Lab 3 - Line Following Robot

Utsav Gupta and Aiden Carley-Clopton

October 12, 2018

Abstract

For this lab we created a Line-Following Robot using DC motors, Arduino motor shield and IR reflectance sensors. We assembled a standard robot chassis platform, mounted the Arduino on it and connected an Adafruit Motor Shield on top of the Arduino. Next, we mounted two IR Reflectance Sensors on a breadboard and wired them up to the Arduino. The breadboard was then mounted on the belly of the chassis using double-sided tape, and the IR sensors were programmed using a basic Differential Drive model. At this point, we tested our basic version of line-following robot and concluded that our Differential Drive model, while quite successful, was somewhat unpolished.

Hence we moved to a Proportional Feedback-based mechanism and saw that the results were much smoother. Finally, we programmed the Arduino to accept commands via Serial input, and created two *status* commands - *start* and *stop* - controlling the movement of the robot and three *speed* commands - *fast*, *medium* and *slow* - controlling the velocity of the robot.

Contents

1 The Setup

1.1 Arduino UNO

An Arduino UNO Rev 3 was used as an interface between the AtMega 328P microcontoller and the rest of the circuit. Since the Arduino uses a simpler version of Embedded C, called Arduino C, and has both digital and analog female headers, it was chosen for the ease of programming and circuit designing.

1.2 IR LED and Photo-Transistor Package

Two IR reflectance sensors (*Figure 1*) were used to gauge whether the robot was following the line. The sensor package contains an IR LED which emits infrared light and a photo-transistor which changes its behavior based on the amount of IR light that hits it. We wired the photo-transistor such that it formed a voltage divider with a 20 kΩ resistor, so that the output voltage varies depending on how much light is reflected by the surface beneath the sensor. We started with a 10 kΩ resistor, but noted that with a 20 kΩ resistor we got a lower number on the line compared to off of it, so we decided to use that value.

Figure 1: IR Sensor *TRCT5000*

We noted that the sensor reads a higher value over the reflective floor tiles than the black tape by having the value print to the Arduino serial monitor. Our controller uses the relative rather than absolute values of the two sensors, thus further calibration was not necessary. A plot of sensor values and motor commands can be seen in *Figure 2*.

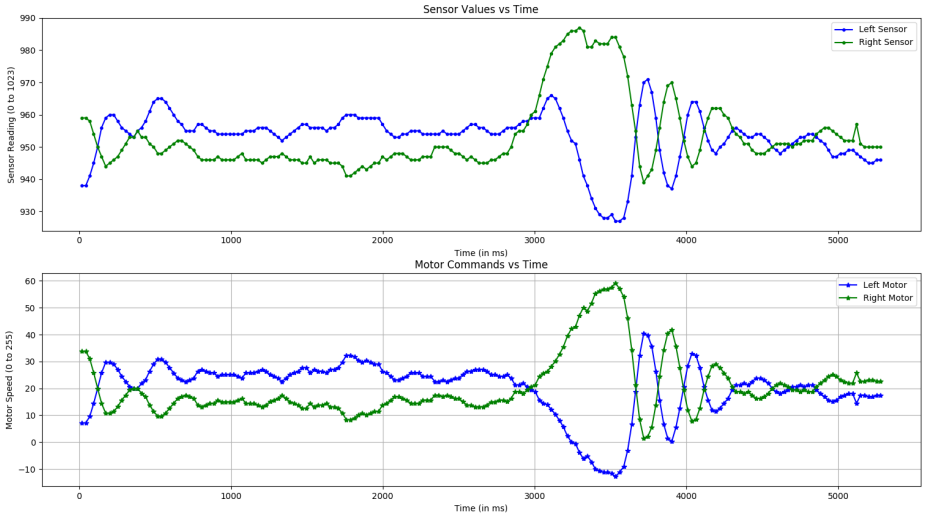


Figure 2: Sensor Data and Motor Speeds versus Time

1.3 Gear Motors

DC Gear Motors are a specific type of electrical motors designed to produce high torque while maintaining a low horsepower, or low speed, motor output. For the purposes of turning the wheels, we used Inline Gear Motors that accompanied the chassis. The motors are rated at 1:48 reduction ratio, which meant that at 5V, we had a high torque of $0.8kg/cm$ at the expense of relatively low speed of 44 meters/minute.

1.4 Adafruit Motor Shield

Motor Shields are PCBs designed to stack on top of microcontroller/microprocessor kits. They are used to power multiple Stepper Motors, Gear Motors or Servos, along with the microcontroller/microprocessor kits they are stacked on.

In order to drive our gear motors off the Arduino, we decided to stack an Adafruit Motor Shield (v2) (*Figure 3*) on the Arduino, and power the motors off it. The Motor Shield can power upto 4 bi-directional Gear Motors, and upto 2 Stepper Motors, from 5 V to 13.5 V power supplies at 1.2 Amps each.

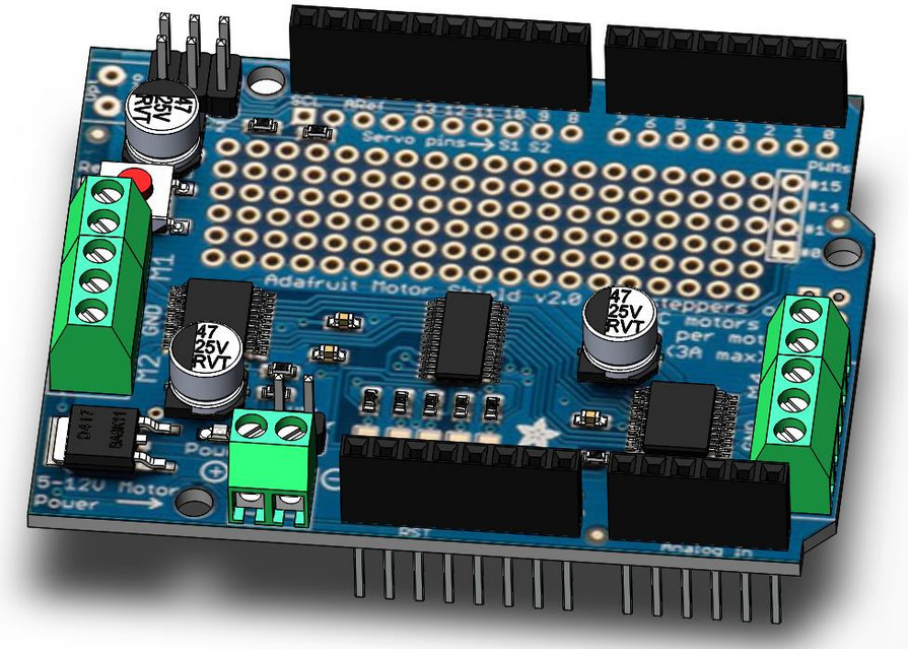


Figure 3: Adafruit Motor Shield

For this lab, we took a male barrel-jack to screw terminal connector, plugged the male barrel-jack terminal into the Arduino's female barrel-jack terminal and connected the screw terminal to a 5 V power supply side via 4-pin Molex wire-to-wire connector.

2 Hardware

2.1 Circuit

The circuit for this lab was relatively simple, with the only components being the IR reflectance sensor packages wired with resistors and the motors, as can be seen in *Figure 4*. The photo-transistor portion of the package forms a voltage divider with a 20kΩ resistor, such that the voltage read by the Arduino analogue input pins changes depending on how much IR light from the LEDs makes it to each of the sensors, which corresponds to the reflectance/absorbance of IR light by the surface the sensor is pointed at. The wiring for the motors is very simple, with one terminal going to a GND connection on the Adafruit Motor Shield, and

the other going to M3 or M4. Sending signals to the motors to drive them is then handled by the motor controller which receives instructions from the Arduino.

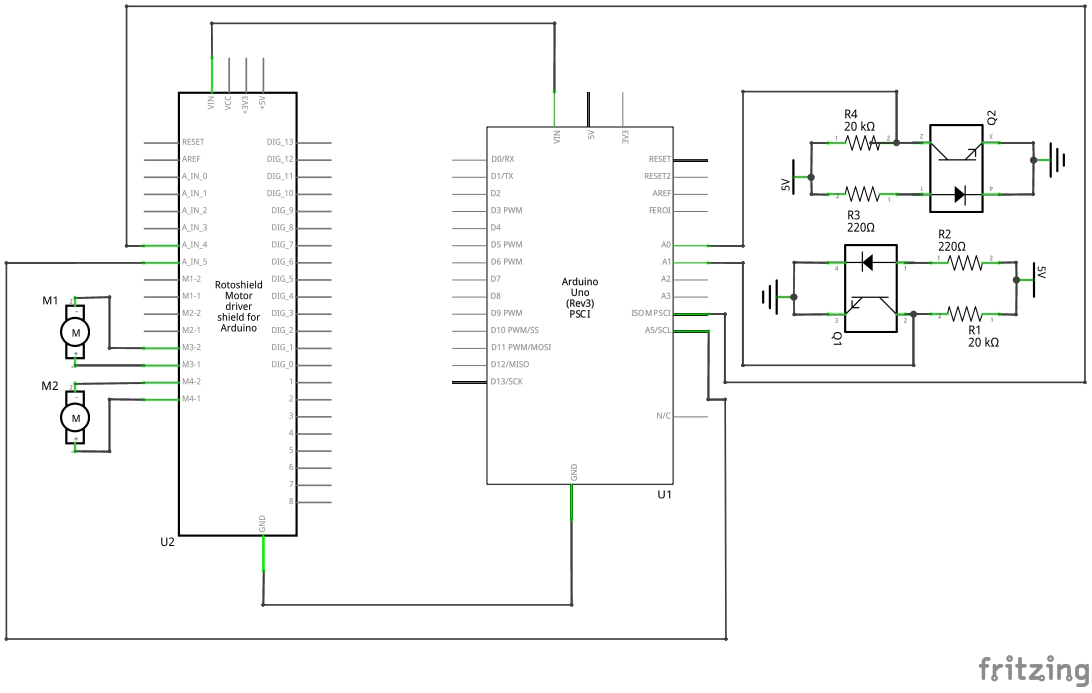
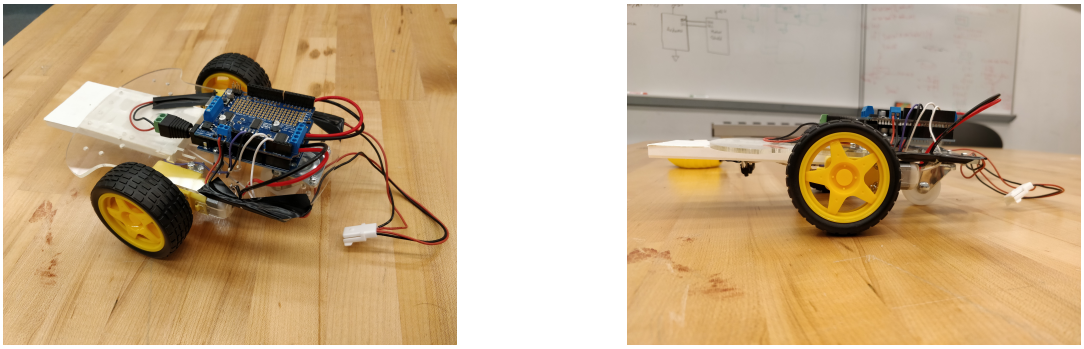


Figure 4: Schematic

2.2 Physical Setup

Our physical setup for this lab was very simple and can be seen in (*Figure 5*). We used holes already in the chassis plate which lined up with holes in the Arduino to mount it using bolts and plastic standoffs. The adhesive back of a breadboard was used to affix it to the bottom of the chassis, with the circuit assembled such that the sensors were on either side of the bread board. This left-right distance is important to allow for the sufficiently different sensor readings which are used by the control system.



(a) Robo

(b) Side View

Figure 5: Physical Setup

3 Software

3.1 Programming the Arduino

We used code on the Arduino to read in values from the sensors, drive the motors, and send sensor data and motor commands over the serial port to be plotted in Python. The code in *Figure 6* established the serial connection, as well as set the baud rate (the rate at which information will be transmitted through the serial port). We set a higher Baud rate so that the data would be sent and received at a higher speed. It also starts communication with the motor shield.

```
  Serial.begin(115200);
  AFMS.begin();   // start powering the motorshield
```

Figure 6: Serial/Motor Shield Setup

3.2 Driving the Motors

We drove the motors using the library associated with the Adafruit motor shield. Each motor was represented as an object and we used the function in (*Figure 7*) to run the motors. This function takes in a speed for the left and right motors, which can be positive or negative numbers. The speed is set by the absolute values of the inputs to the function, and the following conditional statements insure that the motors drive forward for positive values and backwards for negative values.

```
void run_robo(float left, float right) {
  // Function for running the motors
  left_motor -> setSpeed(abs(left));
  right_motor -> setSpeed(abs(right));

  if (left >= 0 && right >= 0) {
    left_motor -> run(FORWARD);
    right_motor -> run(FORWARD);
  }
  else if (left < 0 && right < 0) {
    left_motor -> run(BACKWARD);
    right_motor -> run(BACKWARD);
  }
  else if (left >= 0 && right < 0) {
    left_motor -> run(FORWARD);
    right_motor -> run(BACKWARD);
  }
  else if (left < 0 && right > 0) {
    left_motor -> run(BACKWARD);
    right_motor -> run(FORWARD);
  }
  else {
    Serial.println("Error in F/W & B/W loop, check < 499");
  }
}
```

Figure 7: Motor Driving Code

3.3 Line Following Controller

We decided to use the ratio of the sensor values to control the movement of the robot. The exact formula can be seen in *Figure 8*. The controller for a motor takes the ratio of the sensor on that side over the sensor on the other, cubes it, then subtracts 1. This produces a value which is high when the sensor on that side is off the line and the other is on it, close to zero when both sensors are close to the line, and negative when the sensor on that side is on the line and the other is off of it. This causes the wheel on the side off the line to turn forwards and the one on the line to turn backwards. The variable *sensitivity* determines to what extent this happens.

If this were the whole controller, the robot would rotate back and forth in place until it was on the line and stop. To make the robot move forward, the variable *factor* is added to both motor speeds to bias them towards driving forwards.

```
speedy_left = (pow(left_ir/right_ir, 3) - 1)*sensitivity + factor;
speedy_right = (pow(right_ir/left_ir, 3) - 1)*sensitivity + factor;
```

Figure 8: Controller

3.4 Taking Input from the Serial Monitor

In the final version of our line following code, the base speed of the robot (*factor*) could be chosen from one of three values based on input into the serial monitor. In addition, the robot could be started and stopped in a similar way. The code which checks for serial input can be seen in *Figure 9*. If the input is one of the understood strings, the speed and status (running or stopped) are set. Those values are then used to determine a value of *sensitivity* and *factor* which are later used to set the motor speeds.

```
// Conditions to check Serial input
  if (Serial.available() > 0) {
    input = Serial.readString();
    if (input.substring(0,-1) == "fast") {
      speed = "fast";
    }
    if (input.substring(0,-1) == "slow") {
      speed = "slow";
    }
    if (input.substring(0,-1) == "medium") {
      speed = "medium";
    }
    if (input.substring(0,-1) == "start") {
      status = "start";
    }
    if (input.substring(0,-1) == "stop") {
      status = "stop";
    }
  }
```

Figure 9: Parsing Serial Input

3.5 Sending Data to Python

We prepared a slightly modified version of the line following code to run for a short time and send all of the sensor values and motor commands over the serial port which was accomplished by the code in (*Figure 10*). The Arduino sends a time stamp, IR sensor readings, motor commands, and a boolean flag variable which

tells the python program to stop looking for data after the test is finished. The values are printed separated by commas with a new line at the end to enable the Python program to parse them properly.

```
Serial.print(time);
Serial.print(",");
Serial.print(left_ir);
Serial.print(",");
Serial.print(right_ir);
Serial.print(",");
Serial.print(speedy_left);
Serial.print(",");
Serial.print(speedy_right);
Serial.print(",");
Serial.println(flag);
run_robo(speedy_left, speedy_right);
```

Figure 10: Code to Send Data to Serial Port

3.6 Reading the data in Python

Once we had established Serial connection in Arduino, we started streaming the data to our laptops using the *PySerial* library in Python (*Figure 11*).

```
import serial

# Setting up the communication port
arduinoComPort = '/dev/ttyACM0'

baudRate = 115200
serialPort = serial.Serial(arduinoComPort, baudRate, timeout=1)
```

Figure 11: Reading data in Python using PySerial

The data received over Serial was separated by delimiters, in this case a comma character. Every line of data received was stored in a set of lists, with one for each part of the data, as can be seen in *Figure 12*.

```
while True:
  #
  # ask for a line of data from the serial port, the ".decode()" converts the
  # data from an "array of bytes", to a string
  #
    lineOfData = serialPort.readline().decode()
    print(lineOfData)

    #   Splits the values by ',' to read them separately
    data = lineOfData.split(',')

    #   Appends each piece of data to the appropriate list
    if len(data) == 6:
        time.append(float(data[0]))
        left_sensor.append(float(data[1]))
        right_sensor.append(float(data[2]))
        left_motor.append(float(data[3]))
        right_motor.append(float(data[4]))
        if(int(data[5]) == 1):
            break
```

Figure 12: Processing the data

At the end data transmission and processing, we stored the data in a text file as string representation of Python lists (*Figure 13*). The purpose behind this was to save our time and sanity by allowing us to record data once and test the processing multiple times without having to record again.

```
#   Creates a text file to write the raw data
speeds = open('data/test_data.txt', 'w')

#   Writes the data as Python lists
speeds.write(str(time) + '\n')
speeds.write(str(left_sensor) + '\n')
speeds.write(str(right_sensor) + '\n')
speeds.write(str(left_motor) + '\n')
speeds.write(str(right_motor) + '\n')
```

Figure 13: Writing the data to a text file

3.7 Processing and plotting the data in Python

After running the test and writing the results to a file, we created a Python program to read those values back into lists, process that data, and finally display it in plots. The code to read the data is shown in *Figure 14*. We used the *ast* module to convert the string representation of each list in the text file back into a Python list object.

```
#   Reading data from a .txt file
time = ast.literal_eval(file.readline())
left_sensor = ast.literal_eval(file.readline())
right_sensor = ast.literal_eval(file.readline())
left_motor = ast.literal_eval(file.readline())
right_motor = ast.literal_eval(file.readline())
```

Figure 14: Reading Data from Text File

After reading the data from the file, we plotted the sensor data and motor commands into two sub-plots, which can be seen in *Figure 2*.

4 Reflection

We found this lab to be a learning experience about implementing effective control systems. Early on, a coding error (doing integer instead of float division) caused our system to act in a binary way, only running one motor at a time. This worked, but was stuttering and slow, as seen in this video. We fixed the error, but then the robot would often run off the edge of the line. We gradually built up our final control system from the starting point of the sensor ratio, and it gradually became smoother and more robust. Parsing commands from the serial monitor also proved more difficult than expected, since the Arduino can't get the type of the data coming from the port. In the end, we worked out a system based on Proportional Feedback, which worked as well as seen in this video.

Appendix A Arduino code to control the Motors

```
#include <Keyboard.h>

// Include libraries
#include <Wire.h>
#include <Adafruit_MotorShield.h>
#include "utility/Adafruit_MS_PWMServoDriver.h"

// Defining pins
#define LEFT_SENSOR A0
#define RIGHT_SENSOR A1

// Defining variables
String speed = "medium";
String status = "stop";
float left_ir = 0;
float right_ir = 0;
float sensitivity = 200.0;
float factor = 20; // default speed = 20
float speedy_left = (pow(left_ir/right_ir, 3) - 1)*sensitivity + factor;
float speedy_right = (pow(right_ir/left_ir, 3) - 1)*sensitivity + factor;
String input = "";

// Creating motor object
Adafruit_MotorShield AFMS = Adafruit_MotorShield();

// getMotor(port#)  -->   look at the motor shield
Adafruit_DCMotor *left_motor = AFMS.getMotor(4);
Adafruit_DCMotor *right_motor = AFMS.getMotor(3);

void run_robo(float left, float right) {
  // Function for running the motors
  left_motor -> setSpeed(abs(left));
  right_motor -> setSpeed(abs(right));

  if (left >= 0 && right >= 0) {
    left_motor -> run(FORWARD);
    right_motor -> run(FORWARD);
  }
  else if (left < 0 && right < 0) {
    left_motor -> run(BACKWARD);
    right_motor -> run(BACKWARD);
  }
  else if (left >= 0 && right < 0) {
    left_motor -> run(FORWARD);
    right_motor -> run(BACKWARD);
  }
  else if (left < 0 && right > 0) {
    left_motor -> run(BACKWARD);
    right_motor -> run(FORWARD);Figure 4:  Serial Setup
  }
  else {
    Serial.println("Error in F/W & B/W loop, check < 499");
```

```
  }
}

void setup() {
  // Set analog voltage reference
  analogReference(DEFAULT);

  Serial.begin(115200);   // Set Baud rate
  AFMS.begin();   // Start powering the motorshield
}

void loop() {
  // Read sensor values
  left_ir = analogRead(LEFT_SENSOR);
  right_ir = analogRead(RIGHT_SENSOR);

  // Conditions to check Serial input
  if (Serial.available() > 0) {
    input = Serial.readString();
    if (input.substring(0,-1) == "fast") {
      speed = "fast";
    }
    if (input.substring(0,-1) == "slow") {
      speed = "slow";
    }
    if (input.substring(0,-1) == "medium") {
      speed = "medium";
    }
    if (input.substring(0,-1) == "start") {
      status = "start";
    }
    if (input.substring(0,-1) == "stop") {
      status = "stop";
    }
  }

  // Conditions to set speed
  if (speed == "fast") {
    sensitivity = 400;
    factor = 40;
    speedy_left = (pow(left_ir/right_ir, 3) - 1)*sensitivity + factor;
    speedy_right = (pow(right_ir/left_ir, 3) - 1)*sensitivity + factor;
  }
  else if (speed == "slow") {
    sensitivity = 200;
    factor = 20;
    speedy_left = (pow(left_ir/right_ir, 3) - 1)*sensitivity + factor;
    speedy_right = (pow(right_ir/left_ir, 3) - 1)*sensitivity + factor;
  }
  else if (speed == "medium") {
    sensitivity = 300;
    factor = 30;
    speedy_left = (pow(left_ir/right_ir, 3) - 1)*sensitivity + factor;
    speedy_right = (pow(right_ir/left_ir, 3) - 1)*sensitivity + factor;
```

```
  }
  else {
    Serial.println("Error occurred while setting speed..");
  }

  // Conditions to set status
  if (status == "start") {
    run_robo(speedy_left, speedy_right);
  }
  else if (status == "stop") {
    left_motor -> run(RELEASE);
    right_motor -> run(RELEASE);
  }
  else {
    Serial.println("Error occurred while setting status..");
  }

  // Wait 20ms before running the loop again
  delay(20);
}
```

Appendix B Arduino Code to Log Sensor Values

```
#include <Keyboard.h>

// Include libraries
#include <Wire.h>
#include <Adafruit_MotorShield.h>
#include "utility/Adafruit_MS_PWMServoDriver.h"

// Defining pins
#define LEFT_SENSOR A0
#define RIGHT_SENSOR A1

// Defining global variables
float left_ir = 0;
float right_ir = 0;
// default speed = 20
float speedy_left = ((left_ir/right_ir) - 1)*100.0 + 20;
float speedy_right = ((right_ir/left_ir) - 1)*100.0 + 20;
unsigned long time = 0;
bool flag = false;
int check = 0;

// Creating motor object
Adafruit_MotorShield AFMS = Adafruit_MotorShield();

// getMotor(port#)  -->   look at the motor shield
Adafruit_DCMotor *left_motor = AFMS.getMotor(4);
Adafruit_DCMotor *right_motor = AFMS.getMotor(3);

void run_robo(float left, float right) {
  left_motor -> setSpeed(abs(left));
  right_motor -> setSpeed(abs(right));

  if (left >= 0 && right >= 0) {
    left_motor -> run(FORWARD);
    right_motor -> run(FORWARD);
  }
  else if (left < 0 && right < 0) {
    left_motor -> run(BACKWARD);
    right_motor -> run(BACKWARD);
  }
  else if (left >= 0 && right < 0) {
    left_motor -> run(FORWARD);
    right_motor -> run(BACKWARD);
  }
  else if (left < 0 && right > 0) {
    left_motor -> run(BACKWARD);
    right_motor -> run(FORWARD);
  }
  else {
    Serial.println("Error in F/W & B/W loop, check < 499");
  }
}
```

```
void setup() {
  analogReference(DEFAULT);

  Serial.begin(115200);   // set Baud rate
  AFMS.begin();   // start powering the motorshield
}

void loop() {
  time = millis();

  left_ir = analogRead(LEFT_SENSOR);
  right_ir = analogRead(RIGHT_SENSOR);

  // conditions to set speed
  speedy_left = (pow(left_ir/right_ir, 3) - 1)*200.0 + 20;
  speedy_right = (pow(right_ir/left_ir, 3) - 1)*200.0 + 20;

  if (check < 199) {
    Serial.print(time);
    Serial.print(",");
    Serial.print(left_ir);
    Serial.print(",");
    Serial.print(right_ir);
    Serial.print(",");
    Serial.print(speedy_left);
    Serial.print(",");
    Serial.print(speedy_right);
    Serial.print(",");
    Serial.println(flag);
    run_robo(speedy_left, speedy_right);
  }
  else if (check == 199) {
    flag = true;

    Serial.print(time);
    Serial.print(",");
    Serial.print(left_ir);
    Serial.print(",");
    Serial.print(right_ir);
    Serial.print(",");
    Serial.print(speedy_left);
    Serial.print(",");
    Serial.print(speedy_right);
    Serial.print(",");
    Serial.println(flag);

    left_motor -> run(RELEASE);
    right_motor -> run(RELEASE);
  }
  else if (check == 200) {
    Serial.println("Successfully sent data..");
  }
```

```
  delay(20);

  check += 1;
}
```

Appendix C Python code to communicate with the Arduino

```
#   Uses PySerial to fetch data from the Arduino and writes it to a
#   text file in a string format.

#   Importing library
import serial

#   Set the communication port
arduinoComPort = '/dev/ttyUSB0'

#   Create a Serial port object
baudRate = 115200
serialPort = serial.Serial(arduinoComPort, baudRate, timeout=1)

time = []
left_sensor = []
right_sensor = []
left_motor = []
right_motor = []

while True:
  #
  # ask for a line of data from the serial port, the ".decode()" converts the
  # data from an "array of bytes", to a string
  #
    lineOfData = serialPort.readline().decode()
    print(lineOfData)

    #   Splits the values by ',' to read them separately
    data = lineOfData.split(',')

    #   Appends each piece of data to the appropriate list
    if len(data) == 6:
        time.append(float(data[0]))
        left_sensor.append(float(data[1]))
        right_sensor.append(float(data[2]))
        left_motor.append(float(data[3]))
        right_motor.append(float(data[4]))
        if(int(data[5]) == 1):
            break

#   Creates a text file to write the raw data
speeds = open('data/test_data.txt', 'w')

#   Writes the data as Python lists
speeds.write(str(time) + '\n')
speeds.write(str(left_sensor) + '\n')
```

```
speeds.write(str(right_sensor) + '\n')
speeds.write(str(left_motor) + '\n')
speeds.write(str(right_motor) + '\n')
```

Appendix D Python code to plot Sensor Readings and Wheel Speed

```
#   Uses ast to read data from a file and matplotlib to
#   generate subplots

#   Importing the libraries
import matplotlib.pyplot as plt
import ast

#   Open the file in read mode
file_name = 'data/test_data.txt'
file = open(file_name, mode='r')

#   Reading data from a txt file
time = ast.literal_eval(file.readline())
left_sensor = ast.literal_eval(file.readline())
right_sensor = ast.literal_eval(file.readline())
left_motor = ast.literal_eval(file.readline())
right_motor = ast.literal_eval(file.readline())

#   Plot the data as function of time
fig, axs = plt.subplots(2, 1)
axs[0].plot(time, left_sensor, label = 'Left Sensor', c = 'b', marker = '.')
axs[0].plot(time, right_sensor, label = 'Right Sensor', c = 'g', marker = '.')
axs[1].plot(time, left_motor, label = 'Left Motor', c = 'b', marker = '*')
axs[1].plot(time, right_motor, label = 'Right Motor', c = 'g', marker = '*')
axs[0].set_xlabel('Time (in ms)')
axs[0].set_ylabel('Sensor Reading (0 to 1023)')
axs[1].set_xlabel('Time (in ms)')
axs[1].set_ylabel('Motor Speed (0 to 255)')
axs[0].set_title("Sensor Values vs Time")
axs[1].set_title("Motor Commands vs Time")
axs[0].legend()
axs[1].legend()
plt.grid(True)
plt.show()
```