Principles of Engineering: Lab 3

Nick Steelman and Gracey Wilson

October 13, 2017

1 Background and Materials

In this lab, we used DC motor control and infrared sensors to make a robot travel along a line of tape over tile.

Our tasks were as follows:

- Design and build a circuit that sends data from IR sensors to computer
- Mount wheels, DC motors and IR sensors to chassis
- Write a closed-loop controller which enables robot to follow line of electrical tape over tile
- Visualize sensor and motor output as robot moves over tape and other surfaces

The materials we used included:

- 1 Arduino
- 1 Adafruit motorshield
- 2 infrared distance sensors
- 2 DC motors
- 3 wheels

2 Circuit Decisions and Diagram

The circuit for this lab is quite simple. As shown in Figure 2, it consists of 2 infrared sensors attached to analog inputs on the Arduino with some resistance for filtering, as well as 2 DC motors attached to digital inputs on the Motor Shield. All sensors are connected to ground and 5V as well. All motors are connected to ground and the motor's input voltage.

In order for the IR sensor to work, we needed to calculate the resistance needed in our circuit. From the IR sensor's data sheet, we learned that the typical current is 1mA. We also learned that the typical voltage drop across the sensor is 1V using this graph found on the data sheet.

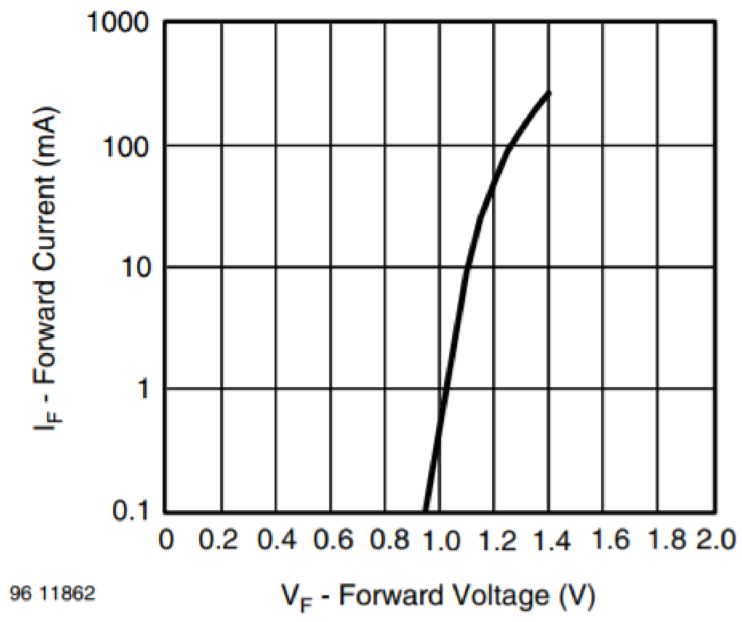

Figure 1: Forward current versus forward voltage.

From there we were able to calculate the resistor value needed:

$$V = IR \tag{1}$$
$$5V - 1V = (1mA) * R \tag{2}$$
$$R = 4V/0.001A \tag{3}$$
$$R = 4000\Omega \tag{4}$$

We started off with this number and found a decent fluctuation of the voltage when testing it with objects of varying distance, but it was not a very large range. So, we explored different similar resistance values until we found a value that gave a reasonably high range of values when looking at tile versus black tape. We ended up with a 10k ohm resister that gave us an output range from 400 to 900. Our final circuit diagram is shown in Figure 2.

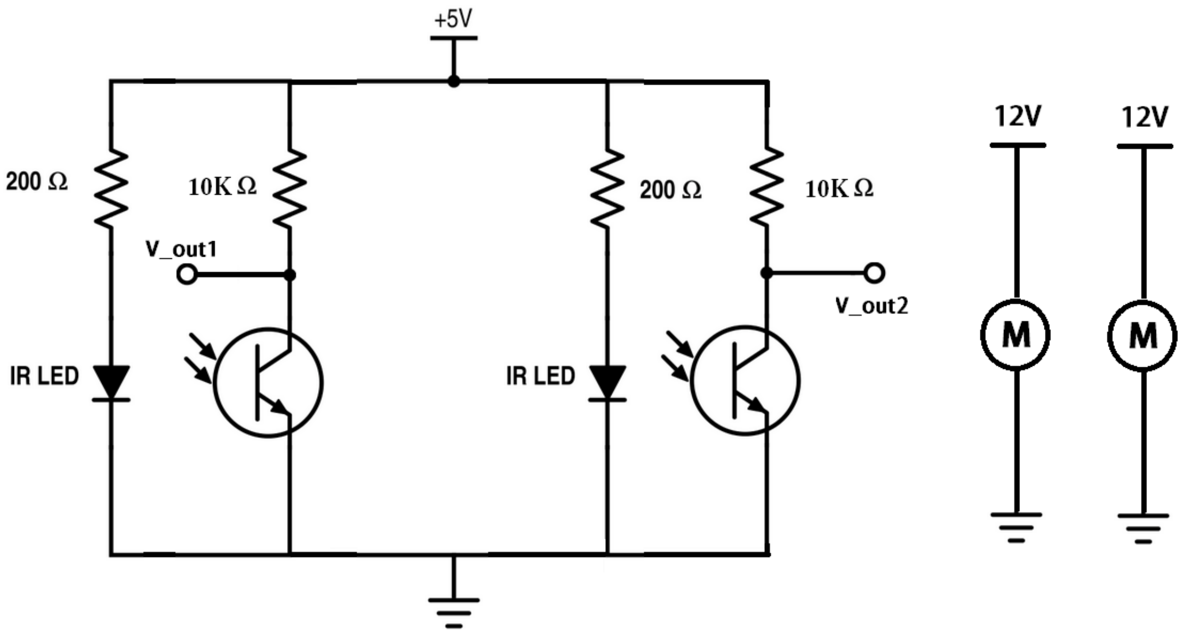

Figure 2: Circuit diagram showing the IR sensors with resistance (left) and DC motors (right).

3 Calibrating the Sensors

In the beginning we tested our IR sensors by running a test program that printed the sensor values to serial monitor. We pointed the sensors at various surfaces and objects at different distances. As described in Section 2, we adjusted the resistor values several times in the search for a clear difference in output between tape and tile. We were eventually satisfied when we achieved an output range from about 400 to 900 with resistors of 10K Ω. We then decided on a threshold of 600 which would reasonably split the minimum and maximum values.

We also plotted the sensor and motor values to visualize our calibration, shown in Figure 5. You can see the correlation between left sensor values and right motor movement, as well as right sensor values and left motor movement.

4 Mount

In order to connect the wheels, DC motors and chassis, we used holes in the chassis and 2 screws on each side.

Mounting the circuit to the robot was more of an iterative process. First, we soldered the circuit to a mini breadboard to make it more portable and easier to connect to the chassis. Also, we knew we wanted the sensors to be fairly close to the ground for the most accurate readings possible. We chose to use cardboard and tape to build a fairly rudimentary structure that holds it extend it closer to the ground. This is because both our learning goals related more to code and circuitry than to mechanical design, so we put more time into those design elements.

Then we used some cardboard and tape to mount the mini breadboard to the underbelly of the chassis around the middle of the chassis, between the wheels. However with a little bit of testing, we realized the error of this approach. When the sensors were in line with the wheels and the wheels moved, the sensors were mostly moved laterally along the line of tape. What we wanted was for the sensors to be moved off of the tape so that they could more accurately tell which motor to move in order to continue tracing the line of tape. With that goal in mind, we moved the breadboard to the front of the chassis. A photo of the final mount is shown in Figure 3.

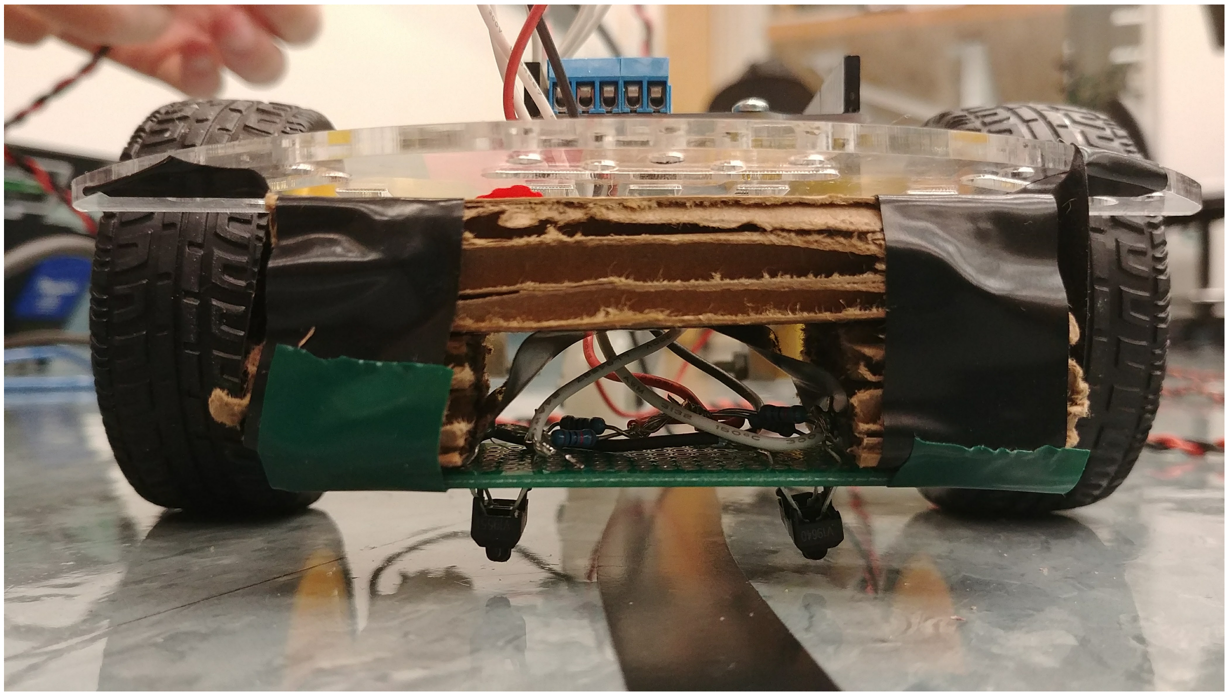

Figure 3: Photo of the chassis from the front, with the circuit mounted underneath.

5 Source Code and Visualization

5.1 Arduino Controller

In order to get the robot to follow the line, we employed a simple "Bang Bang" feedback loop that will control wheel speed directly based on the reading of the sensor. So, while the left sensor reads that it is over a line, the left wheel will stop and the right wheel will continue at a predefined speed, shown in the code below. The code for checking the right value is the same but with the sides switched.

```
while(checkLeft()){
    myMotorR->setSpeed(turnRate);
    myMotorL->setSpeed(0);
}
```

This design is effective for navigation of the course, but it is not very intelligent or fast. In order to increase speed of the car around the track, we tried to code logic that would mimic the action of an actual driver when navigating a course. Namely, drivers will speed up on straightaways and slow down on turns. In order to copy this effect, we added an "acceleration" for the straightaways and turns that modifies the speed of the car. So, when the car is on a straightaway, its speed will continually increase (up to a point) and the turn will be set to minimum, and when the car is at a turn, the speed will continuously drop while the turn rate will increase. The logic for turning is that the same side wheel will continuously decrease in speed the longer it is over tape while the opposite wheel will continuously increase in speed the longer it is over tape. This creates the effect that the car will quickly go through the straight parts and increasingly turn on bends so it adjusts for the sharpness of the turn.

```
while(checkLeft()){
    if(forwardSpeed > lowestSpeed) forwardSpeed -= 5 * speedAcc;//decrement speed by acceleration
    turnRate += turnAcc;//increment turn rate

    myMotorR->setSpeed(forwardSpeed + turnRate/3);//gradually increase the speed of one wheel a ccording to turn rate
    if(turnRate>forwardSpeed){
      myMotorL->setSpeed(0); //if the would be left wheel rate is less then 0 set it to 0
      sendData(0,forwardSpeed + turnRate/3);//send the data to the python program
    }else{
      myMotorL->setSpeed(forwardSpeed - turnRate);//otherwise set it to the gradually decremented value according to the turn rate
      sendData(forwardSpeed - turnRate,forwardSpeed + turnRate/3); //send the data to the python program
    }
  }
```

5.2 Python Visualization

In order to visualize our sensor and motor values, we used Python to read from Arduino's serial monitor. First, we initialized some lists which are later populated with values corresponding to the list name (i.e. leftMotor contains values from the left motor).

```
cxn = Serial('/dev/ttyACM0', baudrate=9600)

fig = plt.figure()

running = True
leftMotor = []
leftSensor = []
rightMotor = []
rightSensor = []
input()
```

Next, we reformat and parse the data. As the comments show below, we remove useless characters and add commas to separate each value. In the most embedded if statement, we populate the 4 initialized lists with the appropriate values. Finally, we have Python stop reading from Arduino's serial monitor when it reaches 500 data points in each list.

```python
while running:
    while not cxn.inWaiting():
        continue
    res =cxn.readline()
    print(res)
    result = res.decode("utf-8","ignore")
    print(result)
    if result[0] == '<':
        result = result[1:-1]   #cut off carrot
        result = result.replace('\n','')   #replace any \n 's with nothing
        result = result.replace('<',',')   #replace any <'s with commas
        data = result.split(',')
        if len(data) == 4:                        #if it is usable data containing one value from each sensor and motor
            print(len(leftMotor))
            leftMotor.append(float(data[0]))     #add the left motor value to the leftMotor list
            leftSensor.append(float(data[1]))    #add the left sensor value to the leftSensor list
            rightMotor.append(float(data[2]))    #add the right motor value to the rightMotor list
            rightSensor.append(float(data[3]))   #add the right motor value to the rightMotor list
    if len(leftMotor) > 500:
        break                       #stop after 500 because we don't need more than ~500 data points to create a useful visualization
```

Since the motor and sensor values were part of very different scales (motor values ranged between 20 and 50; sensor values between 300 and 1000), we scaled them to be within the same range in order to make the two sets more easily comparable. To do this we divided the values in each list by the largest value in that list, making all values range between 0 and 1.

```python
leftMotor  = np.array(leftMotor)
leftSensor= np.array(leftSensor)
rightMotor  = np.array(rightMotor)
rightSensor  = np.array(rightSensor)

leftMotor  = (np.array(leftMotor))/np.amax(rightMotor)
leftSensor  = (np.array(leftSensor) - np.amin(leftSensor))/np.amax(leftSensor)
rightMotor  = (np.array(rightMotor))/np.amax(rightMotor)
rightSensor  = (np.array(rightSensor) - np.amin(rightSensor))/np.amax(rightSensor)
```

Once the data was cleaned up, organized, and scaled, we plotted it using matplotlib.

```python
plt.plot(np.linspace(1,len(leftMotor),len(leftMotor)),leftMotor,label="Left Motor")    #linspace(start,end,number of points)
plt.plot(np.linspace(1,len(leftMotor),len(leftMotor)),leftSensor,label="Left Sensor")    #linspace(start,end,number of points)
plt.plot(np.linspace(1,len(leftMotor),len(leftMotor)),rightMotor,label="Right
    Motor")     #linspace(start,end,number of points)
# plt.plot(np.linspace(1,len(leftMotor),len(leftMotor)),rightSensor,label="
    Right Sensor")     #linspace(start,end,number of points)
plt.legend()
plt.xlabel('Time')
plt.ylabel('Values')
plt.show()
```

The final plot is displayed in Figure 4.

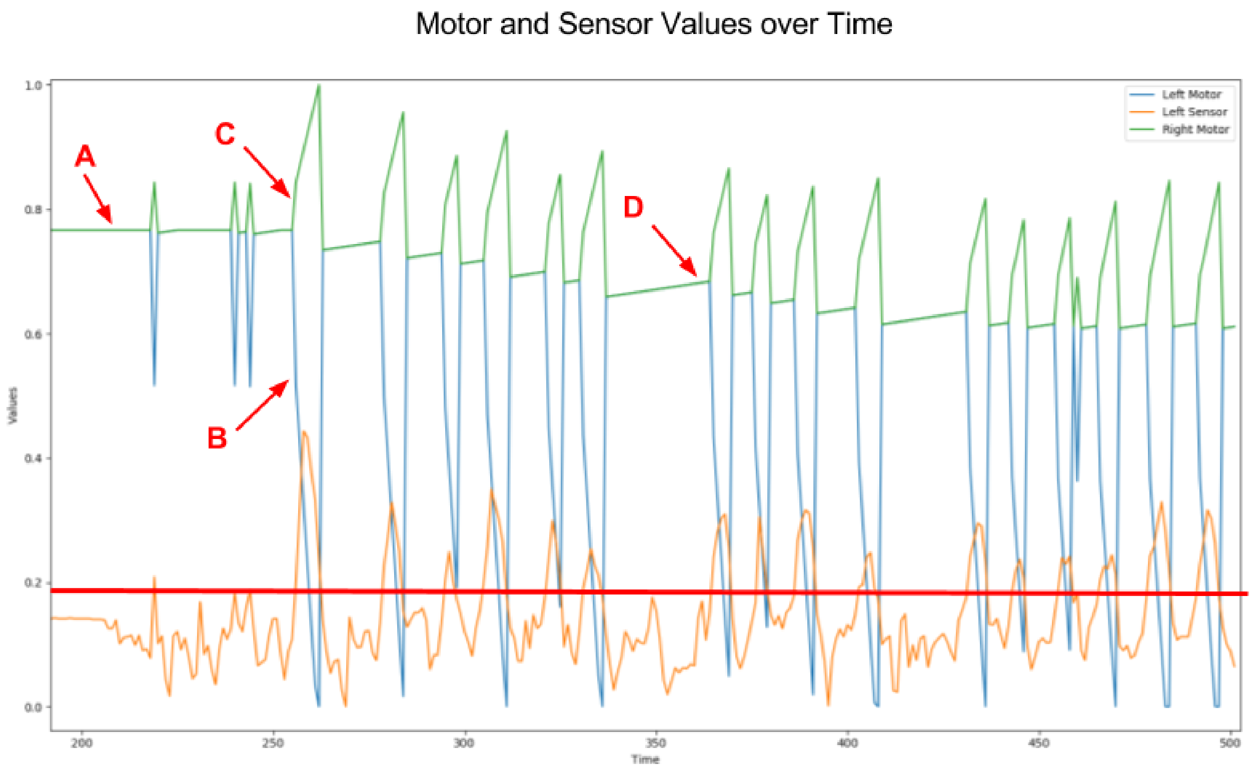

Figure 4: Plot displaying the left and right motor values against the left sensor value as the robot goes around a left hand turn.

Figure 4 shows the expected actions of the line follower when encountered with a left hand turn. As discussed in the source code, both the motor data and the sensor data were scaled according to their max value so that all values fall between 0 and 1. The purpose of this is so that they can more easily be compared since originally the motor data would range between 20 and 50 and the sensor data would range between 300 and 1000. Additionally, the right sensor data was not included in the graph because it was constant the entire time, not pertinent for the understanding of the system, and cluttered up the visualization. The graph for a right hand turn would look largely similar to this one except the blue and green lines would be switched. The red line towards the bottom of the plot shows the threshold for the sensor to return that it sees black tape (on a non scaled plot the value would be 600).

The robot can be seen approaching the curve at top speed at point A. When it approaches a sharp turn at points B and C, the right motor continuously increases its speed as shown at point C, while the left motor sharply continuously decreases its speed as shown at point B. This property causes the effect that when the line follower read tape on one side, the robot will try to maintain speed but will execute a sharper and faster turn the longer it reads tape on that side. So, on slight turns the robot will retain speed while on sharp turns it will slow down until it completes the turn.

Another property of the code, the variable forward speed of the robot, can also be seen in the graph. When it approached the turn, it was travelling top speed, but as it rounded the corner the top speed was decreased as shown at points A vs D. But, when it reaches a straightaway, the speed slowly builds again as shown at point D.

6 Victory Lap

A video of our robot successfully completing the track can be found here. It takes the robot about 1:15 minutes to travel around the entire track.
Link: https://photos.app.goo.gl/eyIUQEpGBSaCeXs52

7 Reflection

One of the biggest technical challenges in this lab was making design decisions in assembling the follower. Problems such as assembling the power supply, correctly soldering the wires, mounting the Arduino and wheels, and assembling the IR sensors to be compact and connect to the Arduino, were the most difficult due to our lack of experience with mechanical and electrical work.

Another challenge we faced was both team members' busy schedules. Additionally, both of us left campus for the 3-day weekend, which made it difficult to collaborate, leaving a large portion of the work for the last few days before the lab was due. However, by dividing tasks and being willing to work individually and combine our work as a team, we successfully completed the lab.

Overall, we gained some valuable knowledge that will be applicable to systems we will work with in the future - and we hope to adapt that much quicker when faced with the next challenge.

We are grateful to have benefited from the expertise and patient assistance of the teaching team and NINJAs. We especially appreciated Daniela Faas's assistance with resistor values, Stan Reifel's explanations of bang-bang controllers, and Trent Dye's feedback on our visualization. Thanks to all for another great learning experience!

8 Appendix

8.1 Arduino Source Code

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

float minTurn = 5;          //min rate at which to turn
float turnRate = 5;         //variable for current turn rate
float turnAcc = 3;          //acceleration while turning (the more time spent turning, the faster the turn)
float lowestSpeed = 20;   //minimum speed
float maxSpeed = 25;     //maximum speed
float forwardSpeed = 20;//variable for current speed
float speedAcc = .03;  //acceleration while not turning
int leftThres = 600;  //threshold to detect line
int rightThres = 600;
int rightSensor = 1000; //variable for right sensor value
int leftSensor = 1000; //variable for left sensor value
Adafruit_MotorShield AFMS = Adafruit_MotorShield();  //declare motor shield
  // Select which 'port' M1, M2, M3 or M4. In this case, M1
Adafruit_DCMotor *myMotorL = AFMS.getMotor(1); //declare motor ports
Adafruit_DCMotor *myMotorR = AFMS.getMotor(2);
void setup() {
  AFMS.begin();
  Serial.begin(9600);
  // You can also make another motor on port M2
  //Adafruit_DCMotor *myOtherMotor = AFMS.getMotor(2);
  myMotorL->setSpeed(10);//initialize the speeds
  myMotorR->setSpeed(10);
  myMotorL->run(FORWARD);
  myMotorR->run(FORWARD);
}

void loop() {
  // put your main code here, to run repeatedly:
  sendData(forwardSpeed, forwardSpeed);                              //print data to Serial moniter for later use in Python
  myMotorL->run(FORWARD);
  myMotorR->run(FORWARD);
  if (forwardSpeed < maxSpeed)forwardSpeed += speedAcc; //increment speed by the acceleration

  turnRate = minTurn;//reset turn rate

  while(checkLeft()){
    if(forwardSpeed > lowestSpeed) forwardSpeed -= 5 * speedAcc;//decrement speed by acceleration
    turnRate += turnAcc;//increment turn rate

    myMotorR->setSpeed(forwardSpeed + turnRate/3);//gradually increase the speed of one wheel a ccording to turn rate
    if(turnRate>forwardSpeed){
      myMotorL->setSpeed(0); //if the would be left wheel rate is less then 0 set it to 0
      sendData(0,forwardSpeed + turnRate/3);//send the data to the python program
    }else{
      myMotorL->setSpeed(forwardSpeed - turnRate);//otherwise set it to the
          gradually decremented value according to the turn rate
      sendData(forwardSpeed - turnRate,forwardSpeed + turnRate/3); //send the
          data to the python program
    }
  }
  myMotorL->setSpeed(forwardSpeed);//reset speed
  myMotorR->setSpeed(forwardSpeed);
  turnRate = minTurn; //reset turn rate

  while(checkRight()){
    if(forwardSpeed > lowestSpeed) forwardSpeed -= 5 * speedAcc;//decrement
        speed by acceleration
    turnRate += turnAcc;//increment turn rate
    myMotorL->setSpeed(forwardSpeed + turnRate/3);//gradually increase the
        speed of one wheel a ccording to turn rate
    if(turnRate>forwardSpeed){
      myMotorR->setSpeed(0);//if the would be right wheel rate is less then 0
          set it to
      sendData(forwardSpeed + turnRate/3,0);//send the data to the python
          program
    }else{
      myMotorR->setSpeed(forwardSpeed - turnRate);//otherwise set it to the
          gradually decremented value according to the turn rate
      sendData(forwardSpeed + turnRate/3,forwardSpeed - turnRate);//send the
          data to the python program
    }                       //print data to Serial moniter for later use in
        Python
  }
  myMotorL->setSpeed(forwardSpeed);
  myMotorR->setSpeed(forwardSpeed);

}

bool checkLeft(){
  leftSensor = analogRead(A1);//get value of left sensor
  return leftSensor > leftThres;//return if it is greater than threshold
}

bool checkRight(){
  rightSensor = analogRead(A0);//get value of right sensor
  return rightSensor > rightThres;//return if it is greate than the thresold
}

void sendData(float leftVal,float rightVal) {//function to send data to the
    python program
  Serial.print('<');//start character
  Serial.print(leftVal);//left motor val
  Serial.print(',');//separator to ease parsing
  Serial.print(analogRead(A1));//left sensor
  Serial.print(',');
  Serial.print(rightVal);//right motor value
  Serial.print(',');
  Serial.print(analogRead(A0));//right sensor value
  Serial.println();
}
```

8.2 Python Visualization Code

```
from serial import Serial, SerialException
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter

# The Serial constructor will take a different first argument on
# Windows. The first argument on Windows will likely be of the form
# 'COMX' where 'X' is a number like 3,4,5 etc.
# Eg.cxn = Serial('COM5', baudrate=9600
#
# NOTE: You won't be able to program your Arduino or run the Serial
# Monitor while the Python script is running.
cxn = Serial('/dev/ttyACM0', baudrate=9600)

fig = plt.figure()

running = True
leftMotor = []
leftSensor = []
rightMotor = []
rightSensor = []
input()

while running:
    while not cxn.inWaiting():
        continue
    res =cxn.readline()
    print(res)
    result = res.decode("utf-8","ignore")
    print(result)
    if result[0] == '<':
        result = result[1:-1]   #cut off carrot
        result = result.replace('\n','')   #replace any \n 's with nothing
        result = result.replace('<',',')   #replace any <'s with commas
        data = result.split(',')
        if len(data) == 4:                          #if it is usable data
            containing one value from each sensor and motor
             print(len(leftMotor))
             leftMotor.append(float(data[0]))     #add the left motor value to
                 the leftMotor list
             leftSensor.append(float(data[1]))    #add the left sensor value to
                 the leftSensor list
             rightMotor.append(float(data[2]))    #add the right motor value to
                 the rightMotor list
            rightSensor.append(float(data[3]))   #add the right motor value to
                the rightMotor list
    if len(leftMotor) > 500:
        break                   #stop after 500 because we don't need more
            than ~500 data points to create a useful visualization

leftMotor = np.array(leftMotor)
leftSensor= np.array(leftSensor)
rightMotor = np.array(rightMotor)
rightSensor = np.array(rightSensor)


leftMotor = (np.array(leftMotor))/np.amax(rightMotor)
leftSensor = (np.array(leftSensor) - np.amin(leftSensor))/np.amax(leftSensor)
rightMotor = (np.array(rightMotor))/np.amax(rightMotor)
rightSensor = (np.array(rightSensor) - np.amin(rightSensor))/np.amax(
    rightSensor)
plt.plot(np.linspace(1,len(leftMotor),len(leftMotor)),leftMotor,label="Left
    Motor")     #linspace(start,end,number of points)
plt.plot(np.linspace(1,len(leftMotor),len(leftMotor)),leftSensor,label="Left
    Sensor")     #linspace(start,end,number of points)
plt.plot(np.linspace(1,len(leftMotor),len(leftMotor)),rightMotor,label="Right
    Motor")     #linspace(start,end,number of points)
# plt.plot(np.linspace(1,len(leftMotor),len(leftMotor)),rightSensor,label="
    Right Sensor")     #linspace(start,end,number of points)
plt.legend()
plt.xlabel('Time')
plt.ylabel('Values')
plt.show()
```