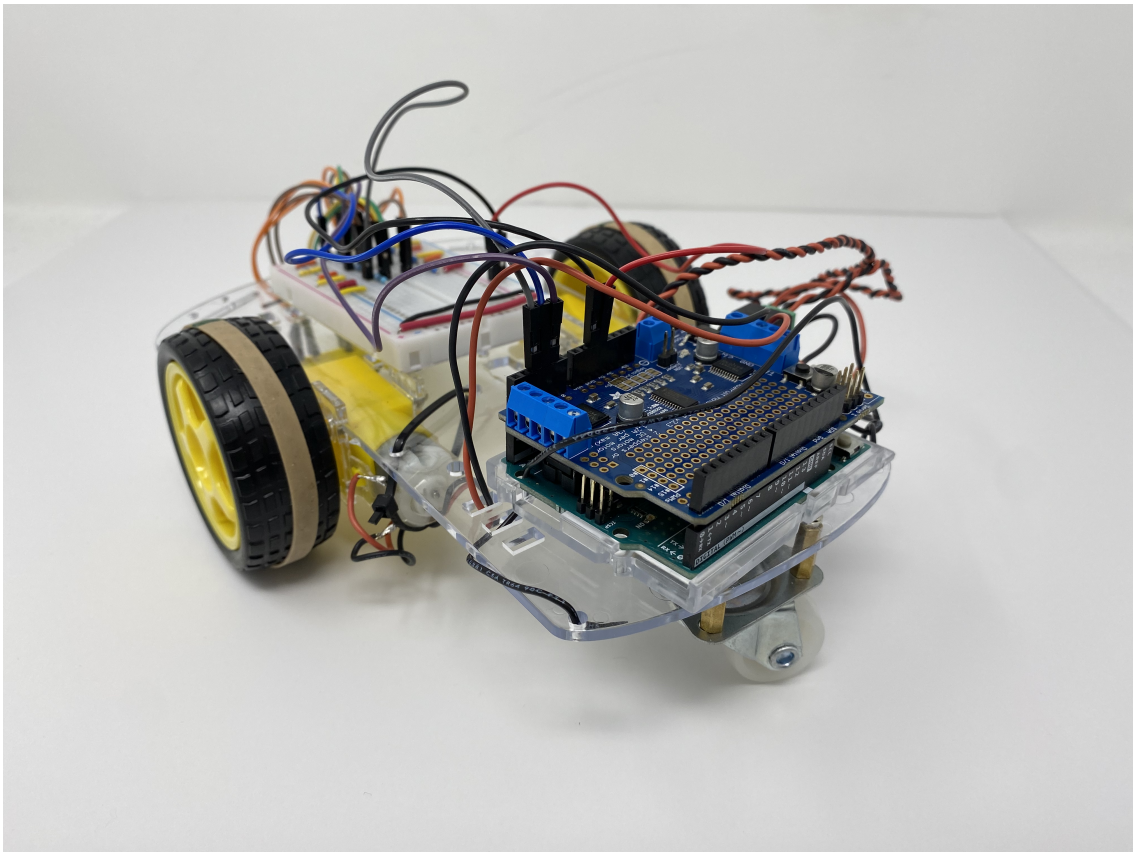
Principles of Integrated Engineering

# DC Motor Control

Avery Clowes
Grant Miner
Annika Pfister

October 22, 2021



# Contents

# 1 Introduction

The primary goal of this project was to create a line following robot that completes a closed loop course as fast as possible. Our group was given a set of DC motors, infrared reflectivity sensors, and a mounting board. To bring these components together, we used an Arduino Uno to read sensor data, a motor shield board to control the motors, a breadboard to electrically connect to the sensors to the Arduino with appropriate resistors, and a sensor mount to optimally position the sensors. The entire assembly was run by code on the Arduino.

Our robot used three calibrated reflectivity sensors to check for the black electrical tape line on the tile floor. The values they returned to the Arduino through analog input ports helped guide the robot along the path without getting stuck. It only stopped when it reached a line of tape that crossed the main course. It successfully drove around the entire course in both directions without error.

The code written for this robot also allows for anyone connected via a serial port to update the analog threshold interpreted by the Arduino as the point at which the black electrical tape is detected. Moving this threshold above the point where the black tape is first detected can drastically change the function and path of the car because the analog values read from the reflectivity sensors can change significantly due to other lighting conditions or dust on the surfaces.

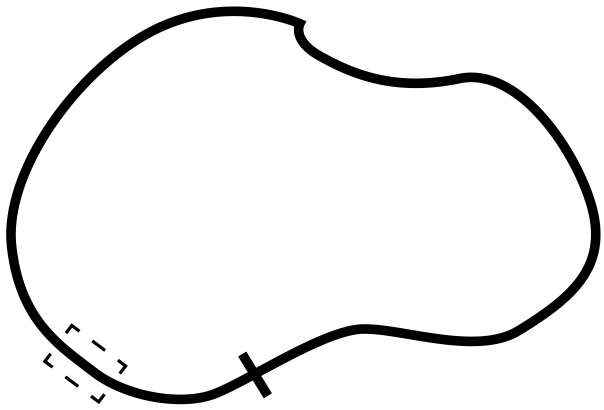


Figure 1: Example course with start/end cross indicator. Dashed square indicates relative robot size

# 2 Mechanical Design and Integration

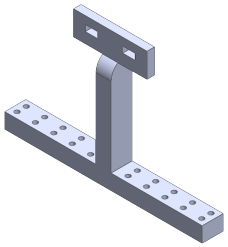
The sensors we used for this project were TCRT5000s, each one composed of a infrared diode and phototransistor. For optimal performance, the datasheet suggested a 2mm working distance from the object to the sensor face. To accomplish this we designed and 3D printed a sensor mount. It has a series of mounting holes along the bottom edge to allow for testing different positions of the sensors. It also has a set of alignment holes at the top for integration with the existing holes in our mounting plate.

The extra mounting holes proved to be very helpful when implementing the robot, as we decided to add a third sensor in between with opposite logic from the other two (to keep the sensor on the line, instead of off of it). We were able to easily move the sensors outward along the mounting holes to change the robot's behavior, though we realized that keeping them closer together actually improved the function of the robot and allowed the robot to move faster.

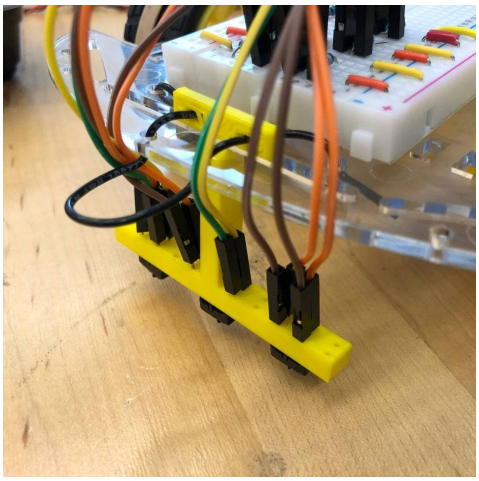
Our mechanical integration also allowed for rapid changes in the chassis design, and the flexibility of our mounting proved more helpful than expected. We were able to very easily remove the sensor mount to add or rewire sensors without having to remove any screws or bolts. The same was true of

our breadboard, especially when we decided to add a third reflectivity circuit to our robot. Below in figure 2, the mount CAD is shown, as well as the integration with the robot.

To assemble our chassis we followed the design instructions from the kit. Importantly though, we incorporated a series of mounting and strain relief methods do decrease the wear on our components. When mounting our motors, we used zip ties that held the power wires to the DC motors by their insulation. This decreased the strain on the solder joints, hopefully improving overall reliability. We also used a series of wires threaded through the chassis holes and into the mounting holes on the Arduino to hold them in place. This allowed us to easily change orientation if necessary.



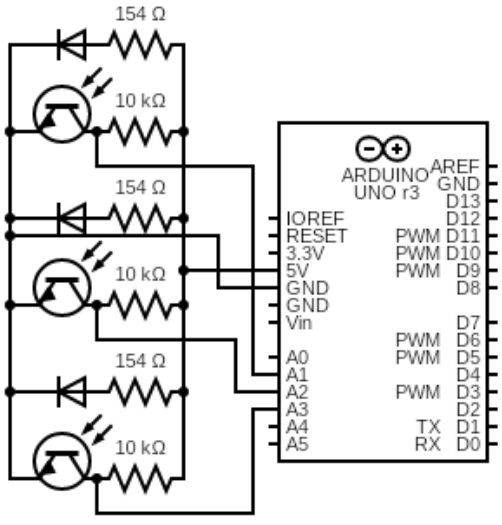
(a) A CAD screenshot of the sensor mount design.



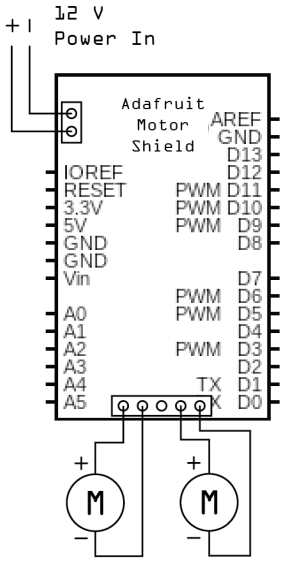
(b) The sensor mount assembled on the robot

Figure 2: Sensor Mount

## 3 Electrical Design




(a) TCRT5000 sensor integration diagram design.




(b) DC motor integration diagram

Figure 3: Circuit Diagrams for sensors and motors

We used a breadboard to assemble our sensor circuit, while the DC motors were connected directly to the Adafruit v2.3 Motor Shield mounted in header pins on the Arduino. DC motors were powered externally via a 12V power supply connected to the motor shield.

### 3.1 Calibration

To calibrate our infrared sensors, we took five measurements on three different surfaces: electrical tape, the tile floor, and a wood table. We averaged these values and used them to inform our threshold, such that it fell between expected readings for the tile floor (maximum 130) and the electrical tape (minimum 600.2), thus allowing the robot to clearly differentiate between the two surfaces.

| | Left (A1) | Right (A2) | Center (A3) |
|---|---|---|---|
| Electrical Tape | 623.6 | 652.8 | 600.2 |
| Tile Floor | 54.2 | 130 | 56 |
| Wood Table | 38.8 | 39.8 | 39 |

Table 1: Average Sensor Readings for Five Trials on Each Surface

## 4 Controller and Code Outline

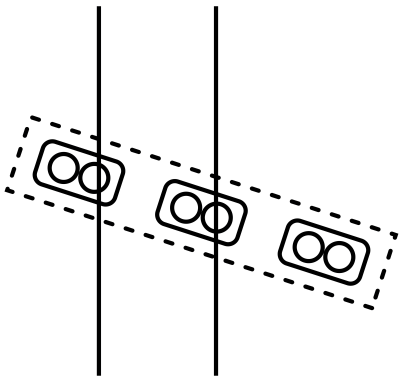
Originally, we had planned to control the Arduino from a Python script, and only control the motor speeds and read from the infrared sensors from the Arduino. We planned to pass all of this information to the Python script, and use Python to determine what the robot should do next. We ran into issues with this approach because both the Arduino and the Python code were trying to write and read from the serial port at the same time.

After attempting to debug this problem for an extended amount of time, we discovered the input box in the Arduino's serial monitor that allows data to be sent to serial by a user. We quickly pivoted, and we rewrote our entire logic and wheel command codebase in Arduino, abandoning Python completely. This choice saved us great amounts of time by reducing debugging and overall complexity.

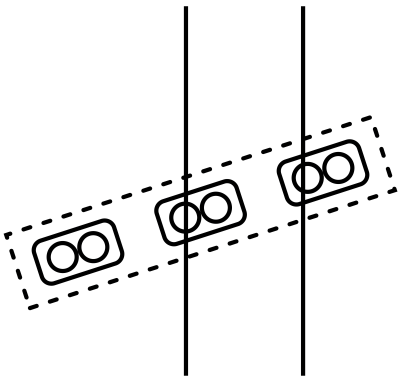
Our new controller uses the analog inputs read from the three sensors to determine the robot's orientation in relation to the line, all in Arduino code.

**Once the robot begins moving, we check each of the following cases continually in our control loop.**

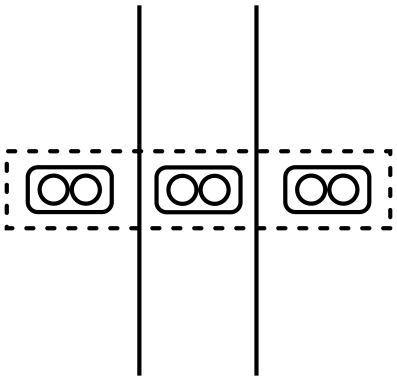
1. Left sensor detects line, so robot turns left wheel backward and right wheel forward to move the left sensor away from the line.
2. Right sensor detects line, so robot turns right wheel backward and left wheel forward to move the right sensor away from the line.
3. Only the middle sensor detects the line, so both wheels turn forward and continue moving along the path.
4. None of the sensors detect the tape, so the robot runs the wheels in opposite directions until it intersects the line again.
5. All sensors detect the tape, so the Arduino understands that it has hit the start/stop line, and stops the two motors.



(a) **Case 1:** The left sensor detects the line

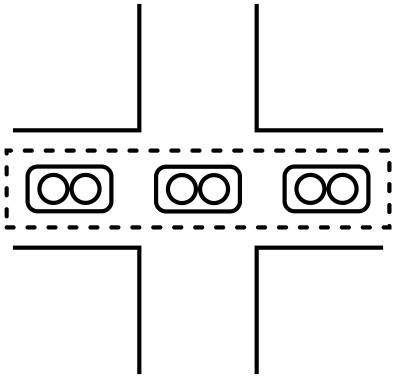


(b) **Case 2:** The right sensor detects the line



(c) **Case 3:** Left and right sensors do not detect the line, but the center does

(d) **Case 4:** No sensor detects the line



(e) **Case 5:** Both sensors detect the line

All of these recovery modes rely on the Arduino's computational speed. Because the Arduino can catch the cases the moment they occur, such as when the entire vehicle leaves the line, it is able to correct itself rapidly.

An interesting edge case is when the robot has left the line, and goes into its rotation mode to find the line again. No matter which side sensor hits the line first, the program will continue to rotate in that direction until the sensor no longer touches the line. This design prevents the robot from getting

stuck if it is to leave the line, though it will get stuck if started far off of the line. The code only spins the robot when all sensors do not see the tape, so if the tape is not within the sensor's path, it will get stuck in a loop. However, if the robot encounters this problem in certain circumstances, it may reverse directions on the loop. There does not seem to be any way to predict its reversal, as this error only happened once among a large number of test runs.

Our code takes the form of a main control loop and a few helper functions to supplement it, which also make the code more readable and understandable. The main module is the Arduino loop() function, which calls the other functions to determine whether it needs to change the motor speeds. The complete code is available in the appendix.

## 4.1 Code Breakdown

The Arduino implements four main functions when it is driving untethered (not connected to a laptop): setup(), loop(), movementLogic(), and seesBlack(). The first function, setup() is included in every Arduino program. It sets initial values and connects variables to the pins they represent. It also sets the motors to initial speeds. Given its ubiquity, we will not explore it in this section.

To start with the helper functions, seesBlack() reads the sensor values and determines whether each sensor sees the electrical tape. The code below reads the analog value from a sensor pin and saves it to an integer, after which the value is checked against the threshold value. It then resets the boolean to zero, so that if it does not see black, the value is already set to zero. Lastly, the analog value is compared to the threshold, and it is only set to true if the sensor sees the color black.

```
1    int left_analog = analogRead(IR1PIN);
2    lb = 0; //set to "False"
3    if (left_analog > threshold) { lb = 1; }
```

Next comes determining what the robot should do. Our seesBlack() code does not return any values. Instead, it updates global variables for the various booleans each time it runs. The next function, movementLogic() takes these booleans as inputs, as well as current speeds for the motors, and the various speeds the motor may take. Refer to the appendix for the complete case logic or see the diagrams and explanations on pages 4-5.

The code below demonstrates the base case where the center sensor sees the black tape, but the left and right sensors both do not detect it. In this case, both motors are set to the "cruise" speed. Other speed options include "slow," for the motor to slow and let the other turn it until it is aligned with the track again. "Catchup" used to be a speed faster than "cruise," but we determined that in the turning state, having both motors turn at the same speed worked perfectly well. "stop_speed" is the speed the motor takes on when it wants to stop. This is defaulted to zero, but if the user does not want the robot to stop at the end of the track, the number can be made to be positive.

```
1    if (!lb && !rb) {
2        ls = cruise;
3        rs = cruise;
4        }
```

This case checking is repeated for all of the other possible cases, updating the global variables for ls and rs, or left speed and right speed, respectively.

Finally comes the loop() function, which runs all of these helper functions together. It also deals with the actual setting of the motors. Below is a basic method for setting the speed of the left motor only.

```
1    seesBlack();
2    movementLogic(lb, rb, cb, threshold, cruise, slow, catchup);
3    motorLeft->run(RELEASE); //stop motor
4
5    if (ls > 0) {
```

```
6           motorLeft->run(FORWARD);
7       }
8       else {
9           motorLeft->run(BACKWARD);
10      }
11
12      motorLeft->setSpeed(abs(ls));
```

The function runs the two helper functions first, to determine what the motors should do. After this, it releases the left motor in case it needs to be run backwards. It then determines whether the speed sent to it by movementLogic is greater than or less than zero, changing the motor's direction appropriately.

The last step is setting the motor's speed to the absolute value of ls, ensuring it rotates at speed ls but with the direction already determined. This code was implemented after bugs were encountered when trying to pass negative values to the motors' setSpeed() function.

The last function in our code is the parameter passing function, recvWithStartEndMarkers(). It was taken from an Arduino forum post, with minor modifications.(1) This function receives data from serial by looking for start and end markers, at which point it is parsed and used to change parameters in the code, *without ever recompiling or stopping the Arduino.*

The function first reads a character from the serial input. It waits until that character is the start character it is programmed with. It then makes a list including all of the characters received from serial until it receives the end character. The code below demonstrates the final assignment to the threshold parameter.

```
1       if (rc == endMarker) { //end of data
2               \\ split list at commas
3               strtokIndx = strtok(receivedChars, ",");
4               threshold = atoi(strtokIndx); //converts to integer
5           }
```

After it terminates reading characters, it makes sure that the final character is the end character, then turns the list into a string by splitting it at the commas. Finally, it turns the string into an integer and sets the threshold equal to that integer. All of this is performed only when there is serial data to read, so the code runs without any serial input going about its course, until the threshold value is changed.

For example, if "<700 >" is written to serial through the input field in the serial monitor, the Arduino will immediately update the threshold value from its default of 500 to the new 700 values, and continue trying to solve the course with this new value as a guide.

As discussed earlier, the threshold should be less than 600 to ensure that the sensors see the tape. Changing the value to 700 may result in the robot ignoring all of the lines completely, or continuing on unaffected, depending on lighting conditions and other factors.

In future iterations we would like to add the opportunity to update other parameters while the robot moves through the course, such as the cruise speed. While we have made the decision not to include our Python code in this report's appendix, it can be found at our Github page: Mini Project 3 Github Page or https://github.com/grantminer/pie-mp3.

# 5 Plot

The figure on the following page demonstrates the function of our robot on a relatively short time scale. The blue lines show the readings of our sensors, and the orange lines show the motor speeds. In this brief period, the robot was going around a left hand turn, so the right sensor rarely sees black at all. The motion is entirely triggered by the moments when the left sensor detects the black tape,

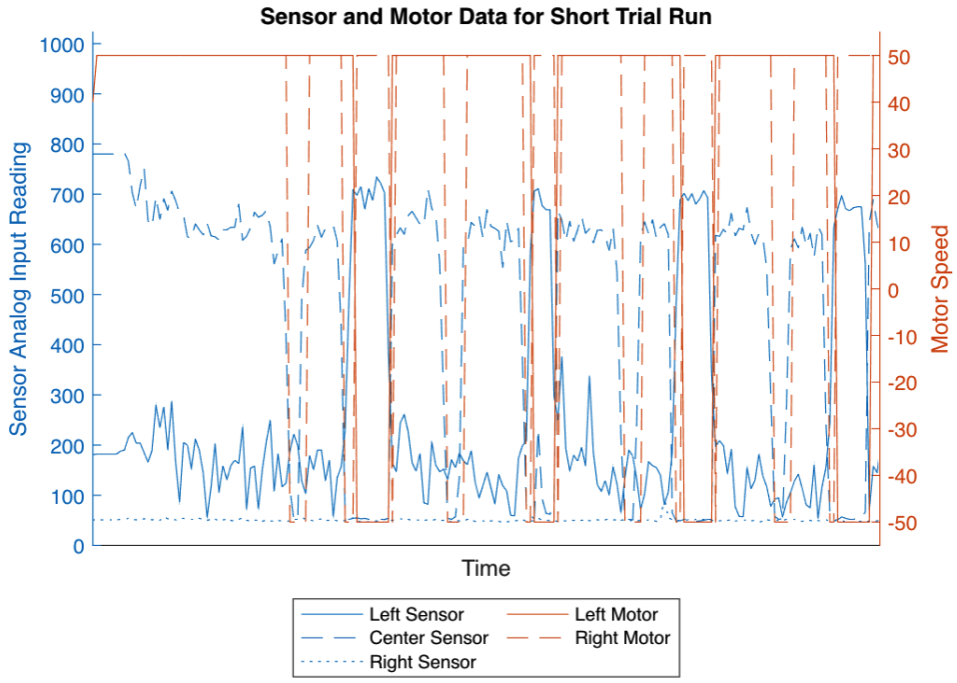



Figure 5: Sensor Readings and Motor Speeds

or when the center sensor no longer detects tape at all, and rotates to adjust. Dips in the center sensor readings are coupled with reversals of the right motor, and jumps in the left sensor readings are coupled with reversals of the left motor.

These behaviors match what we expect, further supported by the fact that the left motor and right motor are never moving in reverse at the same time. This plot also makes clear how obvious it is to the sensors whether or not they are detecting the black tape. We realized from this graph that there was more range between the two extremes than previously though, allowing us more flexibility in our threshold value.

# 6 Video

A video of our robot's journey can be found at https://drive.google.com/file/d/1bI7O2T8CgFSlq_EBbQRWedBmuAHyAGpS/view?usp=sharing

# 7 Reflection

## 7.1 Process

We began by assembling the chassis, then moved to integrating and checking components (ensuring that our sensors and motors worked) and ensured that we could drive the robot. We then identified the basic cases that the sensors would encounter and moved to coding. We initially attempted to send information between the Arduino IDE and Python over the serial port, such that the case logic was driven by Python while Arduino only sent sensor values and received motor speeds.

However, we realized that having both programs accessing the serial port simultaneously introduced port availability complications, and thus elected to implement the requirement that we control the robot via laptop through the serial COM port in the Arduino IDE (utilizing the input/send functionality).

We completely shifted the logic to our Arduino program, and then tested our code on the track,

making adjustments to sensor spacing and the sensor value threshold as necessary. We additionally added rubber bands around the wheels of the robot for increased traction. After encountering difficulty navigating the 90° corner, we also integrated a third sensor and added our fourth case, which prompts the robot to slowly spin until it finds the line again.

## 7.2 Final Product

Our final design drives at a default speed of 50 and is able to successfully complete the track. All code, including logic, is in Arduino. Sensors are connected to the breadboard via jumper cables, and supported and spaced approximately 11 mm apart (with the left and right sensors 32 mm apart) using our 3D printed sensor mount.

Aspects of this project that went well for us were electronic and mechanical integration, particularly the 3D printed mount and sensor integration. We did struggle with navigating coding environments and the serial port restrictions, but were ultimately successful. Our mounting of the breadboard and Arduino are also still focused on versatility and ease of modification, so given more time or a necessity for a more "polished" product, rethinking and refining that aspect of our design would likely be useful.

## 7.3 Learning

We recognize now that we did not explore all of our options before diving in. Instead of exploring methods of inputting via serial, we went directly to Python. This decision cost us hours of code generation and debugging. We realize that in the future we should reach out for guidance in such unfamiliar areas before increasing complexity so quickly.

We are glad that we did not overcomplicate the logic for our project. While proportional-integral-derivative (PID) control would have been an exciting exploration, we opted for a much simpler method, though it may be slower, and were still satisfied with our results.

We also did a relatively good job of not leaving our work until the eleventh hour. Although we did feel the pressure in the days before the due date, we had managed to pull together our robot's function earlier than we expected after our initial setbacks. By working relatively continuously, we were able to comfortably finish our complete implementation and data collection.

## 7.4 Future Directions

While our current control mechanism is functional, it could be interesting to implement PID control. Integration of a fourth IR sensor might also introduce course navigation options that were not a possibility with our design. Additionally, we currently use the serial port to change the threshold. Adding the ability to process multiple inputs and/or change the motor speeds would also be a promising next step.

# 8 Appendix

Arduino Source Code:

```
#include <Adafruit_MotorShield.h>
#include <stdio.h>
#include <stdlib.h>
int IR1PIN = A1; //analog input for left pin
int IR2PIN = A2; //analog input for center pin
int IR3PIN = A3; //analog input for right pin
int threshold = 500; //sensor threshold, informed by calibration
int lb; //T/F case variables for sensors
int rb; //T/F case for right sensor
int cb; //T/F case for center sensor
int cruise = 50; //default case speed settings
int slow = -50;
int catchup = 50;
int stop_speed = 0;
int ls = 40; //default speed settings/global vars for movementLogic
int rs = 40;

// Create the motor shield object with the default I2C address
Adafruit_MotorShield AFMS = Adafruit_MotorShield();
// Select which 'port' M1, M2, M3 or M4. In this case, M1
Adafruit_DCMotor *motorLeft = AFMS.getMotor(2);
// You can also make another motor on port M2
Adafruit_DCMotor *motorRight = AFMS.getMotor(1);
void setup() {
    Serial.begin(9600); // set up Serial library at 9600 bps
    pinMode(IR1PIN, INPUT); //set sensors to input
    pinMode(IR2PIN, INPUT);
    pinMode(IR3PIN, INPUT);
    if (!AFMS.begin()) {//create with the default frequency 1.6KHz
        // if (!AFMS.begin(1000)) {
            Serial.println("Could not find Motor Shield.");
            while (1);
        }
        //Serial.println("Motor Shield found.");
        motorLeft->setSpeed(ls); // Set default speed
        motorRight->setSpeed(rs);
        motorLeft->run(FORWARD); // turn on motors
        motorRight->run(FORWARD);
}

void loop() {
    recvWithStartEndMarkers();
    Serial.println(threshold);
    seesBlack();
    movementLogic(lb, rb, cb, threshold, cruise, slow, catchup);
    Serial.println("Made it!");
    motorLeft->run(RELEASE); //stop motors
    motorRight->run(RELEASE);

    if (ls > 0) {
        motorLeft->run(FORWARD);
    }
    else {
```

```
54          motorLeft->run(BACKWARD);
55      }
56      if (rs > 0) {
57          motorRight->run(FORWARD);
58      }
59      else {
60          motorRight->run(BACKWARD);
61      }
62
63      motorLeft->setSpeed(abs(ls));
64      motorRight->setSpeed(abs(rs));
65
66  }
67
68  int movementLogic(bool lb, bool rb, bool cb, int threshold,
69                    int cruise, int slow, int catchup) {
70
71      if (!lb && !rb) { //Case 3 (when cb sees black)
72          ls = cruise;
73          rs = cruise;
74          } else if (lb && !rb) { //Case 1
75          ls = slow;
76          rs = catchup;
77          } else if (!lb && rb) { //Case 2
78          ls = catchup;
79          rs = slow;
80          } else if (lb && rb) { //Case 5 (when cb sees black)
81          ls = stop_speed;
82          rs = stop_speed;
83          } else {
84          ls = stop_speed;
85          rs = stop_speed;
86          Serial.println("Edge Case (lb:"+ String(lb) + ", rb:" +
87          String(rb) + ", ls:" + String(ls) + ", rs" + String(rs));
88      }
89
90      if (!cb && !lb && !rb) { //Case 4
91          ls = cruise;
92          rs = -1 * cruise;
93      }
94
95  }
96
97  int seesBlack() {
98      int left_analog = analogRead(IR1PIN); //Get sensor values
99      int right_analog = analogRead(IR2PIN);
100     int center_analog = analogRead(IR3PIN);
101
102     lb = 0; //set to "False"
103     rb = 0;
104     cb = 0;
105
106     if (center_analog > threshold) { //sensor sees black
107         cb = 1; //set to "True"
108     }
109     if (left_analog > threshold) {
```

```
110         lb = 1;
111     }
112     if (right_analog > threshold) {
113         rb = 1;
114     }
115 
116     \\ Prints out the data in format to be easily copied into .csv
117 
118     Serial.print(center_analog); Serial.print(",");
119     Serial.print(right_analog); Serial.print(",");
120     Serial.print(ls); Serial.print(","); Serial.println(rs);
121 }
122 
123 void recvWithStartEndMarkers() {
124     static boolean recvInProgress = false; //local variables
125     static byte ndx = 0;
126     char startMarker = '<';
127     char endMarker = '>';
128     char rc;
129     String temp_thresh;
130     const byte numChars = 4;
131     char receivedChars[numChars];
132     char * strtokIndx;
133     boolean newData = false;
134     if (Serial.available() > 0) { //if port is available
135         while (Serial.available() > 0 && newData == false) {
136         //while there is new data
137             rc = Serial.read(); //get a char from buffer
138             if (recvInProgress == true) { //still more chars
139                 if (rc != endMarker) { //check for endline
140                     receivedChars[ndx] = rc; //add char to string
141                     ndx++;
142                     if (ndx >= numChars) {
143                         ndx = numChars - 1;
144                     }
145                 }
146                 else { //end of data input, reset
147                     receivedChars[ndx] = '\0'; //terminate string
148                     recvInProgress = false;
149                     ndx = 0;
150                     newData = true;
151                 }
152             }
153             else if (rc == startMarker) { //beginning of data
154                 recvInProgress = true;
155             }
156         }
157         if (rc == endMarker) { //end of data
158             \\ split at commas, though here there are none
159             strtokIndx = strtok(receivedChars, ",");
160             threshold = atoi(strtokIndx); //converts to integer
161         }
162     }
163 }
```

# 9 Bibliography

## References

[1] https://forum.arduino.cc/t/serial-input-basics/278284#a-more-complete-system-1