted to the chassis.

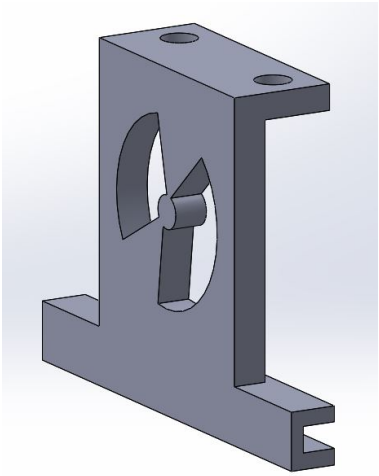

Figure 6: This is the CAD model for our 3D printed IR sensor mount. This image only shows one side of the mount; the two sides are identical to one another. For the actual chassis we printed out two of these sides, and then screwed them unto the bottom of the chassis with the slots facing inwards.

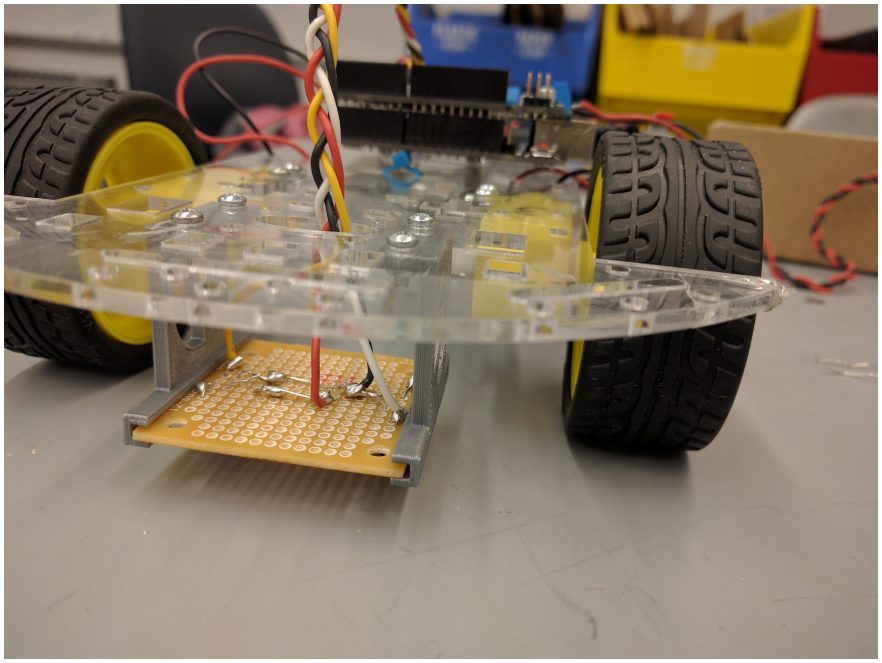

Figure 7: This image shows the 3D printed mount–the gray plastic pieces–after it's already on the chassis. The protoboard can slide in and out of this mount, positioned to be able to thread the 4 wires (power, ground, and analog outputs) through a hole in the chassis and reach the Arduino easily.

Next, we connected our motors to our Arduino using the Adafruit motor shield, shown in Figure 8 (The motor shield is the part directly visible; it is on top of the Arduino). There is too much current going to the motors which the Arduino cannot handle; the motor shield "shields" the Arduino against this and takes care of power distribution. A wiring harness was made to connect with a power supply and power the Arduino and motor shield independently from a laptop. When a laptop is being used to send commands to the Arduino, the Arduino is powered from the laptop while the motor shield gets its power from the outside power supply.

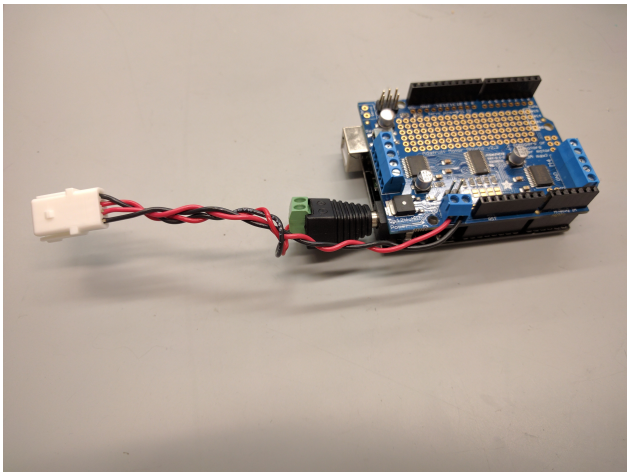

Figure 8: This figure shows the motor shield (and, barely, the Arduino beneath it) we used to hook up our motors without powering them via the Arduino. The red and black wires shown connect the Arduino and the motor shield to power. The motors aren't shown connected, but they attach to the blue screw ports on the left and right side (in this image) of the motor shield.

To provide an elevated support for the Arduino and the motor shield, without entangling wires from motors or getting in the way of wheels, we designed another 3D-printed support, the CAD for which can be seen in Figure 9, while the printed out mount can be seen in Figure 10.

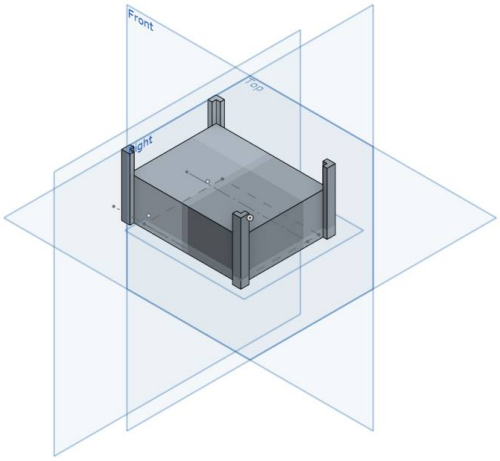

Figure 9: This is the CAD model for our Arduino/motor shield. The Arduino and motor shield are held in place by the 4 tabs sticking up off the platform. The platform itself is elevated to prevent the wiring to the motor shield from interfering with the wheels.

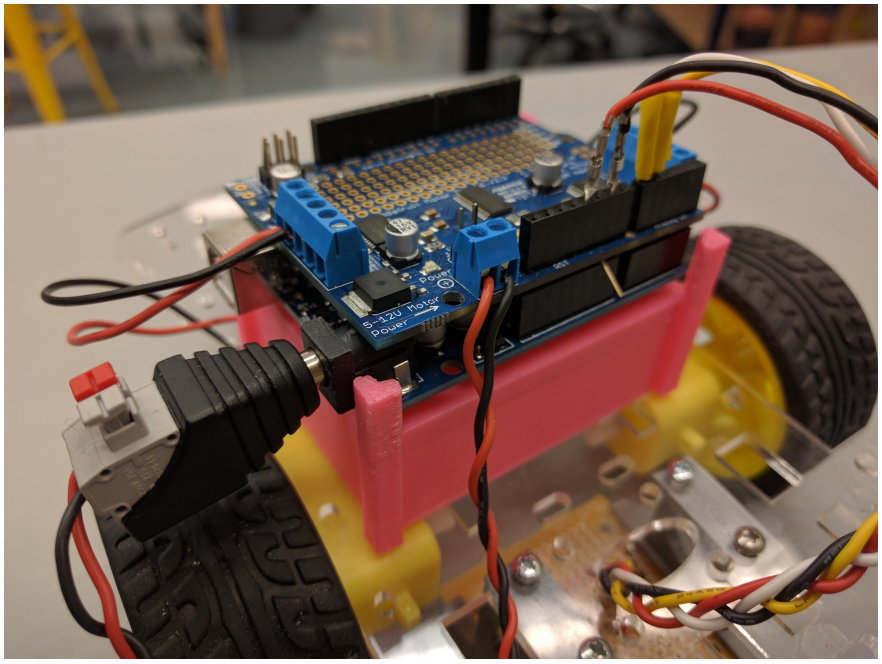

Figure 10: The pink 3D printed part in this image is the Arduino/motor shield mount. It serves to hold the two in place during testing, while also keeping the wiring up out of the way of the wheels.

After making all of these parts, we needed to put them together to create our working chassis. The final assembly can be seen in Figure 11. The power, ground, and output voltage wires from the protoboard are connected to the 5V, GND, A0, and A1 pins respectively of the motor shield. Because the motor shield sits directly on top of all of the Arduino's pins, connecting the pins to the motor shield achieves the same result

as connecting them directly to the Arduino.

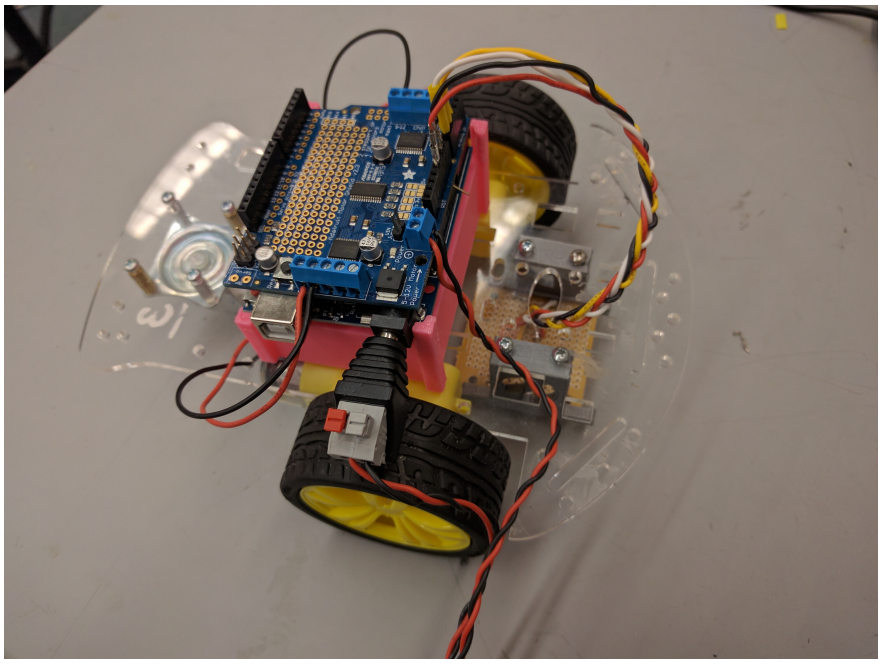

Figure 11: This image shows our final robot assembly (dubbed Derpbot).

2.5 Running our Robot

To allow Derpbot to correct itself in its journey along the tape, we chose to use a bang-bang controller. The chassis moves forwards in a straight line until one of the two IR sensors detects the tape line, signaling that the robot is off course. The chassis turns in the appropriate direction to get back on course, then continues forwards until it detects the tape again.

In this code, we first set up the motor and motor shield, and set the motors to speed 17 (we tried various speeds, but this was the fastest setting that allowed our chassis to navigate all of the track's turns). Then, we created Boolean flags that would return 'true' when that sensor was over the tape, based on the calibration thresholds, as shown below:

```
8  boolean isOffCoursePort = false;
9  boolean isOffCourseStarboard = false;
```

With these flags, the chassis continues to correct itself until it is no longer on the tape. These flags are turned 'on' and 'off' in two boolean functions, 'isOffCoursePort' and 'isOffCourseStarboard'. The port function is shown below:

```
8  boolean isPortOverTape(){
9    int portValue = analogRead(port);
10   isOffCoursePort = portValue > 300;
11   return isOffCoursePort;
12 }
```

These functions take their corresponding IR sensor value (either port or starboard) and check whether that value is above our cutoff value. If it is larger than a scaled threshold voltage, the function returns the flag above to be 'true;' if not, the flag remains 'false.'

With this groundwork laid, we created the loop that gave different instructions to the chassis's motors depending on the sensors' feedback, as shown here:

```
8  void loop() {
9  // Instructions for the motors if the car is off course to the port side
       or starboard side, or if it is on course
10 // (assume that only one out of the two sensors is over the tape at any
       point)
11     if (isPortOverTape()) {
12       portMotor->run(BACKWARD);
13       starboardMotor->run(FORWARD);
14     }
15     else if (isStarboardOverTape()) {
16       starboardMotor->run(BACKWARD);
17       portMotor->run(FORWARD);
18     }
19     else if (!isOffCoursePort && !isOffCourseStarboard){
20       portMotor->run(FORWARD);
21       starboardMotor->run(FORWARD);
22     }
23   }
24 }
```

In the code above, if the 'isPortOverTape' function returned true, we knew that the chassis had skewed off course to the starboard side. To correct this, we ran the port motor backwards, keeping the starboard motor running forwards, thus rotating the chassis back towards the port side until the flag no longer returned 'true.' The same process applied for the 'isStarboardOverTape' function, except we ran the opposite motors backwards/forwards. Finally, if neither of those functions returned true, we simply ran both motors forwards, as that meant the chassis was on course.

As a note, we originally turned our chassis by stopping one motor, while allowing the other to keep running. However, when we tested our chassis on the actual track, we discovered that we could not successfully make it around all of the curves. Our turn radius was too large to make it all the way around the sharper corners, as the turns would take the sensor off the tape before the chassis had completely corrected itself. Thus, we altered the code to run one wheel backwards while the other continued forwards, decreasing our turn radius and increasing our success.

3 Results

The following graph depicts the sensor readings and responses of Derpbot's motors when going along part of the track. Orange readings represent the port sensor and motor while the blue is the starboard side electronics. At the beginning of this section of track, Derpbot's port sensor encountered tape, seen in the spike of the value, causing the starboard motor to continue going full speed forward and powering the port motor backwards. In the middle of the test, Derpbot was physically moved after a hiccup, rendering the data not descriptive of the actual process when the sensors are near the ground. The end of the test track featured a gradual port turn which caused the starboard sensor to hit the tape multiple times and trigger correction from the motors until it was back on track.

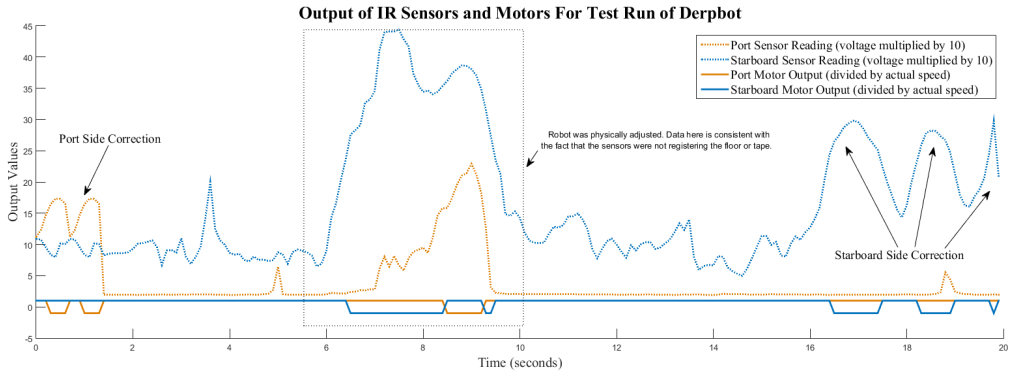

Figure 12: The superposition of motor outputs and voltage readings from the sensors explains how Derpbot operated. When one of the sensors spikes, or encounters tape, the corresponding motor is powered backwards to turn the robot quickly. Such corrections are made as many times as necessary to right Derpbot.

Our robot completed the course in approximately two minutes. Leading up to the final successful test, there were a series of slight failures where the robot had to be pushed back onto the course. With minor adjustments to our sensor calibration values, the motor speed, and the chassis turning procedure, the reliability of the robot improved enough for it to complete the course without help. Here is a link to the video of Derpbot completing the course: video link

4 Reflection

The robot successfully finished the course, but the reliability and speed of the robot could be improved. First, the robot was programmed using a bang-bang controller. As a result, the robot was inefficient in traveling the course, going relatively straight until it went off course and then going the other way. The speed of travel could be improved if we switched to a proportional-integral-derivative (PID) controller. PID is a more sophisticated controller that constantly calculates an error value for the distance between the correct location and where the car is. A correction based on the error value is applied after being calculated, as the controller would suggest, through proportional, integral, and derivative methods. The speed of the motors themselves could also be increased if we were to continue. Depending on which power connectors we were using, Derpbot was successful from speed 17 to 21. If we could figure out the proportionality of the speed at which we powered the motors backwards vs forwards for making tight enough turns, it would be completely possible to make Derpbot complete the course faster. It would be interesting to somehow automatically adjust the calibration of the IR sensors depending on the environmental lighting conditions. This would reduce the need for manual re-calibration when the location of the course changes.

In terms of design, we could have made the support for the boards attach to the chassis with screws. We would have also wanted to re-solder our protoboard the way it was intended to be used. Our code does not take into account a scenario where both of the sensors are on tape. If it was going at high enough speeds, this could be a very likely possibility, and an interesting problem to solve.

Every member of the group tried their hand at parts of the lab they were not completely comfortable with. We tried to make sure that everyone got the chance to do something they do not usually do in a supportive environment. We sacrificed the aforementioned improvements for this opportunity, but are very satisfied with what we got out of this lab.

5 Source Code

```
8  //Incorporate the motorshield and motors
9  #include <Wire.h>
10 #include <Adafruit_MotorShield.h>
11 #include "utility/Adafruit_MS_PWMServoDriver.h"
12 Adafruit_MotorShield AFMS = Adafruit_MotorShield();
13 Adafruit_DCMotor *portMotor = AFMS.getMotor(4);
14 Adafruit_DCMotor *starboardMotor = AFMS.getMotor(1);
15
16 //Create flags that are assigned when one of the IR sensors crosses the
      tape
17 boolean isOffCoursePort = false;
18 boolean isOffCourseStarboard = false;
19
20 //Assign pins to the two IR sensors
21 int port = A0;
22 int starboard = A1;
23 int v = 17
24
25 void setup() {
26   //Set up the motors and their speed
27   AFMS.begin();
28   portMotor->setSpeed(v);
29   starboardMotor->setSpeed(v);
30 }
31
32 void loop() {
33 // Instructions for the motors if the car is off course to the port side
      or starboard side, or if it is on course
34 // (assume that only one out of the two sensors is over the tape at any
      point)
35     if (isPortOverTape()) {
36       portMotor->run(BACKWARD);
37       starboardMotor->run(FORWARD);
38     }
39     else if (isStarboardOverTape()) {
40       starboardMotor->run(BACKWARD);
41       portMotor->run(FORWARD);
42     }
43     else if (!isOffCoursePort && !isOffCourseStarboard){
44       portMotor->run(FORWARD);
45       starboardMotor->run(FORWARD);
46     }
47   }
48 }
49
50 /*Checks if the port sensor is over the tape.
51 If not, sets the offCourse flag to false. If on tape, sets the flag to
      true.
52 Returns isOffCoursePort*/
53 boolean isPortOverTape(){
54   int portValue = analogRead(port);
55   isOffCoursePort = portValue > 300;
```

```
56    return isOffCoursePort;
57  }
58
59  /*Checks if the starboard sensor is over the tape.
60  If not, sets the offCourse flag to false. If on tape, sets the flag to
        true.
61  Returns isOffCourseStarboard*/
62  boolean isStarboardOverTape(){
63    int starboardValue = analogRead(starboard);
64    isOffCourseStarboard = starboardValue > 500;
65    return isOffCourseStarboard;
66  }
```

6 Links

Full link to video: https://photos.google.com/share/AF1QipN8huB806HVib5ZeDLacNehl6m7VEN1dFCbix4FXHCEsRh2W

Project on Github: https://github.com/ksoltan/Line-Following-Robot