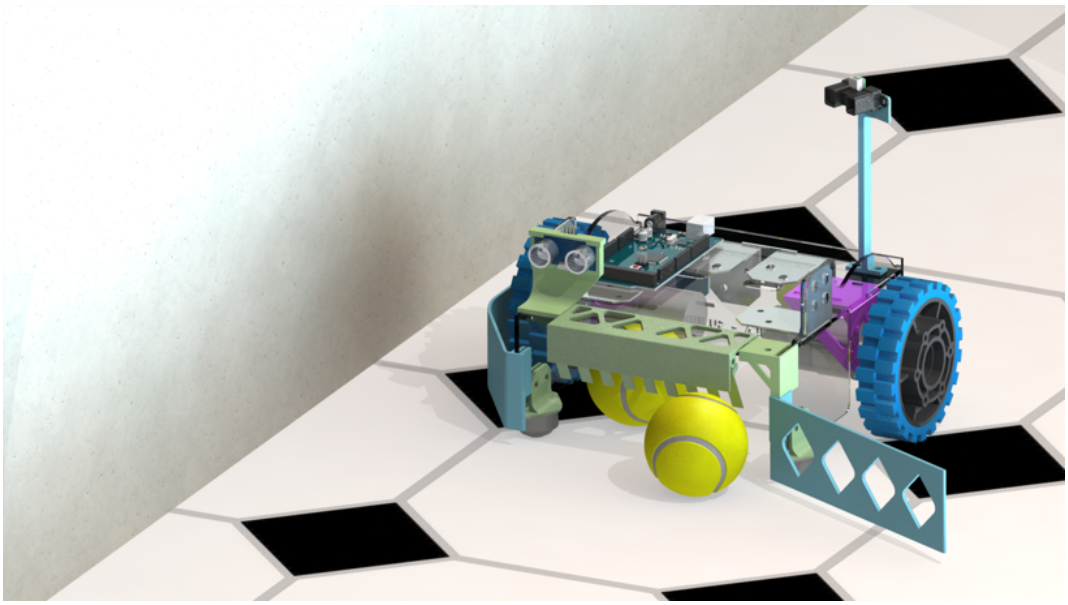
Competition Lab
Maze Search and Rescue Robot

Group 5

Juliette Mitrovich

Sheila Moroney

Sujani Patel



Lab Summary

The purpose of this lab was to create a robot that could navigate a small maze, collect at least two tennis balls, and exit the maze. The more tennis balls the robot collected and the faster the maze was completed resulted in a higher score. We learned a lot from this lab. Since we used a passive mechanism for ball collection, much of our focus was directed on the robot control and wall following. We also learned how to integrate sensors into the robot, including an IR sensor and a limit switch. Overall, this lab was a fun way to test what we learned in class and in labs throughout the semester!

Overall design of the robot / Robot chassis design and construction

Our final design of our robot, as seen below in **Figure 1**, was a small acrylic frame with two grippy wheels and included an IR sensor and an Ultrasonic sensor to follow walls within a maze. The acrylic was cut on the laser cutter at the Learning Factory and easily assembled with a combination of metal brackets, 3D printed brackets, and E6000 glue. This process made sure the robot was sturdy. We used a group of 3D printed parts to passively collect tennis balls as we followed the walls of the maze.

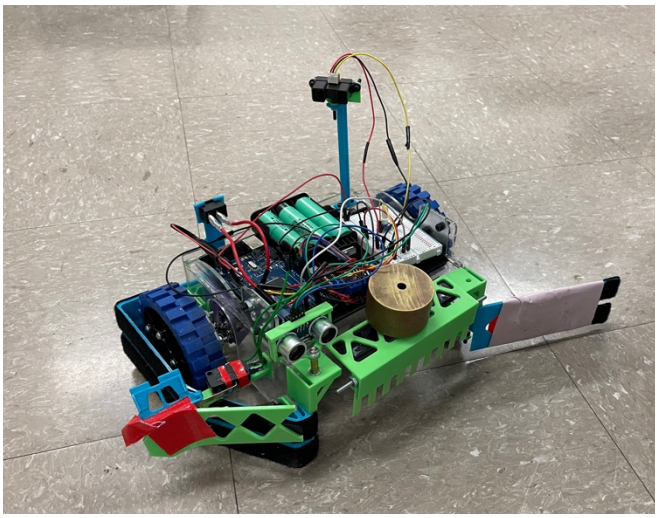


Figure 1. Final Robot

Ball Collection

Since the main goal of the competition was to collect tennis balls quickly, we designed our robot with that in mind first. We chose to use a 3D-printed, one-way hinge gate, that easily let the tennis balls pass through, but stopped them from leaving. The hinge, shown in **Figure 2** below, was designed to swing easily with minimum contact points and to be 3D printed quickly. We used two long machine screws as the axle for the hinge.

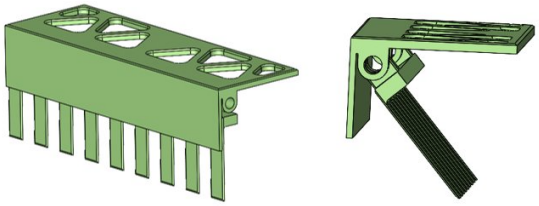


Figure 2. CAD Model of One-Way Hinge Gate

To reduce the amount of weight on the robot, the tennis balls roll freely underneath the acrylic chassis as seen in **Figure 3**. We designed the chassis so it would be big enough to fit 3 balls underneath.

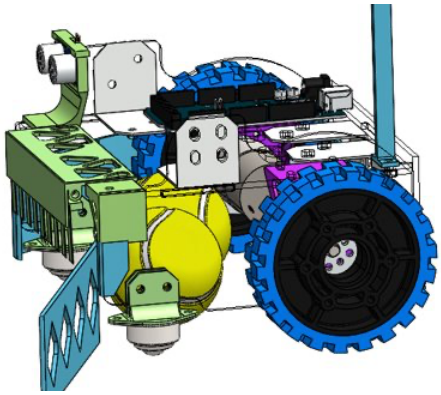


Figure 3. CAD Model robot with two tennis balls under the chassis

After we started testing our robot in the test maze, we noticed that some of the tennis balls were too far from the wall for our robot to drive over while wall following. So, we added a long, left arm at an angle to try to guide the farther balls under our robot, as shown below in **Figure 4**. In some cases, the ball would bounce off the arm in the opposite direction. To fix this, wrapped the main part of the arm with duct tape, sticky side out, to stop the balls from bouncing away.

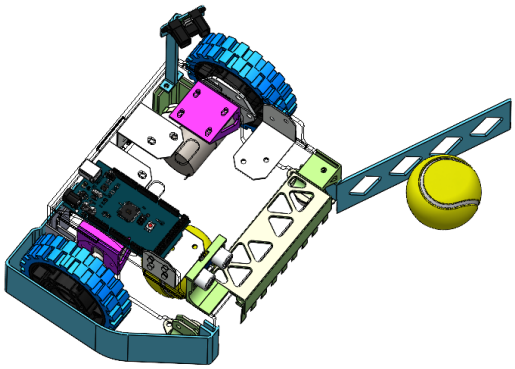


Figure 4. CAD Model robot with arm guiding tennis ball toward chassis

Batteries and power management

To power the robot, we chose to use rechargeable 18650 lithium-ion batteries. Three of these batteries gave us 12.3 volts, enough to power the motors and the Arduino Mega. Powering the motors and the Arduino from the same supply was important because it allowed us to have a common ground. This protected us from accidentally causing a ground fault loop and destroying the devices. The wiring diagram can be found below in **Figure 5**.

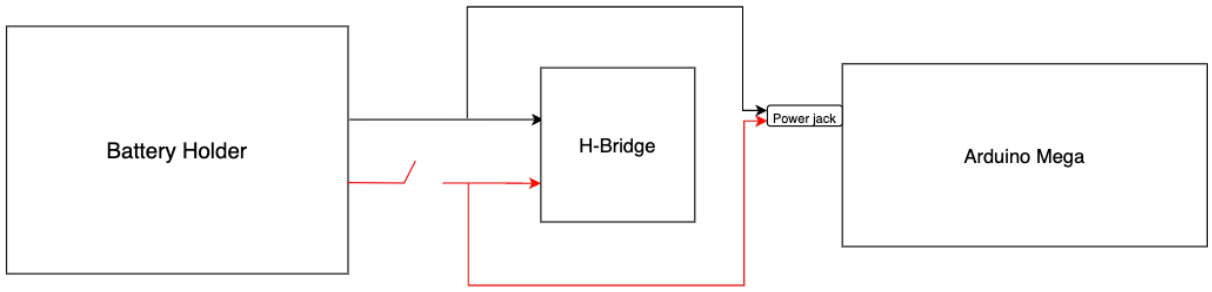


Figure 5. Power management wiring diagram

To power the H-Bridge, we connected the wires from the battery holder to the screw terminals. We then soldered two more wires in parallel to a male power jack. This was used to power the Arduino Mega. Finally, we soldered a single pull single throw (SPST) switch between the power line. This allowed us to easily turn the robot on and off without having to remove a battery from the pack. The batteries lasted an extremely long time, with us only needing to charge them once before the competition to allow for maximum power.

Robot behavior and control

Speed Control and Steering

To drive our robot, we used two 12-volt, 300 RPM DC motors purchased from Amazon. Because these are simple DC motors, we needed a way to interface them with the Arduino. To do this, we used an L298N, a dual H-Bridge motor-driver which allows for speed and direction control of two DC motors at the same time. On the module, there are two screw terminal blocks for two motors, A and B, another screw terminal to power the H-Bridge/motors, and six input pins to control the speed and direction.

To control the speed of the motors, we used the enable A (ENA) and enable B (ENB) pins on the H-Bridge. These pins are controlled by pulse width modulation (PWM) and take an input range of 0 to 255, with 255 being full power and 0 being no power. Even though 0 is theoretically no power, the range of what values can move the motor is much smaller. We found that the lowest value that was able to move the wheels/robot was around 80. To control the direction of the motors, we used four pins; IN1 and IN2 to control motor A and IN3 and IN4 to control motor B. When pin IN1 is HIGH and IN2 is LOW, the motor moves in one direction.

When IN1 is LOW and IN2 is HIGH, the motor moves in the opposite direction. The same applies for IN3 and IN4. What direction the motor moves when one pin is high, and the other is low depends on how you connect the power and ground of the motor. In our case, we plugged in the motor, connected the pins to the Arduino, and tested each case to figure out what made the motors move forwards and backwards. We made sure to connect the motors in a way so both motors would move forward when the odd pins were HIGH (1 and 3) and move backwards when the even pins were HIGH (2 and 4). Now that we understood how to control the speed and direction of the motors, we began developing the main code to control the robot through the maze.

Maze Navigation

To successfully navigate through the maze, we developed a controller to allow the robot to follow along the right wall with a fail-safe in case it ran into the walls. We chose to create a proportional derivative (PD) controller using data from two sensors, one IR sensor and one ultrasonic sensor. The IR sensor was set at about a 45º angle on the back left side of the robot. The ultrasonic sensor was at the front of the robot facing directly forwards. The fail-safe was triggered by a limit switch on the right front side of the robot.

We started off by getting the robot to follow along a straight wall using data from the IR sensor. With a PD controller, you have four variables you must know to calculate the gain. In this case, the gain represents how much you should increase/decrease the motor speeds. The four variables are P, D, Kp, and Kd. P is the proportional term, or the calculated error. For wall following, the error is represented by the difference between how close the robot should be to the wall and how close it is. D is the derivative term, or the calculated error minus the previously calculated error. Kp and Kd are the proportional and derivative gain values, respectively. These are variables whose values we chose to achieve the desired performance of the robot. We chose to find said values through trial and error. After many trials, we found the best values to be Kp = 3, and Kd = 55.

Next, we moved on to getting the robot to make a U-turn, or an outer corner turn. Please see **Figure 6** below for a visual representation of a U-turn.

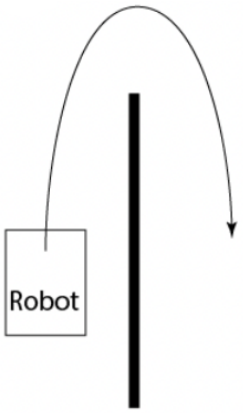


Figure 6. U-turn at the end of a wall

To start, we let the robot try and make the outer turn without making changes to the code. On the first try, the robot took off straight ahead when it no longer saw the wall. After some investigating, we discovered two things. One, when the IR sensor saw no wall, it read an infinitely large value, making the gains large. Two, because the gain was large, it was causing the new calculated motor speeds from the gains to go below 0 and above 255. This was problematic because the motors will stall if they are given a value outside of the allowed PWM range. The fix this we limited the maximum value the IR sensor could see to 80cm, we limited the maximum speed of the motors to 255, and we calculated the absolute value of the motor speeds. Once these changes were made, we tested the robot again and it was able to turn the corner. However, the right wheel got stuck. To remedy this, we moved the IR sensor from the front of the robot to the back. Because the wheels of our robot were at the back of the robot, having the sensor at the front caused it to detect no wall before the robot had cleared it. By making this change, the robot successfully made the outer turn.

The final step of the wall following was to tackle the inner corner turn. Please see **Figure *7*** below for a visual representation of an inner corner turn.

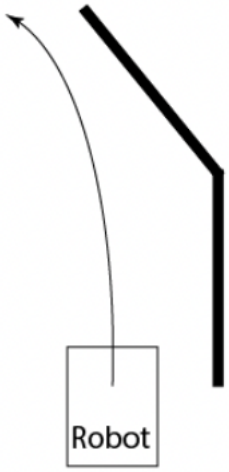


Figure 7. Turn at a corner

The same way we first tested the outer corner, we let the robot attempt the inner corner turn without changing the code. On the first run, the robot started to make the turn, but did not do so fast enough and crashed into the wall. To fix this, we added an ultrasonic sensor to the front of the robot to detect the distance to a front facing wall. With this sensor value, we implemented a condition saying, if the ultrasonic sensor detected a forward distance less than 15cm, make a zero-point left turn. Once we implemented this condition and tested the robot again, it made the inner corner turn flawlessly.

Even though the wall following was working well, the right-side wheel would sometimes get stuck on the side wall and/or the U-turn corners in the practice maze. When this happened, the wheels would stall, and our robot had no way to get unstuck. Our solution was to add a limit switch. This switch acted as an interrupt in the code and indicated, if the switch gets triggered, you have hit a wall on your right side. Back up, turn left away from the wall, then continue wall following. Once we implemented this, the robot was able to get out of the test maze nearly every time without getting stuck. For the final code used to navigate the robot through the maze, please see **Figure A-1** in Appendix A.

Results

During the competition, the robot had a few hiccups in performance. On the first run, the robot navigated very well through the maze, collecting a tennis ball at the beginning, and lining up to collect another one right at the end. However, when the robot rounded the final outer corner, instead of following the wall, it ran straight into the wall, getting stuck. This was unprecedented because the robot handled all the other outer corners very well. In between runs, we went to the hallway to test the algorithm again and it worked great. Therefore, we concluded that it was a fluke and decided to do the second maze run without changing any code. This was the right call because on the second run, the robot was able to collect two balls and get out in a timely manner. The robot could have been faster, but it was hitting the side wall more often than usual because the IR sensor angle had shifted between runs. For the final run, since we already had a score we were satisfied with, we made some quick modifications to try and get as many tennis balls as possible. These included extending the arm longer to grab the balls that were farther away from the wall than anticipated and adding Velcro to the arm to grab the balls. The robot was moving well through the maze until it managed to hit a corner on the only part of the right side that could not trigger the limit switch. In the end, there were certainly parts of the robot that could have been better, but we were still happy with how it performed.

Conclusion and Discussion

For the final lab competition, we successfully designed, built, and implemented a robot capable of autonomously navigating through a maze while collecting tennis balls. The process

to get to the final product was not without struggle. One issue we ran into was that our original chassis design was too large, and it could not navigate through the maze without getting stuck. Luckily, we discovered this issue with enough time to design and laser cut a new chassis with the left-over acrylic from the original materials we ordered. **Figure 8** below shows a before and after picture of the chassis size change.

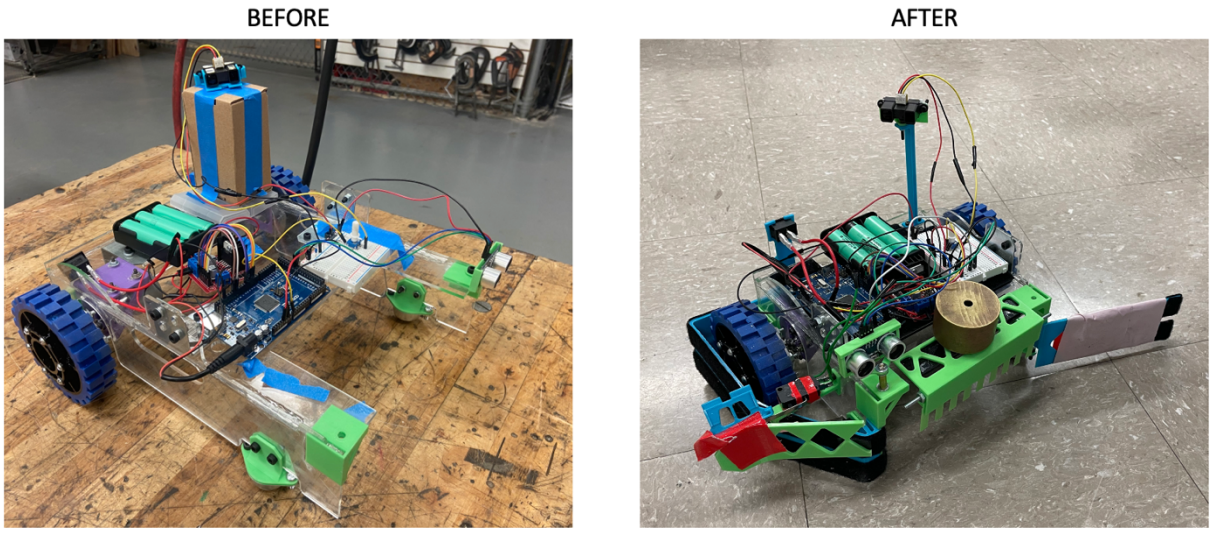


Figure 8. Before and after picture of the changes made to the chassis size

If we were to complete this project again or had more time to work on it, there are two main things we would focus on. First, we would spend more time testing the limit switch to find its optimal placement on the robot, as well as 3D print a better housing. This was a last minute add on to the robot, so we did not have enough time to thoroughly test all aspects of the device. Second, we would work on reducing the initial weight of the robot so that when tennis balls are attached to the Velcro arm, it would not slow the robot down significantly. All in all, we genuinely enjoyed this lab and how all our work throughout the semester guided us to being successful in the end.

Appendix A

Source code.

Figure A-1. Final maze navigation

```
/*
  ME 545 Final Lab Competition - Maze navigation
  Juliette Mitrovich, Sheila Moroney, Sujani Patel
*/
// Libraries
#include <math.h>

///// ULTRASONIC SENSOR /////
// Ultrasoinic sensor pins
const int echoPin_fwd = 32;
const int trigPin_fwd = 33;

// Ultrasonic sensor calculation variables
long duration_fwd;
float distance_fwd;

// Ultrasonic range condition
const float tooClose_fwd = 26; // how close the robot can get to a front facing wall

// FUNCTION //
float senseUltrasonic_frontWall() {
 // calculate the distance in front of the robot (cm)
 digitalWrite(trigPin_fwd, LOW);
 delayMicroseconds(2);
 digitalWrite(trigPin_fwd, HIGH);
 delayMicroseconds(10);
 digitalWrite(trigPin_fwd, LOW);
 duration_fwd = pulseIn(echoPin_fwd, HIGH);
 distance_fwd = duration_fwd * 0.034 / 2; // convert duration to distance
 // limit how far the front sensor can see because we only care about the close values for
turning
 if (distance_fwd > 35) {
  distance_fwd = 35;
 }
```

```
  return distance_fwd;
}

///// IR SENSOR /////
// IR sensor 1 variables
float IRsensor = A0; // set IRsensor as analog pin 0
float sensorValue; // variable to store the output of IR sensor

// digital filter coefficients, (fc = 27.77Hz, fs = 125Hz )
const float a1 = -0.0875;
const float b0 = 0.4563;
const float b1 = 0.4563;

// digital filter variables
volatile float Vold = 0; // initialize the first input
volatile float yold = 0; // initialize the first output
volatile float digitalFilter; // initialize variable to store first order digital filter values
volatile float distance_right; // variable to store the distance conversion of the IR sensor

// FUNCTION //
float senseIR() {
  // Collect IR sensor data (through and analog filter
  sensorValue = analogRead(IRsensor);
  // digitially filter IR sensor data
  digitalFilter = (b0 * sensorValue) + (b1 * Vold) - (a1 * yold);
  yold = digitalFilter;
  Vold = sensorValue;
  // Convert filtered IR sensor data to distance
  distance_right = (16569.0 / (digitalFilter + 25)) - 11;
  // limit the max range the sensor can see to avoid large gains
  if (60 < distance_right) {
    distance_right = 60;
  }
  return distance_right;
}

///// PROPORTIONAL DERIVATIVE (PD) CONTROL /////
const float setPoint_side = 39; // always want to be ~15cm from the side wall
```

```
// number is > 15 because the sensor is on the back left side and this is a right wall following robot
// allows you to stay within the correct operating range of the IR sensor (18cm - 150cm)

volatile float P; // proportional term
volatile float D; // derivative term
float Kp = 3.0; // proportional gain
float Kd = 55.0; // derivative gain

volatile float error_side; // variable to store calculated error
volatile float lastError_side; // variable to store the previous error for the derivative term

volatile float motorSpd_gain; // calculated gain from PID control

// FUNCTION //
float PD_control() {
  // find the error and previous error and calculate the necessary gain for the motor speeds
  error_side = setPoint_side - distance_right; // error calculation
  P = error_side;
  D = error_side - lastError_side;
  lastError_side = error_side;
  motorSpd_gain = P * Kp + D * Kd;
  return motorSpd_gain;
}

///// H-BRIDGE AND MOTORS /////
const float avgSpd = 110.0; // set speed motors should move at when going straight
volatile float rightSpd;
volatile float leftSpd;

// Motor pins
const int leftMotor1 = 5; // OUT 3 pin
const int leftMotor2 = 4; // OUT 4 pin

const int rightMotor1 = 3; // OUT 1 pin
const int rightMotor2 = 2; // OUT 2 pin

const int ENB_left = 10; // PWN pin to control left motor speed
```

```
const int ENA_right = 9; // PWM pin to control right motor speed

/*
   Pin convention
   Motor A: direction controlled by pins IN1 and IN2
   Motor B: direction controlled by pins IN3 and IN4
   if IN1 is HIGH and IN2 is low (or IN3 and IN4), motor will move one direction
   if you switch it the motor will move in the other direction DEPENDING ON HOW THE POWER
AND GROUND
   WIRES OF THE MOTOR ARE PLUGGED IN TO THE H-BRIDGE
*/

// FUNCTION //
void goFwd(float leftSpd, float rightSpd) {
  // set pins and speed to drive the robot forward (left fwd, right fwd)
  // convention for
  digitalWrite(leftMotor1, LOW);
  digitalWrite(leftMotor2, HIGH);
  digitalWrite(rightMotor1, LOW);
  digitalWrite(rightMotor2, HIGH);
  analogWrite(ENB_left, leftSpd);
  analogWrite(ENA_right, rightSpd);
}

void goBwd(float leftSpd, float rightSpd) {
  // set pins and speed to drive the robot backward (left bwd, right bwd)
  // convention for
  digitalWrite(leftMotor1, HIGH);
  digitalWrite(leftMotor2, LOW);
  digitalWrite(rightMotor1, HIGH);
  digitalWrite(rightMotor2, LOW);
  analogWrite(ENB_left, leftSpd);
  analogWrite(ENA_right, rightSpd);
}

void turnLeft(float leftSpd, float rightSpd) {
  // set pins and speed to allow robot to do a zero-point left turn (left fwd, right bwd)
  digitalWrite(leftMotor1, HIGH);
```

```
  digitalWrite(leftMotor2, LOW);
  digitalWrite(rightMotor1, LOW);
  digitalWrite(rightMotor2, HIGH);
  analogWrite(ENB_left, leftSpd);
  analogWrite(ENA_right, rightSpd);
}

///// LIMIT SWITCH /////
int L_switch_pin = 40; // limit switch input pin
int L_switch;

void setup() {
  Serial.begin(115200); // begin serial monitor for debugging

  // set ultrasonic sensor pin modes
  pinMode(echoPin_fwd, INPUT);
  pinMode(trigPin_fwd, OUTPUT);

  // set motor pin modes
  pinMode(rightMotor1, OUTPUT);
  pinMode(rightMotor2, OUTPUT);
  pinMode(leftMotor1, OUTPUT);
  pinMode(leftMotor2, OUTPUT);

  // set IR sensor pin mode
  pinMode(IRsensor, INPUT);

  // set limit switch pin mode
  pinMode(L_switch_pin, INPUT);
}

void loop() {
  // call the neccessary functions for the robot to complete wall following
  senseIR();
  senseUltrasonic_frontWall();
  PD_control();
  L_switch = digitalRead(L_switch_pin);
```

```
// set the speed of the wheels
// abs() used because the H-bridge does not allow negative speed values
rightSpd = abs(avgSpd + motorSpd_gain);
leftSpd = abs(avgSpd - motorSpd_gain);

// set the max speed of each motor to 255 because PWM range is 0-255
if (rightSpd > 255.0) {
  rightSpd = 255.0;
}
if (leftSpd > 255.0) {
  leftSpd = 255.0;
}

if (distance_fwd > tooClose_fwd) {
  // if you are not too close to the front wall, follow the right side wall
  Kp = 3.0;
  Kd = 60.0;
  goFwd(leftSpd, rightSpd);
}
if (distance_fwd < tooClose_fwd) {
  // when you come to an inner corner, do a zero point left turn at a set speed
  turnLeft(180, 180);
}
if (L_switch == LOW) {
  // if you hit the wall on the right side, back up, turn left slightly and drive forward
  delay(100);
  goBwd(150, 150);
  delay(500);
  turnLeft(110, 110);
  delay(200);
  goFwd(150, 150);
  delay(300);
}
```