Lab 3: Line Following Robot

Prava Dhulipalla and Meaghen Sausville

October 13, 2017

1 Introduction

The objective of Lab 3 was to construct a robot that followed a path (marked by black tape). The criteria for this was that we had to use some sort of control mechanism to optimize speed around the track (similarly, we had to make sure our code only had to be uploaded once and the robot would run), implement inferfacing so that input can be put into our code and it would affect the output of the robot, and work under certain design constraints. These constaints were that we had to share a chassis and a wheel set with a partner team in the other section, and that we only had access to our physical chassis after a week of working on the project.

With these criteria in mind, we created a robot that had two IR distance sensors connected to a small carriage with two servo wheels. As the robot moves, the IR sensors send data back that determine whether the robot is off course or not, and then corrects it.

2 Materials

We had access to certain materials in order to construct and run our project. Here are the materials we used:

- 1x Arduino Uno R3
- 1x solderless breadboard
- 2x IR reflectance sensors
- 2x 4700 Ohm resistors
- 2x 50000 Ohm resistors
- 2x 200 Ohm resistors
- 8x Male-To-Female connectors
- 4x 40" wires
- 1x 12V power supply
- 2x four pin Molex wire-to-wire connector
- 4x genderless crimp terminals
- 1x female power jack
- 1x male power jack
- 2x Servo motors
- 1x chassis
- 4x 3/4" 4-40 screws
- 4x 4-40 hexnuts
- 1x 144 in^2 MDF
- wood glue

3 Design Constraints

As mentioned, there were certain criteria under which we had to operate in order to better reflect real-world constraints. Therefore, we only had access to one servo motor, and somebody in a later time section working on the same project would have the other servo motor. However, it turned out that the later time section had fewer teams, so we ended up not having to share a chassis. However, to make it 'more fair' and more like the other groups' experiences, we decided to leave the constructed robot 'skeleton' (with the servo motors and the chassis) in the room. We also constructed our addition to the chassis to be removable so that nothing was directly attached to the chassis. In order to make it removable, refer to the Physical Setup section.

4 Physical Setup

4.1 Chassis

Our attachment to the chassis had to be both removable and large enough to hold the Arduino and a breadboard. In order to do that, we CAD-ed and laser cut a box made of five distinct pieces– a base and four sides. We designed a box that could be screwed onto the the chassis base that was large enough for both the breadboard and the Arduino. Because the attachment was screwed on, it could easily be removed in order for other teams to use the same chassis. The setup is demonstrated in Figure 1 below.

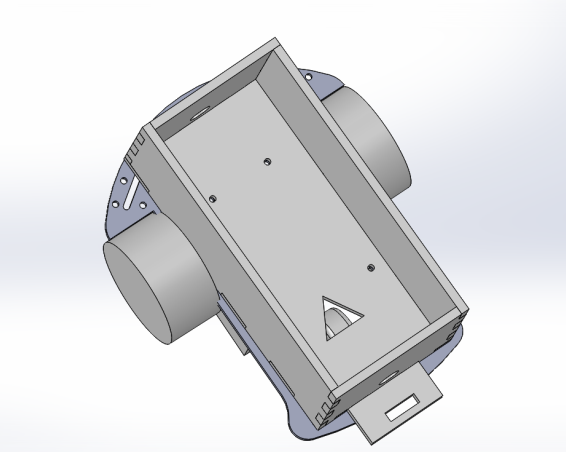

Figure 1: The chassis assembly

This attachment fits the breadboard with the circuitry and then the Arduino with the Adafruit Motorshield. The sensors needed to be outside of the box in order to see the floor. To accomplish this we made a lip stick out of the front of the box over the wheels with a hole just big enough for the two sensors to press fit into. There was a hole in the front of the box in order for wires from the breadboard to reach the sensors. This way the sensors could read the tape line but were still connected to our circuit. The sensors were very close together in order to implement our control mechanism - which turns when one of the sensors is reading light and the other dark. If both of them are reading dark, then the robot will go in a straight line. More details about this are located in the Programming section.

If we had a second iteration of building this robot, there are certain ways we can make this attachment to the chassis better. The main way is to nix the breadboard entirely and introduce a protoboard, which would have made the attachment less large and heavy. However, we don't mind the way we did our first-pass with a breadboard. One, because it introduced a source of complication that we had to overcome, and two, Meaghen found the process of CAD-ing the box a useful learning process - especially figuring out how the pieces fit together and how everything was supposed to line up with the chassis. We also would have liked to make the wire access holes in the front and back of the chassis a bit larger to make more room for cables to pass through. In addition, the length of the motor wires was not accounted for, so the Arduino ended up resting on

the bottom rather than on the top (side note: we also have no idea how we CAD-ed a nice looking and well-fitting attachment that also acts super jank . . .).

4.2 Wiring

More details on the circuitry of the robot will be found in the Electrical Setup section. However, as for the physical wiring, the only way this affected our mechanical set-up is to have holes where the wires need to go through (hence the hole in the front and the back).

Even this could use some work, because the Arduino cable wasn't able to fit through the hole and instead awkwardly fits in through the top. If the hole was larger or the laser cut piece had a divet that could fit the wire in, then this problem could be remedied.

5 Electrical Setup

5.1 Getting Power and Connecting the Motors

The circuit was more complicated than the last lab's because it had a long umbilical cord of power. The chain of command is as follows: there was a 12 volt power supply that was plugged into the wall. The power supply was connected to female power jack that had four long wires attached– two ground, two power. The four wires were wound together to keep them from becoming tangled. At their other end they were connected to a four pin Molex wire-to-wire connector. From that connector the wires branched out into two different directions, a power and a ground going both ways. One set connected to the Adafruit motor shield that attached to the Arduino, and the second set attached to the Arduino itself. This setup is demonstrated in Figure 2 below.

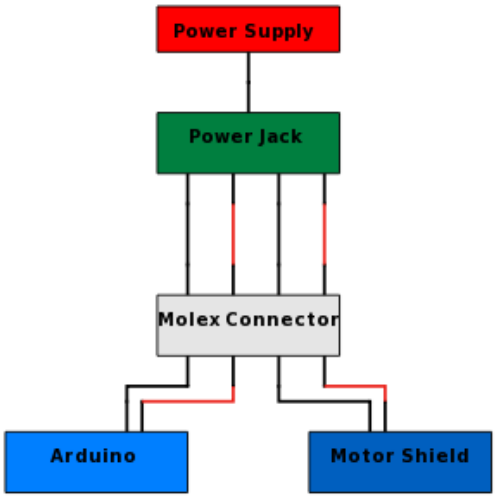

Figure 2: The power for our Arduino and motor shield

5.2 Wiring the Sensors

The circuit diagram for the IR sensor can be seen in Figure3. In order to attach two sensors, we had two separate circuits made on the solderless breadboard (as mentioned before, one way we could have elevated this lab is to use a protoboard and attached the circuitry on that, soldering the connections, or using a smaller solderless breadboard, in order to reduce the area that the breadboard took up).

To find the resistor values (for the phototransistor, the resistor values for the LED were given to us), we looked on the spec sheet for the IR sensor and found that the ideal collector emitter saturation current was 0.1 mA, which according to a graph on the data sheet was supposed to correspond with a forward current of 1.09 mA. However, the data sheet said that the correspondence was discovered by reflecting the IR sensor off a mirror at a much closer distance than our sensors

were at, and the floor is also much less reflective than a mirror. Therefore, it was suggested by a ninja that we halve the forward current to 0.5 mA in order to reduce railing, and to continue reducing the current until our sensors reported minimal railing.

We first used Ohms law, Voltage = Current $\cdot$ Resistance, in order to determine the resistance needed for our circuits. We knew our input voltage would be 5 volts, the voltage drop across our resistor would the be 0.3 V (suggested by the spec sheet and the Ninja and our resistance was 0.5 mA. Therefore:

$$5 - 0.3 = 0.5 * 10^{-}3 * R$$

$$R = 9400$$

In order to satisfy this equation, we used two resistors in series of 4700 ohms each, whose combined resistance was 9400 ohms. However, with this value our sensors kept railing and reading a very high value. It was previously suggested to us that if this happened, continue decreasing the amount of current passed by increasing the resistance value. The railing stopped occuring when we replaced one of the 4700 ohm resistors with a 50,000 ohm resistor, which led to a combined resistance of 54,700 ohms and a current of 0.09 mA.

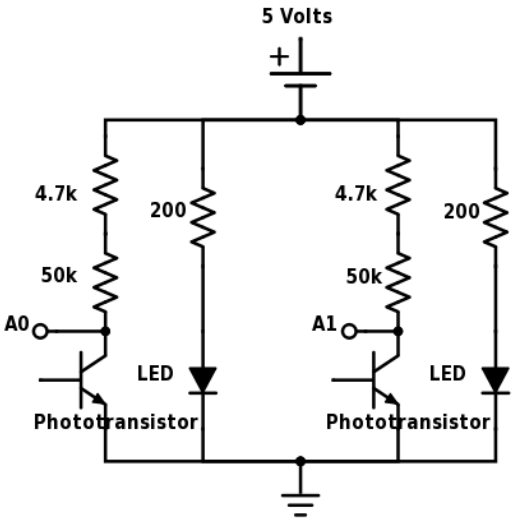

Figure 3: Circuit for the IR sensors

Our resistor values are much higher than most other groups because of the extra distance between our sensors and the floor– most groups held their sensors very close the the floor which decreased the amount of railing. If we had constructed our box in a way that held the IR sensors closer to the floor, we would likely have had less railing with a smaller value resistor.

One last thing to note is that there are two wires leading to A0 and A1 respectively on the Arduino in between the resistors and the phototransistor. This is how the Arduino actually reads the sensor values– the different readings are sent along those wires to the Arduino and if we didn't have them, the sensors would still work but we wouldn't be able to access the data. The total circuit, where this circuit is connected to the Arduino, is depicted below in Figure 4.

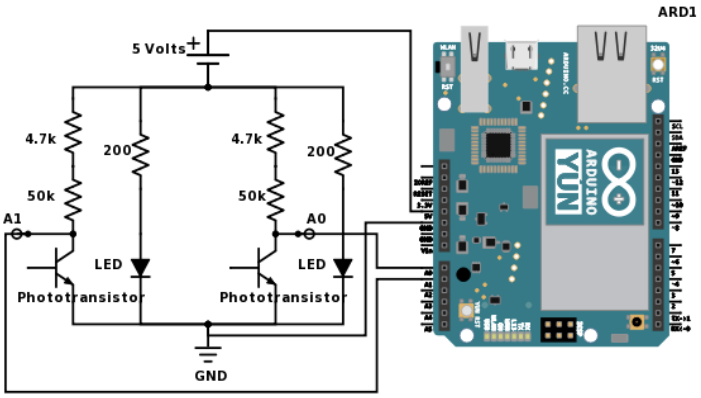

Figure 4: Circuit for the IR sensors connected to the Arduino

5.3 Calibration for IR Sensors

In order to implement the control mechanism, we needed to calibrate the sensors and figure out the values they read. More details can be found in the Programming: Control section, but the way we implemented the movement of the robot was essentially through four cases. These cases are dependent on the following sensor reading controls:

- both sensors sensing ground (both not on track)
- right sensor sensing ground (not on track)
- left sensor sensing ground (not on track)
- both sensors sensing dark (both on track)

To figure out the first two cases, which is essentially just checking if one of the sensors is sensing ground, we based the threshold values on to figure out whether the state of the robot is on the line or not, and from there turn right or left. We had to figure out what this threshold is.

The following table shows what the left and the right raw sensor readings in terms of different surroundings:

case	left	right
black tape	664	449
floor	73	68
floor with black tape	650	667

As can be seen, there is a vast difference between the floor and the black tape on the floor raw sensor reading values, which means we can set a threshold - 200 worked well for our purposes - to distinguish between sensing the floor and sensing the black tape on the floor. Below 200, it is sensing the floor. Above 200, it is sensing the black tape.

Our actual procedure for calibration was as simple as expected. While the sensors were mounted on the chassis, we moved the sensors over a 1) plank of wood with black tape on it, 2) the floor, and 3) the black tape on the floor.

6 Programming

In order to control the robot and plot it's trajectory around the course, we had use Arduino code in order to drive its trajectory and Matlab code in order to plot it.

6.1 Control

This is not code, but this is an important part of the robot that was implemented via Arduino code. We elected to use bang bang control. Bang bang control (aka a hysteresis controller) is a feedback controller that switches abruptly between two states. In this case, there are four states - move right based on certain sensor values, move left based on certain other sensor values, move straight ahead based on certain other sensor values, and our final case is turning left or right based on a previous state (when it is completely sensing the floor). This control mechanism essentially just takes in sensory inputs and chooses which case to perform based on that.

If we were to iterate on the control mechanism, we would probably elect to use P-control, then PI-control, then PID-control. The reason we didn't this time boils down to the fact that we were actually discouraged to do so due to time constraints and the fact it didn't add value to our learning goals. When asking the professors about PID-control, we were encouraged to do Bang Bang control instead because 1) it is easier to implement and 2) it still fulfills the lab requirements. Secondly, both my partner and I had already learned about PID control in another class, so we weren't really interested in it for PoE. In addition, we felt that implementing it would require more frustration bur not add more knowledge (side note: not saying we are too smart for PID control or anything, reading this back to ourselves we realize we kind of sound pretentious).

6.2 Arduino Code

There are six main important sections embedded within the Arduino code.

This function block was important because it defined some very important variables. First, the result, so that I could use the Serial monitor in order to project the IR sensor data on the motor speed data. The second variable was a byte that was primarily for debugging (it would change based on which case in the rest of the code was being used, and it showed us the right case was called and that other times it was called correctly, but that a seemingly broken motor was the issue). The final variable helped me define my case of the motor sensing both white. Instead of going backwards, turnFlag (which acts as a state-holding element) will turn left or right depending on whether the robot initially wanted to go left or right, respectively.

```
//other necessary variables
String result; //outputs necessary values
byte whatToDo;  //helped to determine if the sensors are linking to the right value
byte turnFlag = 0; //if it is 0, previous state is turning left; if 1, previous
    state is right --> helps when it reads white
```

This function block is very simple but very vital. Part of this lab was to be able to interface from the laptop to the robot in order to better the control system. We noticed that a lot of the times, the robot was either going too fast or too slow when it came to turns. By setting the 'normspeed' (aka the speed of the robot when it is going straight), it allows a better control of our whole system. Too slow, and the pace of the robot won't be optimized. Too fast, the sensor values don't recognize the black tape in comparison to the floor. Note that it is in void loop and thus can a value each iteration.

```
void loop() {
  //reads from serial loop and assigns it to value
  if (Serial.available()) {
    normspeed = Serial.read();
  }
```

This is case one for the robot control. If the left sensor reading is over a calibration-determined threshold (meaning it is sensing black tape) and the sight sensor is reading under this threshold (meaning it is the floor), the robot turns left. Note that in here, we set 'whatToDo' and turnFlag. Also, the wheels much be set backwards instead of forwards because the motors are backwards.

```
  if (readingL > threshold and readingR < threshold)
  {
    leftMotor->setSpeed(slowspeed);
    rightMotor->setSpeed(turnspeed);

    //sets these values for output
    leftspeed = slowspeed;
    rightspeed = turnspeed;

```

```
    //sets debugging variable and flag to the right value
    whatToDo = 1;
    turnFlag = 0;

    // runs the wheels --> has to be run backwards
    rightMotor->run(BACKWARD);
    leftMotor->run(BACKWARD);
  }
```

This case is the same as the previous case, except it turns right in order to stop it from veering left.

```
// if the left sensor reading is a black tape line (aka greater than threshold),
    then move right (onto the line)
  else if (readingR > threshold and readingL < threshold)
  {
    leftMotor->setSpeed(turnspeed);
    rightMotor->setSpeed(slowspeed);

    leftspeed = turnspeed;
    rightspeed = slowspeed;

    //sets debugging variable and flag to the right value
    whatToDo = 2;
    turnFlag = 1;

    rightMotor->run(BACKWARD);
    leftMotor->run(BACKWARD);
  }
```

This is our last case - if both of the sensors are reading very low values, which means they are on the floor. In this case, I decided to use a flag. If the previous move was a left, thwn 'turnFlag' is 0, and would be 1 if the previous move was a right instead. By keeping track of this, it then continues travelling in the direction it had attempted to do before. This was implemented because simply backing up the robot a small portion didn't work.

```
   // if both sensors are essentially reading the floor
  else if (readingL < threshold and readingR < threshold)
  {
    //sets debugging variable

    //if in previous state it wanted to go left, go left
    if (turnFlag == 0) {
    leftMotor->setSpeed(slowspeed);
    rightMotor->setSpeed(turnspeed);

    leftspeed = slowspeed;
    rightspeed = turnspeed;

    whatToDo = 3;

    }

     // if in previous state it wanted to go right, go right
    if(turnFlag == 1) {
    turnFlag = 1;

    leftMotor->setSpeed(turnspeed);
    rightMotor->setSpeed(slowspeed);

    leftspeed = turnspeed;
    rightspeed = slowspeed;
    }
    whatToDo = 4;

    rightMotor->run(BACKWARD);
    leftMotor->run(BACKWARD);

  }
```

If both sensors read greater than the threshold value, it means both sensors is on the line. This means that the robot will head straight (aka continue going along the path it is already traveling).

7 Final Robot

Here are two pictures of our final robot:

Figure 5: Front view of our robot.

Figure 6: Side view of our robot.

Refer to Figure 1 for comparison to our CAD model. Our robot ended up physically the way we expected it it to be but as mentioned before if we had future iterations to work on this, we would make the wire access hole in the front larger, place the sensors closer to the ground, and add some aesthetically pleasing design elements.

7.1 Video

Here is the video of our line following robot following a line: Video on Google Drive

7.2 IR Sensor Data and Motor Speeds

(Please note that for some reason, my calibration started becoming more prone to error so I had to increase the threshold to a raw sensor threshold of 350.)

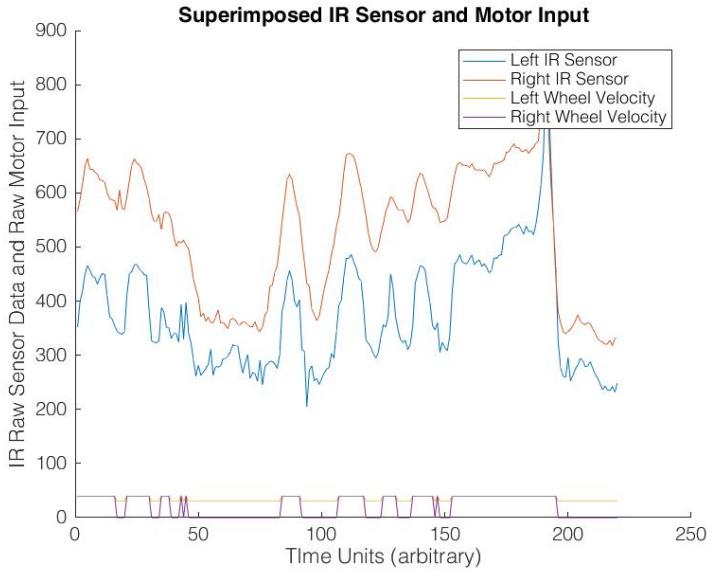

Figure 7: Superimposed graph of the IR Sensor data and motor speeds for each wheel. The IR Sensor data is in blue and orange, and the motor speeds are purple and yellow.

This is the superimposed image of the IR sensor data against the motor speeds. This data seems to make sense - the robot was mostly going straight, except near the end, the left sensor starts reading a lower value than the right sensor (which means that the left sensor was sensing ground and the right sensor was reading the tape). To compensate for this, the robot had to turn right. To turn right, the right wheel velocity becomes 0 and the left wheel velocity becomes 30. Visually, the robot did turn right around a bend, so this data validates the robots' actions.

8 Problems and Reflection

Throughout this lab, my partner and I had our fair share of problems. One, with the actual devices themselves. Not only was our Adafruit Motorshield already shorted by the time we had used it, but one of our motors/wheels was clearly broken (see the Video section). Although the debugging variable 'whatToDo' in the code section had actually pointed to the right direction, sometimes the robot itself would freeze up - very randomly. This didn't seem to get better with increasing or decreasing the speeds. Due to this, it was suggested that we probably would have to 'nudge' our robot once or twice - which we had to. However, when we weren't recording (and fellow PoE students Nate Sampo and Will Derksen can attest to this), the robot followed the line perfectly. However, every single attempt at recording led to a very bad run (for seemingly no reason) - for instance, it the sensors would randomly stop working, the motor wheel would come off, something would become unplugged, etc. We was told to chock this up to bad luck and submit our video with the caveat that it does work!

Also throughout this lab, we learned more about the integration between mechanical, electrical, and coding systems. All three parts of this project were very important and each part contributed something important to the final robot. This made us realize two things. One, it is really important to communicate what is being done in each system. There were a couple times when it was unclear what was being worked on and what was being done. We feel that if more communication was being done across all three, there would have been a more simplified mechanical (since electrical didn't have to use an entire breadboard), and differently structured code (the mechanical system was essentially just built around the code), there would have been a more cohesive project. But this taught us a lot about idea iteration and ways to 'fix' ones mistakes - after all, a kind of cumbersome mechanical, electrical, and software system, the robot ended up working.

9 Appendix

Here is the entirety of our Arduino code:

```
// imports necessary libraries
#include <Wire.h>
#include <Adafruit_MotorShield.h>
#include "utility/Adafruit_MS_PWMServoDriver.h"

// adds both motors
Adafruit_MotorShield AFMS0 = Adafruit_MotorShield();
Adafruit_MotorShield AFMS1 = Adafruit_MotorShield();

// left motor connected to M3, right to M4 on the motor shield
Adafruit_DCMotor *leftMotor = AFMS0.getMotor(3);
Adafruit_DCMotor *rightMotor = AFMS1.getMotor(4);

// sensor variables
//sensor connections
const int sensorL = A0;
const int sensorR = A1;
//sensor reading variables
int readingL = 0;
int readingR = 0;
//threshold - below, reading floor; above, reading black tape
const int threshold = 350;

// speed variables
int normspeed = 40;  // default speed
const int turnspeed = 30;  // speed one wheel would move at in order to turn
const int slowspeed = 0;  // speed other wheel would move at in order to enable a
    turn

//output variables
int leftspeed;
int rightspeed;

//other necessary variables
String result; //outputs necessary values
byte whatToDo;  //helped to determine if the sensors are linking to the right value
byte turnFlag = 0; //if it is 0, previous state is turning left; if 1, previous
    state is right --> helps when it reads white


// 'begins' motors, sets up sensors, and starts Serial
void setup() {
  pinMode(sensorL, INPUT);
  pinMode(sensorR, INPUT);
  AFMS0.begin();
  AFMS1.begin();
  Serial.begin(9600);
}

// bang bang control loop and movement
void loop() {
  //reads from serial loop and assigns it to value
  if (Serial.available()) {
    normspeed = Serial.read();
  }

  // reads from both sensors
  readingL = analogRead(sensorL);
  readingR = analogRead(sensorR);

  // if the right sensor reading is a black tape line (aka greater than threshold),
      then move left (onto the line)
  if (readingR > threshold and readingL < threshold)
  {
    leftMotor->setSpeed(slowspeed);
    rightMotor->setSpeed(turnspeed);

    //sets these values for output
    leftspeed = slowspeed;
    rightspeed = turnspeed;

    //sets debugging variable and flag to the right value
    whatToDo = 1;
```

```
71      turnFlag = 0;
72
73      // runs the wheels --> has to be run backwards
74      rightMotor->run(BACKWARD);
75      leftMotor->run(BACKWARD);
76    }
77
78    // if the left sensor reading is a black tape line (aka greater than threshold),
          then move right (onto the line)
79    else if (readingL > threshold and readingR < threshold)
80    {
81      leftMotor->setSpeed(turnspeed);
82      rightMotor->setSpeed(slowspeed);
83
84      leftspeed = turnspeed;
85      rightspeed = slowspeed;
86
87      //sets debugging variable and flag to the right value
88      whatToDo = 2;
89      turnFlag = 1;
90
91      rightMotor->run(BACKWARD);
92      leftMotor->run(BACKWARD);
93    }
94
95    // if both sensors are essentially reading the floor
96    else if (readingL < threshold and readingR < threshold)
97    {
98      //sets debugging variable
99
100     //if in previous state it wanted to go left, go left
101     if (turnFlag == 0) {
102     leftMotor->setSpeed(slowspeed);
103     rightMotor->setSpeed(turnspeed);
104
105     leftspeed = slowspeed;
106     rightspeed = turnspeed;
107
108     whatToDo = 3;
109
110     }
111
112      // if in previous state it wanted to go right, go right
113     if(turnFlag == 1) {
114     turnFlag = 1;
115
116     leftMotor->setSpeed(turnspeed);
117     rightMotor->setSpeed(slowspeed);
118
119     leftspeed = turnspeed;
120     rightspeed = slowspeed;
121     }
122     whatToDo = 4;
123
124     rightMotor->run(BACKWARD);
125     leftMotor->run(BACKWARD);
126
127   }
128
129   //both sensors must be reading the tape line, so just go straight
130   else
131   {
132     leftMotor->setSpeed(normspeed);
133     rightMotor->setSpeed(normspeed);
134
135     leftspeed = normspeed;
136     rightspeed = normspeed;;
137
138     whatToDo = 0;
139     // runs the wheels (so the desired action will occur)
140
141     rightMotor->run(BACKWARD);
142     leftMotor->run(BACKWARD);
```

```
143     }
144
145     result = result + readingL + " " + readingR + " " + leftspeed + " " + rightspeed
            + " " + whatToDo;
146     //allowed to use delays - if just printingout values
147     delay(5);
148     Serial.println(result);
149     result = "";
150 }
```