Line Following Robot

Mark Goldwater and Erika Serna

October 12, 2018

1 Introduction

In this lab, we were tasked with creating a small wheeled robot that would follow a black line on the floor of the PoE lab using a closed-loop control system. In order to accomplish this task, we were given two IR reflectance sensors, a motor shield, a 12 V power supply, an acrylic chassis, two DC motors (in a gearbox assembly), and some wire to connect the motors as well as provide power to the boards.

CLICK HERE TO SEE A VIDEO OF OUR LINE FOLLOWING ROBOT.

(link also here: https://youtu.be/DFVtTAh9q0w).

2 Constructing the Wiggle Bot

2.1 Mechanical

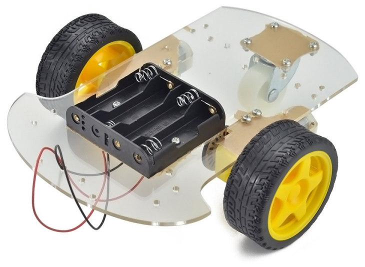

Figure 1: This is a picture of the mechanical setup only that comes with the car kit. It was taken from the car's amazon page. Note that we did not use the battery holder that came with the kit but rather a 12 V power supply.

In order to construct the mechanical aspects of our car, we first followed the instructions in the car kit to build the basic chassis and motor setup. We screwed the motors onto the underside of the chassis, slipped the wheels on, and then screwed on the third small wheel which provided support to one side of the vehicle since the car's main wheels are centered on the chassis.

Next, to mount the Arduino and motor controller we were able to use the base that our Arduino Uno R3 came with. This small plastic base has an indent to hold the Arduino mount securely as well as screw holes on two of the corners that happen to line themselves with screw holes that exist in the chassis. After our Arduino was secured on our robot, we obtained a small breadboard and set that on our chassis directly in front of the Arduino (We go into more detail regarding electronics in subsection 2.2.

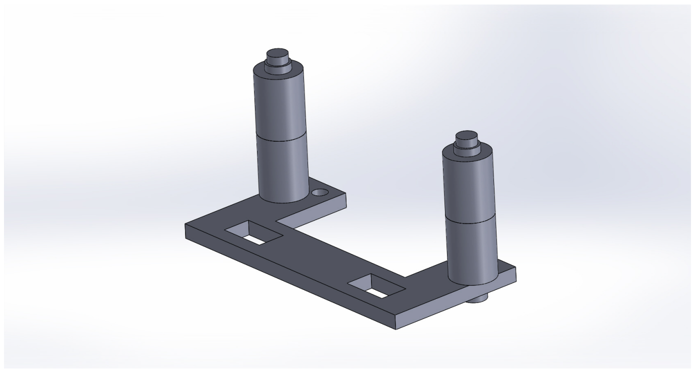

Figure 2: This is a rendering of the 3D printed part and spacers we used to mount the sensors onto the car.

In the end, we decided to mount the IR sensors about 0.5 cm off the ground in the front of the robot. In order to achieve this, we 3D printed a flat piece which had holes to press fit the sensors and then used spacers so that we could screw it in to the acrylic chassis of the car and have the sensors be positioned close to the ground. This setup can be seen in Figure 2. The spacers are the cylinders stacked on top of the flat piece which have screw going though them, and the two rectangular holes towards the front of the flat piece are for the sensors.

2.2 Electrical

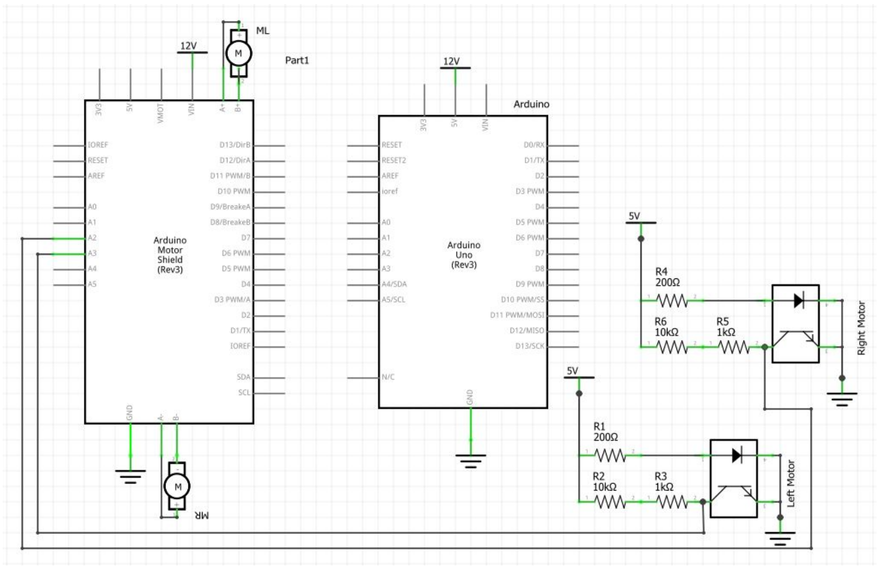

Figure 3: This is a schematic of our electrical setup. The Motor Shield is stacked onto the Arduino so the items attached to the communication pins on the Motor Shield are also connected to the Arduino. As shown in the figure, there are two motors and two IR sensors connected.

To make our full circuit we used an Arduino, an Adafruit Motor Shield, and a small breadboard. The Motor Shield is stacked onto the Arduino and powered through a Molex Wire connection.

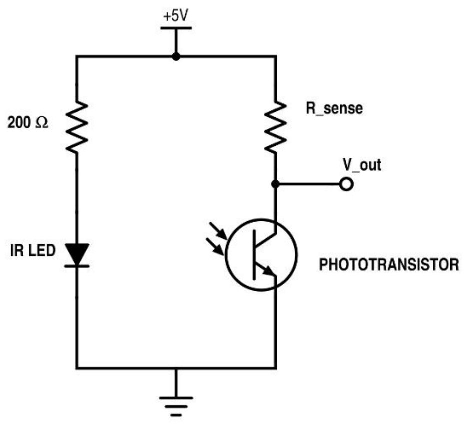

Figure 4: This is a schematic of the circuit that we used for each of our IR diodes and phototransistors.

In order to have our robot follow the line efficiently, we wanted our sensor to read a slow change in values as it swept from the floor to the line or vice versa. In order to do this, we started with one sensor on a bread board (using the circuit from Figure 4) which we. When we did that we had decided through trial and error that a 50 kΩ resistor at a height of about 3.1 cm would work best to give us a wide range of sensor values for the sensor being on and off the line. With these specifications, we got a range of about 100 in the sensor values. The sensor values were read by an analog pin in the Arduino board connected to the V_{out} that is labeled in the schematic in Figure 4.

Following that experiment we built our robot and tested it with the control software that we wrote (Section 4). At this point our robot worked fairly well, but we knew that we could improve the sensitivity of the data we collect from the IR sensors if we made them closer to the ground, for this would leave less room for error by giving the IR light a small distance to travel between its emitted light from the LED and reading by the phototransistor. Moreover, the data sheet for the sensor mentioned that its tests were very close to the surface it was measuring the values of which further supports our decision to lower the sensors. The mechanical apparatus we used to lower the sensors is described in the previous section (Section 2.1).

2.3 Final Car

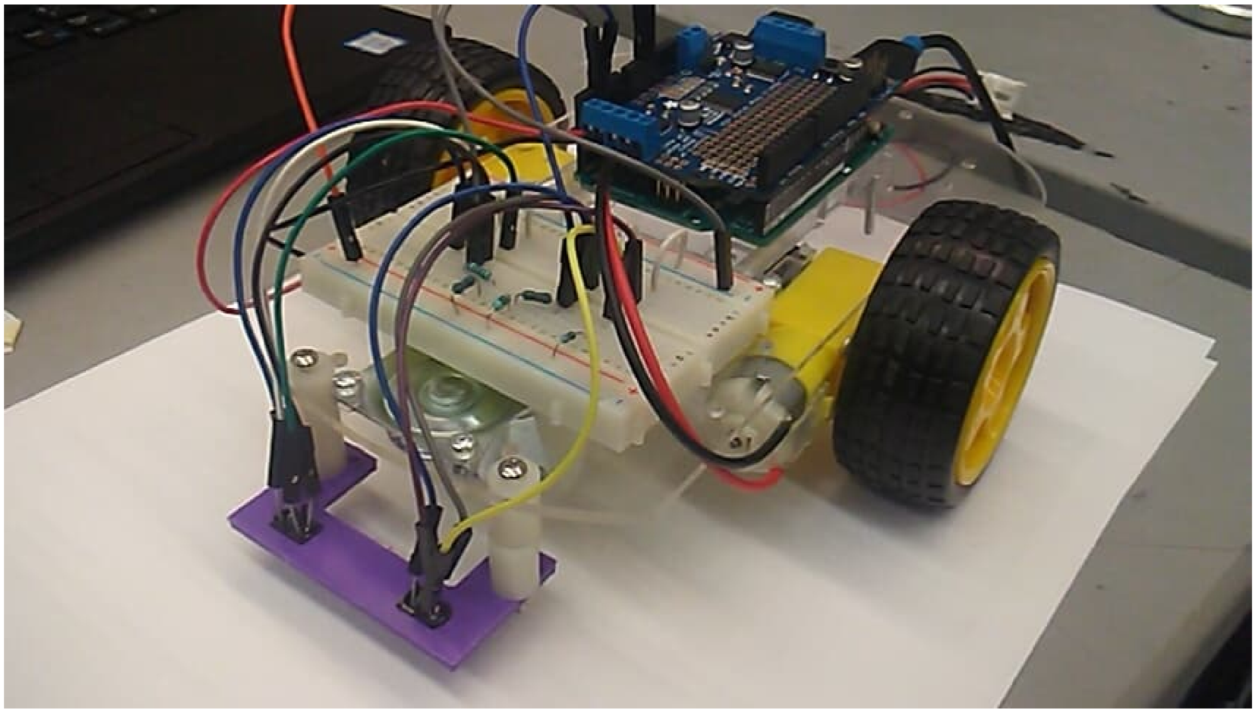

Figure 5: Through teamwork, we were able to incorporate parts we already had, like the 3D printed piece Mark had built in SolidWorks and some spacers Serna grabbed, to construct our robot and focus on the code. We also saved ourselves the trouble of soldering by using male-to-female connectors for the sensors.

To make all of the electrical connections in our car, we used male-to-female and male-to-male jumper wires. Moreover, all of our mechanical parts were attached with screws and we saved the time of redoing our sensor holder by just adding physical spacers instead of having to edit the CAD and reprint. A final picture of our car can be seen in Figure 5.

3 Calibration of IR Sensors

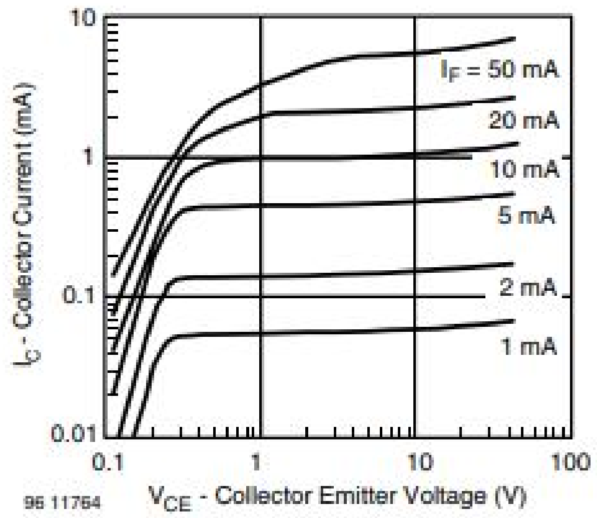

Figure 6: This graph shows the collector emitter saturation voltage vs. the collector current. Ideally we want our circuit to act in the range which makes it respond linearly to the appropriate range of input data (i.e. the floor and black electrical tape)

The graph in Figure 6 shows that for a given forward current through the IR LED, different collector current values (which are caused by different amounts of IR light that the phototransistor receives) will either cause the voltage to change noticeably, producing a large and variable range of sensor values, or will be saturated and not alter the collector emitter voltage in a significant way. In order to find this value we decided to go through the process of an intelligent trail and error.

The logic we used to pick a resistor value to start with was that the resistor **R_sense** in Figure 4 and the phototransistor made a voltage divider. In order for significant changes in the effective resistance (NOT actual resistance) of the phototransistor to make a considerable change to V_{out}, **R_sense** must be larger compared to the effective resistance of the phototransistor. This conclusion was made from the voltage divider equation $V_{out} = V_{in}\frac{R_2}{R_1+R_2}$ where R_1 is **R_sense** and R_2 is the effective resistance of the phototransistor.

When our robot's sensors were at about 3.1 cm above the ground, we found that the optimal value for **R_sense**, through trial and error, was 50 kΩ. This gave us a sensor value range of about 100. However, when we lowered the sensors to about 0.5 cm above the ground we found that a resistor value of 11 kΩ created a sensor range of 400, which was a significant improvement and what we stuck with for the remainder of the project.

4 Control Method

In order to control the robot, we decided to use the ratio of the left sensor to the right sensor and the ratio of the right to the left. The basic control code can be seen below. Note that **c** is a global variable and is a constant that determines how fast the robot traverses the path.

4.1 Basic Movement

```
void loop()
{
  read_serial();

  // read left and right sensor values
  val_r = analogRead(rightSensor);
  val_l = analogRead(leftSensor);

  leftspeed = (val_l/val_r)*c;
  rightspeed = (val_r/val_l)*c;

  // if calculated speed are about the same go straight
  if (abs(leftspeed - rightspeed) < 10)
  {
    // keep speeds the same. Below code not necessary, but good to demonstrate flow of the pro
    leftspeed = leftspeed;
    rightspeed = rightspeed;
  }
  // if left speed is really small make it smaller to make turn sharper
  else if (leftspeed < rightspeed)
  {
    leftspeed = leftspeed / 2.0;
  }
  // same for the other wheel
  else if (leftspeed > rightspeed)
  {
    rightspeed = rightspeed / 2.0;
  }

  // send speeds to wheels
  myLeftMotor->setSpeed((int) leftspeed);
  myRightMotor->setSpeed((int) rightspeed);
}
```

For now, let's ignore the function **read_serial()**, for it is discussed in the following subsection (4.2). Essentially what we do here is read the left and right sensor values and assign the speeds in the following way: **leftspeed = (val_l/val_r)*c** and **rightspeed = (val_r/val_l)*c**. In order to understand why we chose to assign speeds this way consider this example: the robot begins to stray away from the line by turning off it towards the right. As the right sensor begins to read the floor, its sensor reading would decrease and the left sensor value would increase as it slides above the line. Using the equations we established, **leftspeed** in this scenario would decrease dramatically and **rightspeed** would increase dramatically.

In addition to this logic, if the absolute value of the difference between the two speeds is greater than 10, we cut the smaller speed in half in order to make the turn more aggressive and keep the robot on track. This helped a lot especially regarding the sharp turns. Otherwise, we kept the speeds at the same values as originally calculated, for they should be relatively equal already.

4.2 Changing Motion Via Serial Connection

The purpose of the function **read_serial()** is to change the speed of the robot based off of user input in the serial monitor. While the robot is in operation (and as long is the robot is connected to the computer through USB, this function (defined below) will work).

```
void read_serial()
{
  // check if anything new is in serial monitor
  if (Serial.available() > 0)
  {
    // read the incoming:
    incoming = Serial.readString();

    // go through options of what the input could be and
    // respond appropriately
    if (incoming == "fast")
    {
      c = 25;
      Serial.print("Your speed constant is now equal to: ");
      Serial.println(c);
    }
    else if (incoming == "slow")
    {
      c = 20;
      Serial.print("Your speed constant is now equal to: ");
      Serial.println(c);
    }
    else
    {
      Serial.println("That command is invalid. Please enter 'fast' or 'slow'");
    }
  }
}
```

This is called with every run of the **loop()** function. It first checks if there is anything inputted in the serial monitor by checking the amount of bytes available by reading the serial port. If there is something, it then takes in that string and checks if it is ”fast”, ”slow”, or anything else. If it is ”fast” or ”slow” the speed constant (c) is changed appropriately and the user is notified. If anything else is inputted, the user is told that their input was invalid and given commands that will work.

5 Data Collection

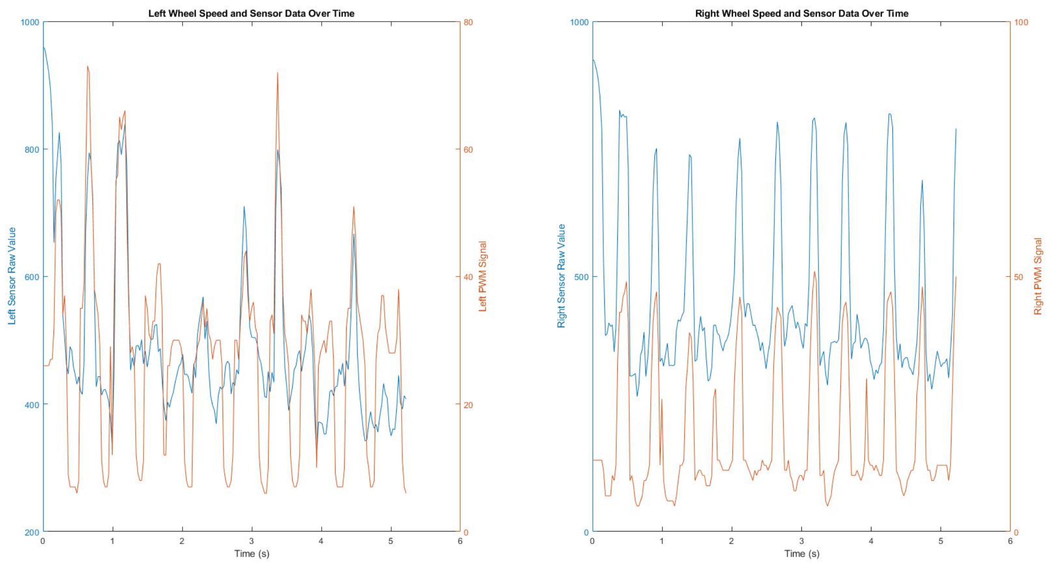

Figure 7: These graphs show the relative raw sensor data from the left and right IR sensor and the PWM signal that was sent to the motors.

In order to collect data we used a python script to read from the serial monitor and turn the collected data into a .csv file which we graphed with MATLAB. We decided to collect data from the robot for a period of ten seconds. In other words, we printed ten seconds worth of sensor data and wheel speeds to the serial monitor with the following code where **float start = millis()** was called in the **void setup()** function.

```
if (millis() - start <= 10000.0)
  {
    Serial.print(millis());            Serial.print(",");
    Serial.print(val_l);               Serial.print(",");
    Serial.print(val_r);               Serial.print(",");
    Serial.print((int) leftspeed);     Serial.print(",");
    Serial.println((int) rightspeed);
  }
  else
  {
    Serial.print(-1);               Serial.print(",");
    Serial.print(-1);               Serial.print(",");
    Serial.print(-1);               Serial.print(",");
    Serial.print(-1);               Serial.print(",");
    Serial.println(-1);
  }
```

This code would print our desired data to the serial monitor while **millis() - start is less than 10000.** (while ten seconds had not yet passed). While this condition is true, the code would print, on each line, the current time (in ms), the left and right sensor values, and the left and right speed all separated by commas. After ten seconds, it would start printing the number -1 in place of all the data which is how our python script knew to stop recording and save the .csv.

The core of the Python code we used to collect the data from the Arduino serial monitor can be seen below.

```
while True:

    # The code that goes here is shown in the appendix

    time, left_val, right_val, leftspeed, rightspeed = (x for x in lineOfData.split(','))

        if (str(left_val) == "-1"):
          break

        t_list.append(time)
        lv_list.append(left_val)
        rv_list.append(right_val)
        ls_list.append(leftspeed)
        rs_list.append(rightspeed)

        print(lv_list)

with open('data1.csv', 'w', newline='') as csvfile:
  fieldnames = ['time','left_val', 'right_val', 'leftspeed', 'rightspeed']
  writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

  writer.writeheader()
  for i in range(len(lv_list)):
    writer.writerow({'time': t_list[i], 'left_val': lv_list[i], ...
    'right_val': rv_list[i], 'leftspeed': ls_list[i], 'rightspeed': rs_list[i]})
```

What this code does is it reads the current line from the serial monitor, assigns the values to their proper names based on what the data is. After these variables are assigned, it checks if the first one is -1 (which it should never be unless the data is done being collected and we are intentionally printing this value). Then each data value is added to its respective list. At the point when -1 is printed, the while loop breaks and the bottom section of this code sets up a csv file by establishing its columns and then iterates through the lists of data, placing them into their appropriate columns. The csv file is saved in the same directory as this code.

Lastly, we wrote a very short MATLAB script to take the csv file and graph the data (Figure 7). The code is shown below.

```
% data for the left wheel
subplot(1,2,1)
x = time(1:200)/1000;
y1 = left_val(1:200);
y2 = leftspeed(1:200);
[hAx,hLine1,hLine2] = plotyy(x,y1,x,y2);
```

```
title('LEft Wheel Speed and Sensor Data Over Time')
xlabel('Time (s)')

ylabel(hAx(2),'Left PWM Signal')
ylabel(hAx(1),'Left Sensor Raw Value')

% data for the right wheel
subplot(1,2,2)
x = time(1:200)/1000;
y1 = right_val(1:200);
y2 = rightspeed(1:200);
[hAx1,hLine3,hLine4] = plotyy(x,y1,x,y2);

title('Right Wheel Speed and Sensor Data Over Time')
xlabel('Time (s)')

ylabel(hAx1(2),'Right PWM Signal')
ylabel(hAx1(1),'Right Sensor Raw Value')
```

In this code we simply create a subplot (one plot for the left and right wheel data) that plots the first 200 data points of our collected data for each wheel.

In these graphs one can see that when the raw sensor value of one wheel is very low, its speed dips down. This is because of the halving of the slower wheel we implemented in order for the robot to be able to handle the sharper turns on the track. In general, one of the issues with our robot was that one motor went at a different speed than the other when they were given the same PWM signal. This can be seen pretty clearly in the dampening of the PWM signal in the right graph in Figure 7. Moreover, the wheels had some trouble gripping onto the floor. Through looking at this data, one can see the relationship between the raw data and PWM signal for the left and right motor/sensor pair.

6 Reflection

Coming on to this project, we decided to be partners again because our strengths complemented each other. Mark was able to help Serna with coding, Serna was able to help Mark with SolidWorks, and together we worked on the electronics.

One of the goals for our project was to front-load the work in the beginning so that we could learn as much as possible by having the person with weaker skills in an area ”do” and the person with stronger skills ”guide” without worrying about finishing on time. This method worked extremely well and our weaker skills have improved significantly.

At the end of this project, we both felt tired of the project because we spent so much time on it. Thankfully because we had more time for this project, even though we both got sick, we were still able to stick to our goal of learning in our weaker skill.

We had two main struggles with this project: choosing the proper resistor and following the robot with the cables.

Regarding the resistor problem, we both knew what we wanted out our of values but because we keep changing the height of the IR sensor, we spent a lot time going back and forth.

With the cable problem, we just kept accidentally dragging our robot with the power cord and the Arduino cable. This proved to be more of a frustration than an actual problem, especially when we were trying to record our video and collect data. For our video, you can see that we intentionally had someone ”walk” the robot.

Overall, we can conclude that our work styles worked nicely together, and we both got a lot out of this lab.

7 Appendix

7.1 Arduino Control Code Source Code

```
#include <Wire.h>
#include <Adafruit_MotorShield.h>

// 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 *myRightMotor = AFMS.getMotor(3);
// You can also make another motor on port M2
Adafruit_DCMotor *myLeftMotor = AFMS.getMotor(4);

const int rightSensor = 2;
const int leftSensor = 3;

double val_r = 0;
double val_l = 0;

double c = 26.0;

double leftspeed;
double rightspeed;

String incoming;

float start = millis();

void setup()
{
  Serial.begin(9600);
  AFMS.begin();  // create with the default frequency 1.6KHz

  myRightMotor->run(FORWARD);
  myLeftMotor->run(FORWARD);
  Serial.print(start);
}
```

```
void loop()
{
  read_serial();
  //SENSORS SENSORS SENSORS
  val_r = analogRead(rightSensor);
  val_l = analogRead(leftSensor);
  //Serial.print("Right Sensor:"); Serial.print(val_r);
  //Serial.print("Left Sensor:"); Serial.println(val_l);

  leftspeed = (val_l/val_r)*c;
  rightspeed = (val_r/val_l)*c;

  if (abs(leftspeed - rightspeed) < 3)
  {
    // keep speeds the same. Below code not necessary, but good to demonstrate flow of the pro
    leftspeed = leftspeed;
    rightspeed = rightspeed*1.16;
  }
  if (leftspeed < rightspeed)
  {
    leftspeed = leftspeed / 2.0;
  }
  else if (leftspeed > rightspeed)
  {
    rightspeed = rightspeed*1.16 / 2.0;
  }

  if (millis() - start <= 10000.0)
  {
    Serial.print(millis());           Serial.print(",");
    Serial.print(val_l);              Serial.print(",");
    Serial.print(val_r);              Serial.print(",");
    Serial.print((int) leftspeed);    Serial.print(",");
    Serial.println((int) rightspeed);
  }
  else
  {
    Serial.print(-1);               Serial.print(",");
    Serial.print(-1);               Serial.print(",");
    Serial.print(-1);               Serial.print(",");
    Serial.print(-1);               Serial.print(",");
    Serial.println(-1);
  }
  myLeftMotor->setSpeed((int) leftspeed);
  myRightMotor->setSpeed((int) rightspeed);
}

void read_serial()
```

```
{
  if (Serial.available() > 0)
  {
    // read the incoming:
    incoming = Serial.readString();
    if (incoming == "fast")
    {
      c = 25;
      Serial.print("Your speed constant is now equal to: ");
      Serial.println(c);
    }
    else if (incoming == "slow")
    {
      c = 20;
      Serial.print("Your speed constant is now equal to: ");
      Serial.println(c);
    }
    else
    {
      Serial.println("That command is invalid. Please enter 'fast' or 'slow'");
    }
  }
}
```

7.2 Python Data Collection Source Code

```
import serial
import csv
import time

arduinoComPort = '/dev/ttyACM0'

baudRate = 9600

t_list, lv_list, rv_list, ls_list, rs_list = [], [], [], [], []

# open the serial port
serialPort = serial.Serial(arduinoComPort, baudRate, timeout=1)

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()
  #
  # check if data was received
  #
  if len(lineOfData) > 0:
    #
    # data was received, convert it into 4 integers
    #
    time, left_val, right_val, leftspeed, rightspeed = (x for x in lineOfData.split(','))

    if (str(left_val) == "-1"):
      break

    t_list.append(time)
    lv_list.append(left_val)
    rv_list.append(right_val)
    ls_list.append(leftspeed)
    rs_list.append(rightspeed)

    print(lv_list)

with open('data1.csv', 'w', newline='') as csvfile:
  fieldnames = ['time','left_val', 'right_val', 'leftspeed', 'rightspeed']
  writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

  writer.writeheader()
  for i in range(len(lv_list)):
    writer.writerow({'time': t_list[i], 'left_val': lv_list[i], ...
    'right_val': rv_list[i], 'leftspeed': ls_list[i], 'rightspeed': rs_list[i]})
```

7.3 MATLAB Plotting Source Code

```
x = time(1:200)/1000;
y1 = left_val(1:200);
y2 = leftspeed(1:200);
[hAx,hLine1,hLine2] = plotyy(x,y1,x,y2);

title('LEft Wheel Speed and Sensor Data Over Time')
xlabel('Time (s)')

ylabel(hAx(2),'Left PWM Signal')
ylabel(hAx(1),'Left Sensor Raw Value')

% data for the right wheel
subplot(1,2,2)
x = time(1:200)/1000;
y1 = right_val(1:200);
y2 = rightspeed(1:200);
[hAx1,hLine3,hLine4] = plotyy(x,y1,x,y2);

title('Right Wheel Speed and Sensor Data Over Time')
xlabel('Time (s)')
```

```
ylabel(hAx1(2),'Right PWM Signal')
ylabel(hAx1(1),'Right Sensor Raw Value')
```