OBSTACLE AVOIDANCE/MAZE SOLVER

Brian Rojas and Khushpreet Buttar

December 11, 2015

Merced College – Fall 2015
CPSC 42- 1737

Summary

This report shows the process behind a robot that has two functions. One of the functions it has is to avoid obstacles and the other function is to solve mazes. To build this we bought a kit for the chassis, to hold the necessary components. The brain of the robot is a microcontroller called Arduino Uno, which powers the motor shield and the servo for the head. This project helped us understand that what works in theory might not work in real life application, therefore you have to take into account as many variables as you can that might have an effect on the project.

Table of Contents

1.0 Introduction

Version 1 of The Robot's task is to avoid any obstacle in its way. Using the ultrasonic sensor to calculate the distance it can successfully move without crashing. Version 2 of The Robot is to calculate its surroundings and find the exit path of a maze using the left hand solution, where you can solve a maze by always turning left when you can.

2.0 Building the Robot

Building the robot was not too difficult, because of the kit that was bought. Assembling the head that would hold the servo was difficult, because we wanted it to be easily replaceable in case the servo would fail. There were only two parts we had to buy, the motor shield and the chassis kit. Rest of the parts we already had from previous projects. The parts we used included a SG90 Servo, Ultrasonic Sensor, L293D Motor Shield, and Makerfair robot car chassis kit.

Parts:

Servo

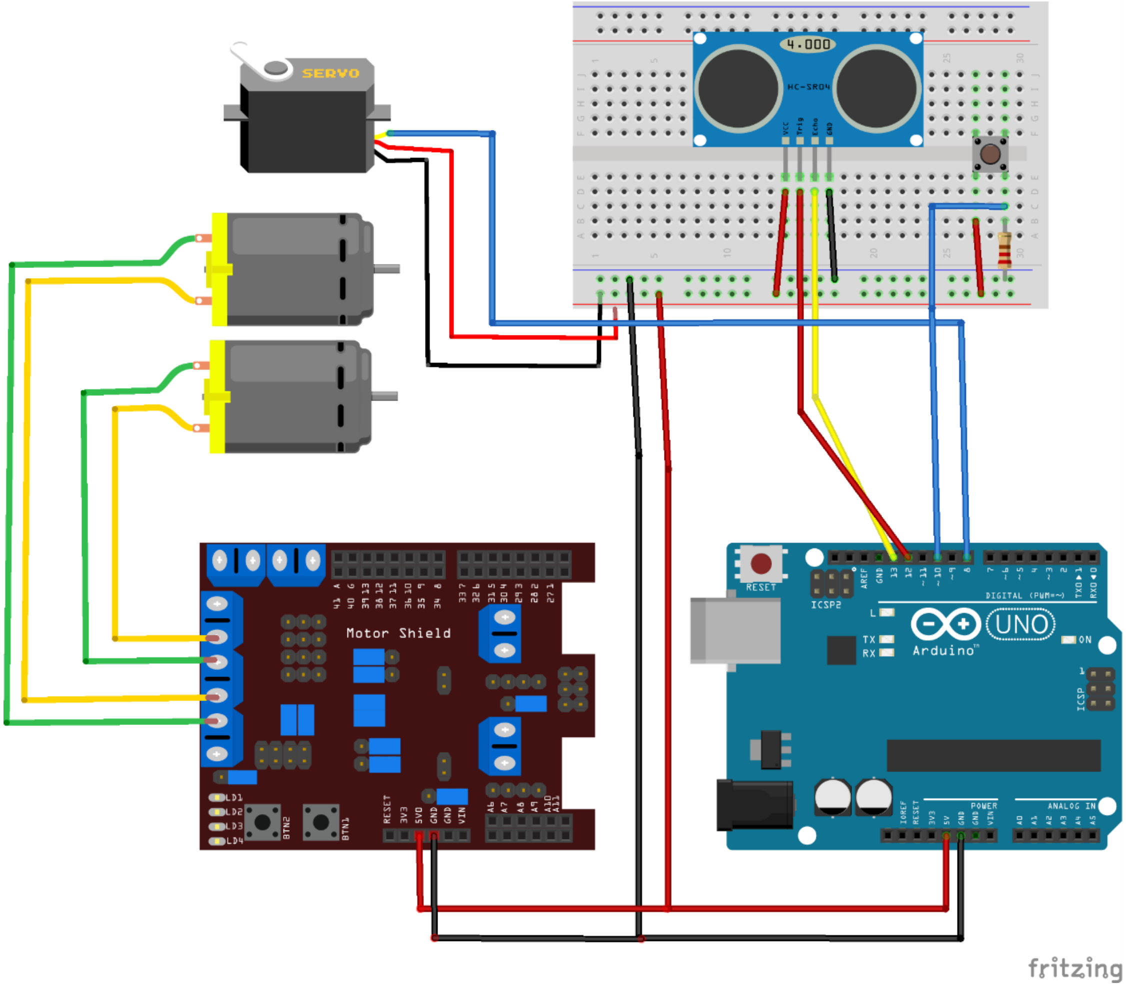
Ultrasonic Sensor

Motor Shield

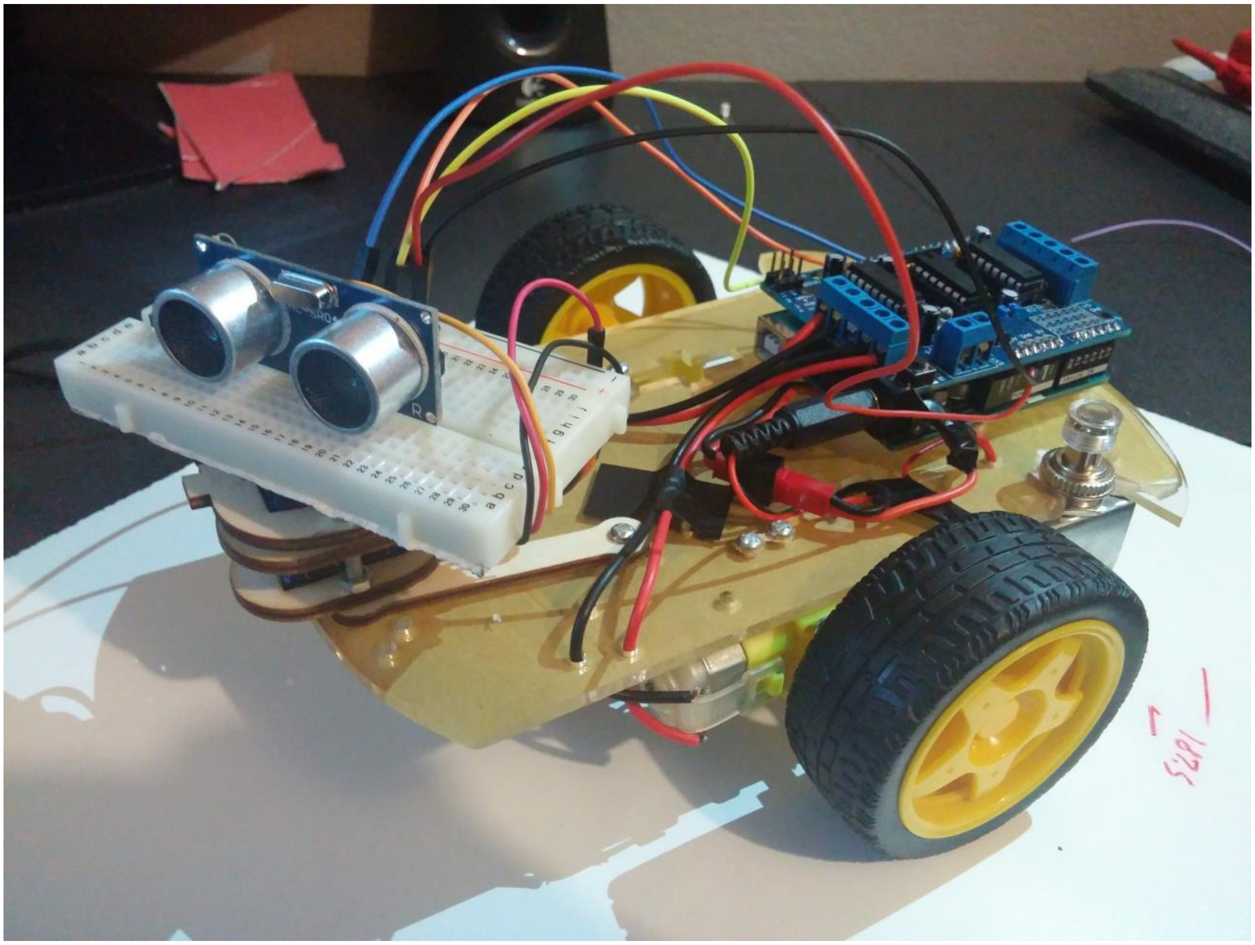
Chassis Kit

3.0 Building the Circuit

Once all the parts are accounted for, it was time to start building the circuit. A small breadboard was used as the circuit board. This way it was easier to switch out any broken wires, due to the head spinning.



4.0 Finished Product

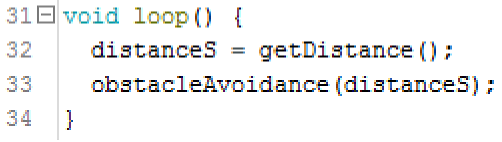


5.0 The Software

This project was basically two different projects, version 1 and version 2. Once version 1 was completed, we had time to think about what else can be done with the robot without making any major changes to the hardware, so we decided to program version 2 to solve mazes using the left hand rule.

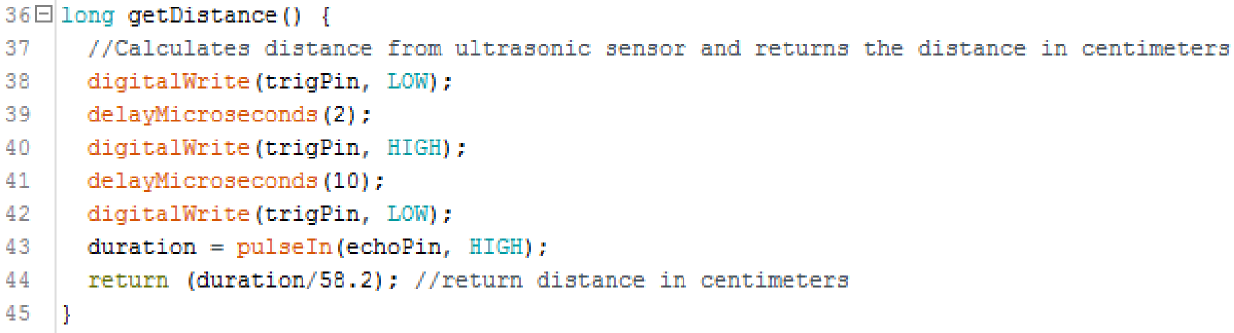
5.1 Version 1: Obstacle Avoidance

Main Loop

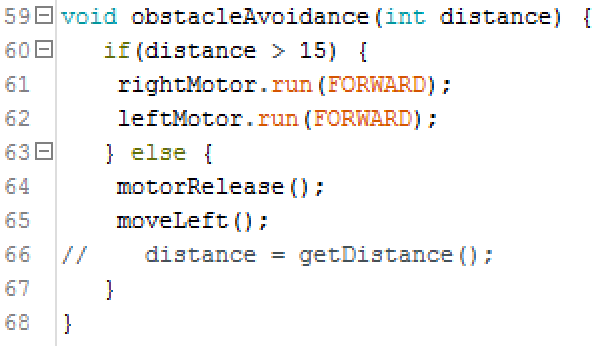


```
void loop() {
  distanceS = getDistance();
  obstacleAvoidance(distanceS);
}
```

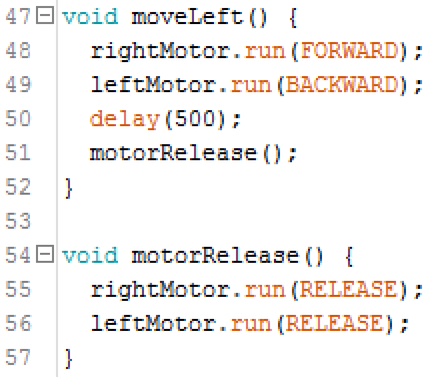
Functions



```
long getDistance() {
  //Calculates distance from ultrasonic sensor and returns the distance in centimeters
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  duration = pulseIn(echoPin, HIGH);
  return (duration/58.2); //return distance in centimeters
}
```



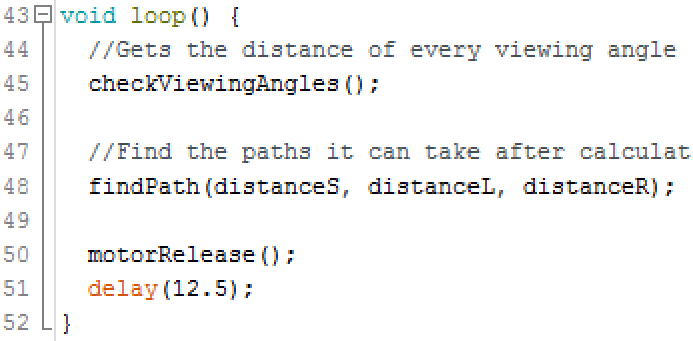
```
void obstacleAvoidance(int distance) {
   if(distance > 15) {
    rightMotor.run(FORWARD);
    leftMotor.run(FORWARD);
   } else {
    motorRelease();
    moveLeft();
//     distance = getDistance();
   }
}
```



```
void moveLeft() {
  rightMotor.run(FORWARD);
  leftMotor.run(BACKWARD);
  delay(500);
  motorRelease();
}

void motorRelease() {
  rightMotor.run(RELEASE);
  leftMotor.run(RELEASE);
}
```

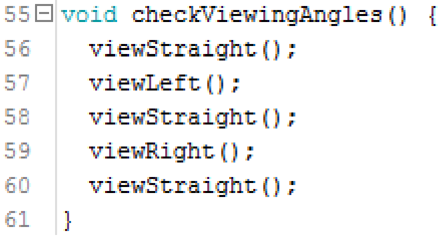
5.2 Version 2: Maze Solver

Main Loop

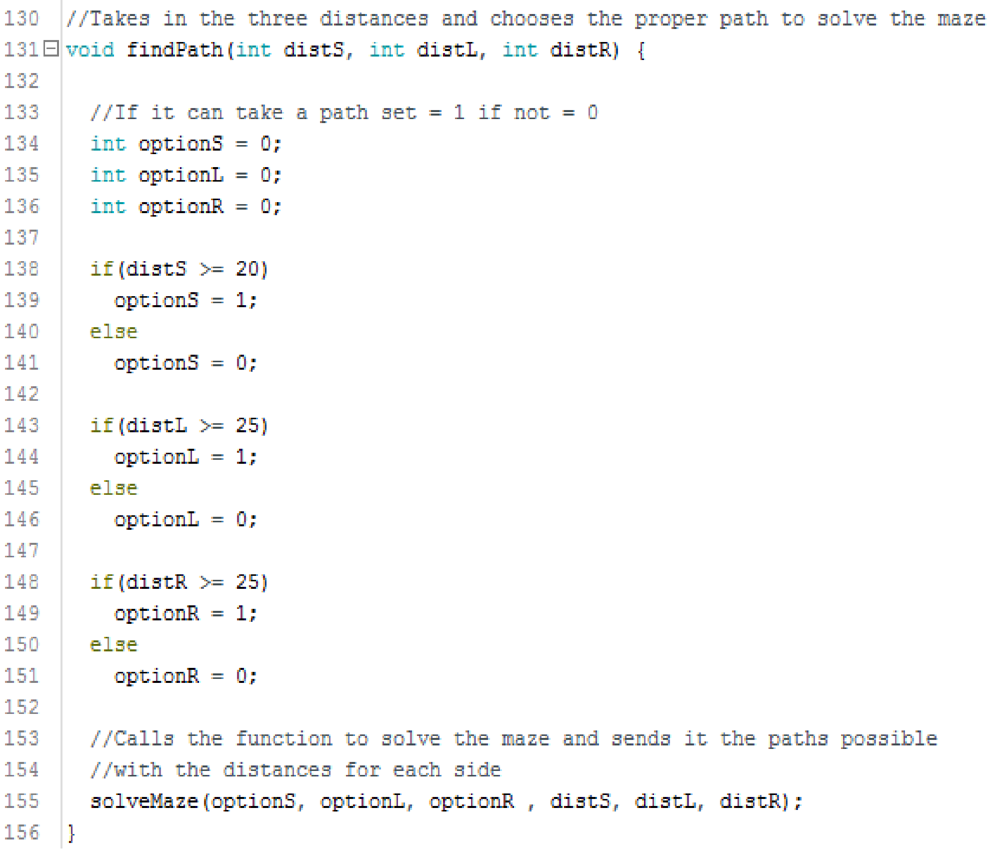


```
void loop() {
  //Gets the distance of every viewing angle
  checkViewingAngles();

  //Find the paths it can take after calculating the distance
  findPath(distanceS, distanceL, distanceR);

  motorRelease();
  delay(12.5);
}
```

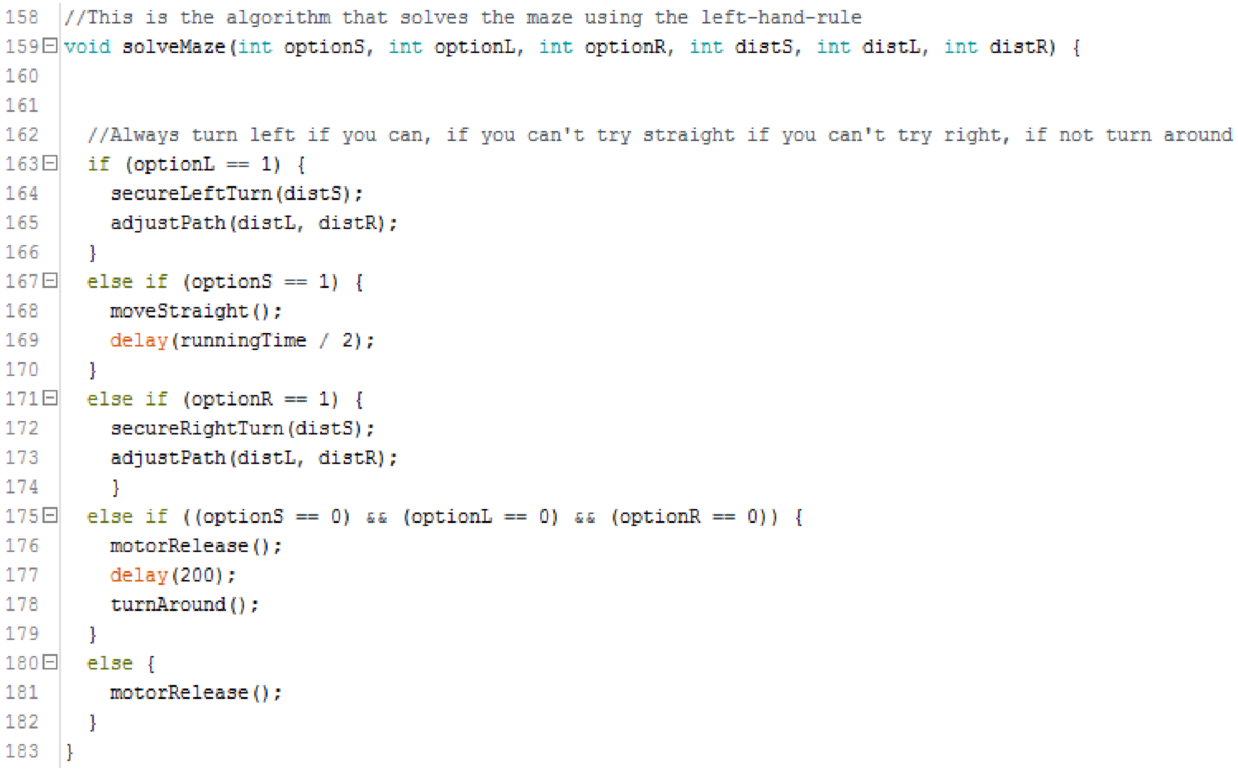
Functions



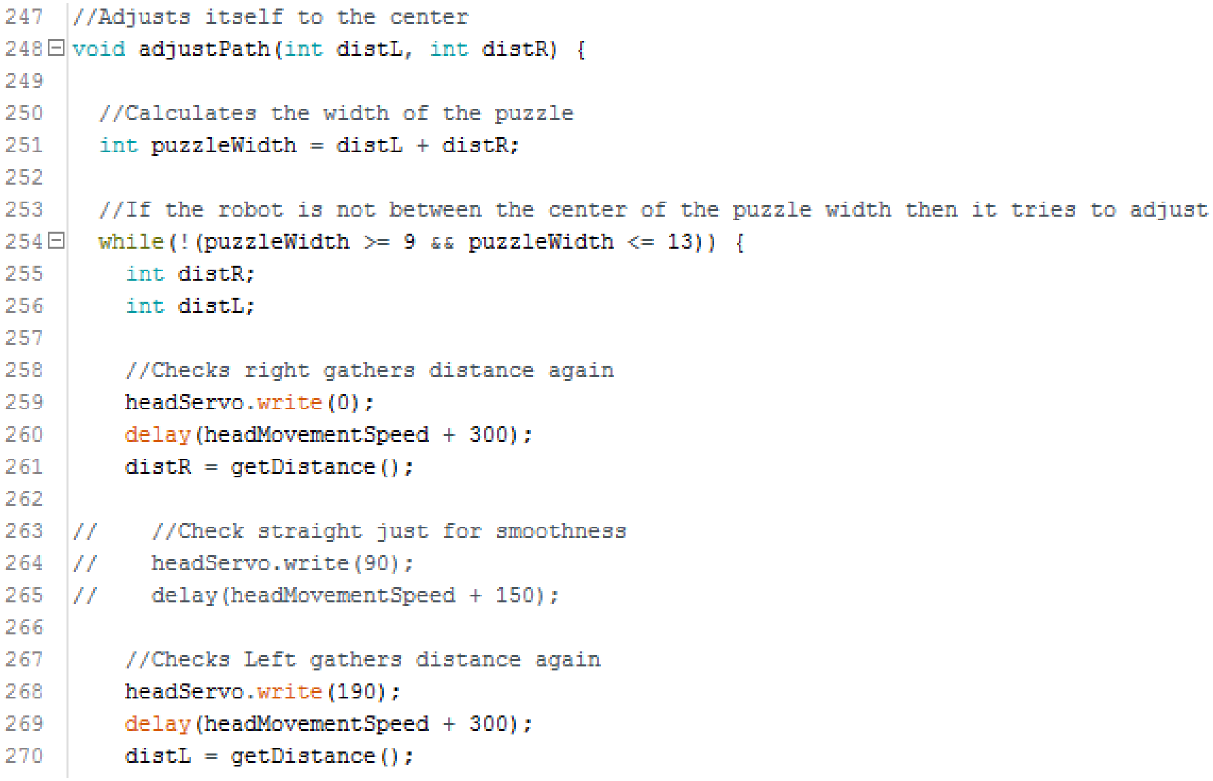
```
void checkViewingAngles() {
  viewStraight();
  viewLeft();
  viewStraight();
  viewRight();
  viewStraight();
}
```



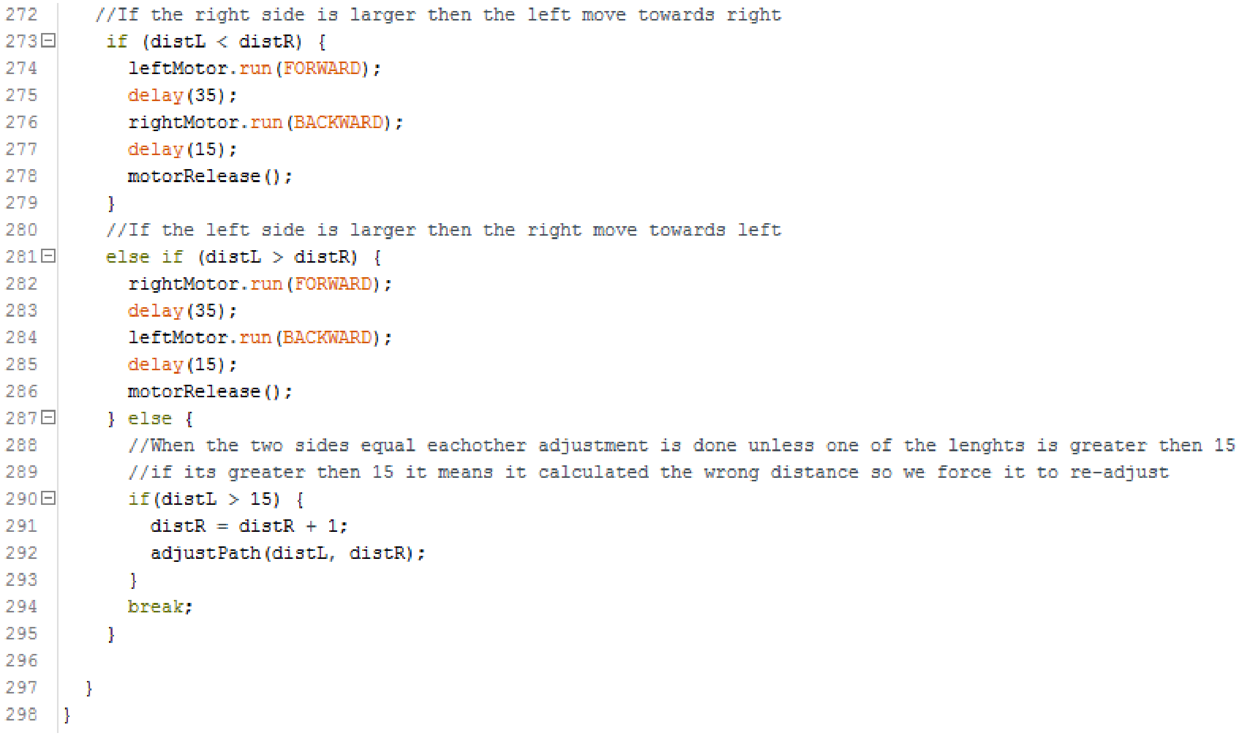
```
//Takes in the three distances and chooses the proper path to solve the maze
void findPath(int distS, int distL, int distR) {

  //If it can take a path set = 1 if not = 0
  int optionS = 0;
  int optionL = 0;
  int optionR = 0;

  if(distS >= 20)
    optionS = 1;
  else
    optionS = 0;

  if(distL >= 25)
    optionL = 1;
  else
    optionL = 0;

  if(distR >= 25)
    optionR = 1;
  else
    optionR = 0;

  //Calls the function to solve the maze and sends it the paths possible
  //with the distances for each side
  solveMaze(optionS, optionL, optionR , distS, distL, distR);
}
```



```
//This is the algorithm that solves the maze using the left-hand-rule
void solveMaze(int optionS, int optionL, int optionR, int distS, int distL, int distR) {


  //Always turn left if you can, if you can't try straight if you can't try right, if not turn around
  if (optionL == 1) {
    secureLeftTurn(distS);
    adjustPath(distL, distR);
  }
  else if (optionS == 1) {
    moveStraight();
    delay(runningTime / 2);
  }
  else if (optionR == 1) {
    secureRightTurn(distS);
    adjustPath(distL, distR);
    }
  else if ((optionS == 0) && (optionL == 0) && (optionR == 0)) {
    motorRelease();
    delay(200);
    turnAround();
  }
  else {
    motorRelease();
  }
}
```



```
//Adjusts itself to the center
void adjustPath(int distL, int distR) {

  //Calculates the width of the puzzle
  int puzzleWidth = distL + distR;

  //If the robot is not between the center of the puzzle width then it tries to adjust
  while(!(puzzleWidth >= 9 && puzzleWidth <= 13)) {
    int distR;
    int distL;

    //Checks right gathers distance again
    headServo.write(0);
    delay(headMovementSpeed + 300);
    distR = getDistance();

//    //Check straight just for smoothness
//    headServo.write(90);
//    delay(headMovementSpeed + 150);

    //Checks Left gathers distance again
    headServo.write(190);
    delay(headMovementSpeed + 300);
    distL = getDistance();
```



```
   //If the right side is larger then the left move towards right
    if (distL < distR) {
      leftMotor.run(FORWARD);
      delay(35);
      rightMotor.run(BACKWARD);
      delay(15);
      motorRelease();
    }
    //If the left side is larger then the right move towards left
    else if (distL > distR) {
      rightMotor.run(FORWARD);
      delay(35);
      leftMotor.run(BACKWARD);
      delay(15);
      motorRelease();
    } else {
      //When the two sides equal eachother adjustment is done unless one of the lenghts is greater then 15
      //if its greater then 15 it means it calculated the wrong distance so we force it to re-adjust
      if(distL > 15) {
        distR = distR + 1;
        adjustPath(distL, distR);
      }
      break;
    }

  }
}
```

6.0 Conclusion

It was a fun and educating project to finish. There were some problems that we had a really hard time with but, we managed to fix it with programming and hardware adjustments. A few things that would have made this project easier would be to use three ultrasonic sensors instead of one. This way we can always check the robots surroundings and make smoother movements. Another thing we could have done is use servos as wheels instead of dc motors, which would make more accurate turns. There are a lot of things that can still be improved in both hardware and code. Sometimes you have to keep in mind that if your project is going to be used as a real world application, make sure you account for any variables that might affect its overall performance.

7.0 References

McCabe, Patrick. *Instructables* . n.d. http://www.instructables.com/id/Maze-Solving-Robot/. 10 11 2015.