Lab 3: Obstacle Avoidance and Line Following Behaviors

ECE 564: Fundamentals of Autonomous Robots Lab
Team 1: Jacob Cassady, Chase Crutcher, and Olalekan Olakitan Olowo
August 28, 2019

The group members have worked together and face-to-face at all stages of this project work. The contributions of members to the report and to the codes are equal.
(Initials of group members)

Introduction

This lab was broken up into three parts. The first part of the lab is to develop an obstacle avoidance algorithm for the wallaby robot. The second part is to develop a line following algorithm. And lastly, the goal is to combine both of these algorithms into one. To accomplish these tasks, different sensors were added onto the wallaby robot and source code was developed in C.

Part 1: Obstacle Avoidance

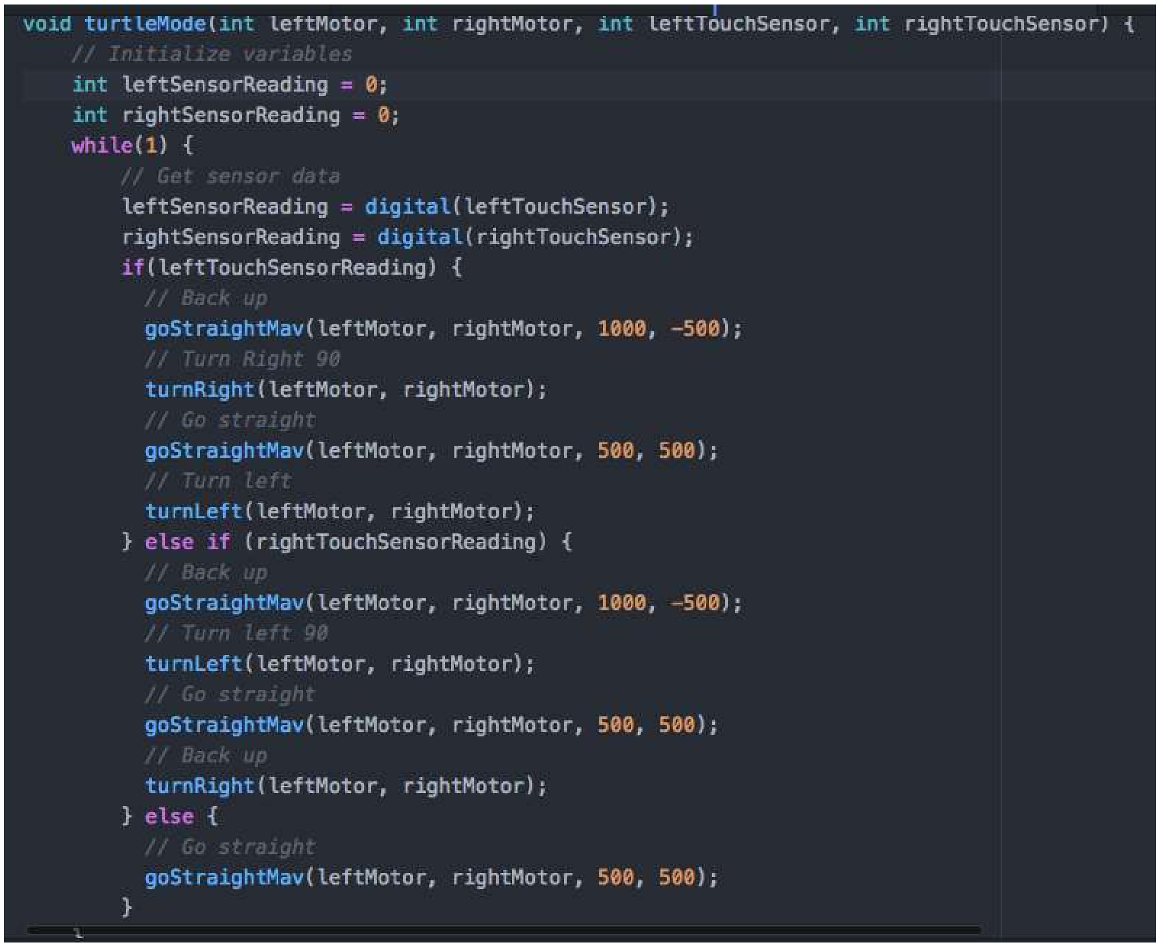
Algorithm Description

Part 1 of the lab focused on Obstacle avoidance. We added two touch sensors to the wallaby robot to develop a reactive system that moves to avoid objects. The touch sensor utilized is similar to the one shown in **Figure 1**.

Figure 1: Touch Sensor

To implement the object avoidance algorithm, we made use of several functions from previous labs. For an exhaustive list of functions and their implementations, please view the appendix section of this document.

The algorithm consists of a main while loop that is continuous. Data is retrieved from the touch sensors at the start of the loop. If the left touch sensor's reading is a 1, the robot is commanded to back up, turn to the right, go straight a little ways, and turn back to the left. Subsequently, if the right touch sensor's reading is a 1, the robot is commanded to back up, turn to the left, go straight a little ways, and turn back to the left. If neither of the touch sensors have a high reading, the robot is commanded to continue moving straight for 500 ms. This value dictates the frequency the robot is checking for responses from the touch sensors. For the exact implementation of the algorithm, please see **Figure 2** below.



```
void turtleMode(int leftMotor, int rightMotor, int leftTouchSensor, int rightTouchSensor) {
    // Initialize variables
    int leftSensorReading = 0;
    int rightSensorReading = 0;
    while(1) {
        // Get sensor data
        leftSensorReading = digital(leftTouchSensor);
        rightSensorReading = digital(rightTouchSensor);
        if(leftTouchSensorReading) {
          // Back up
          goStraightMav(leftMotor, rightMotor, 1000, -500);
          // Turn Right 90
          turnRight(leftMotor, rightMotor);
          // Go straight
          goStraightMav(leftMotor, rightMotor, 500, 500);
          // Turn left
          turnLeft(leftMotor, rightMotor);
        } else if (rightTouchSensorReading) {
          // Back up
          goStraightMav(leftMotor, rightMotor, 1000, -500);
          // Turn left 90
          turnLeft(leftMotor, rightMotor);
          // Go straight
          goStraightMav(leftMotor, rightMotor, 500, 500);
          // Back up
          turnRight(leftMotor, rightMotor);
        } else {
          // Go straight
          goStraightMav(leftMotor, rightMotor, 500, 500);
        }
```

Figure 2: turtleMode Algorithm

The algorithm displayed above follows the reactive paradigm. No planning is done, only sensing and acting. Furthermore, as a behavior it is reflexive. These classifications are due to quick nature of the program without any room for planning.

Exercise Solutions

Exercise 1

In order to avoid obstacles, the Demobot needs to turn away from them. For instance, if the right touch sensor is activated then that would let the Demobot know there is an obstacle on its right side. In order to avoid the obstacle on its right it would need to back up, turn to the left, move forward, turn back right and continue its path. If both sensors are activated, then that would imply the

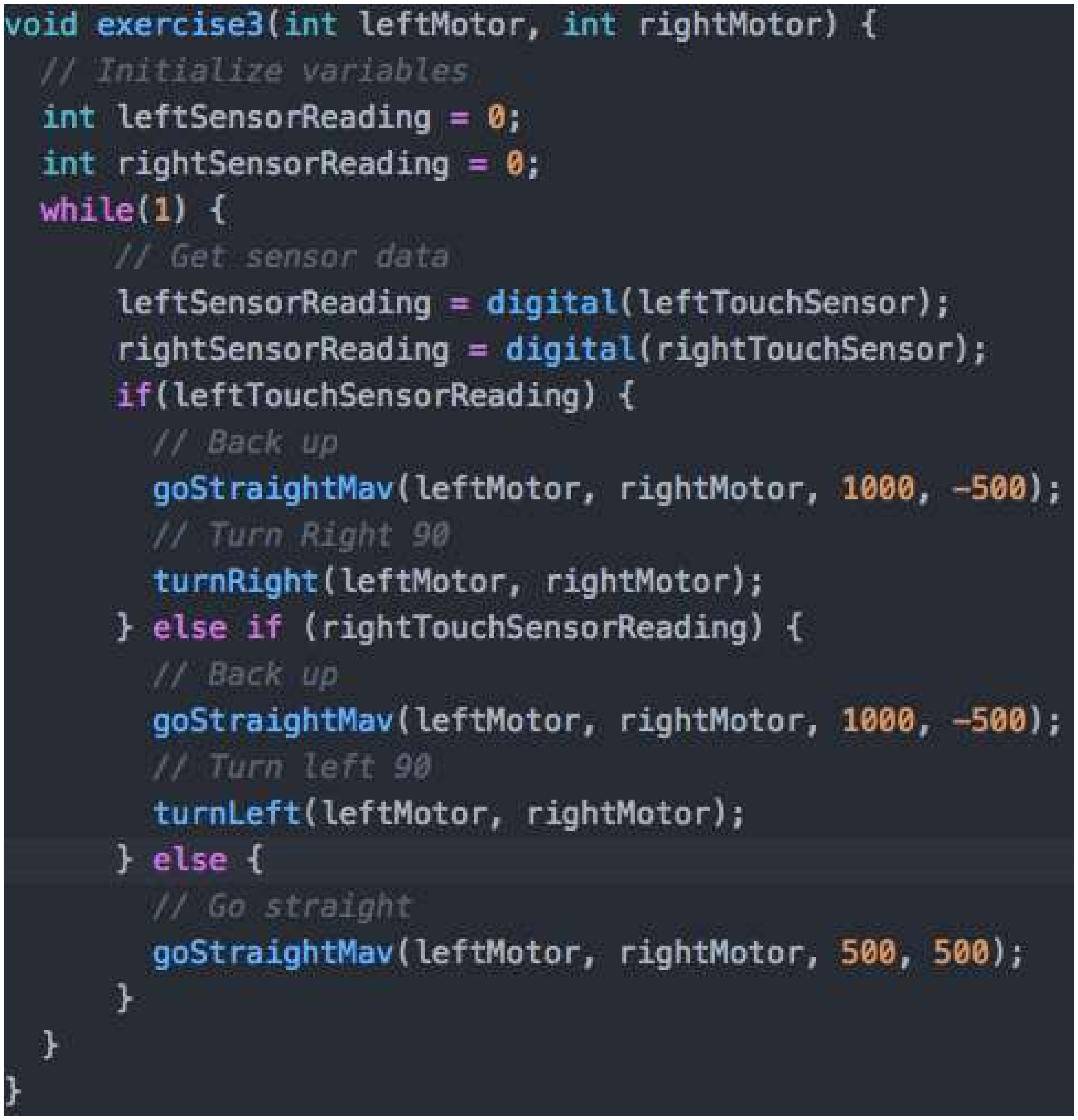
Demobot has hit a wall which would mean the Demobot needs to back up and choose to turn either left or right to change its path.

Exercise 2

A fix to turning in the wrong direction is to hard code that the Demobot will always make a left turn when the right sensor is activated and vis versa. Another method would be to adjust where the touch sensors are when having the problem of both activating at the same time. A solution would be to code the Demobot to randomly pick a direction to go, meaning it would choose left half the time and right the other half of the time. Another solution would be to take another look at the orientation of the motor plugs and consider alternating their polarity.

Exercise 3

Below is the code that was used to complete the obstacle avoidance exercise.



```
void exercise3(int leftMotor, int rightMotor) {
  // Initialize variables
  int leftSensorReading = 0;
  int rightSensorReading = 0;
  while(1) {
      // Get sensor data
      leftSensorReading = digital(leftTouchSensor);
      rightSensorReading = digital(rightTouchSensor);
      if(leftTouchSensorReading) {
        // Back up
        goStraightMav(leftMotor, rightMotor, 1000, -500);
        // Turn Right 90
        turnRight(leftMotor, rightMotor);
      } else if (rightTouchSensorReading) {
        // Back up
        goStraightMav(leftMotor, rightMotor, 1000, -500);
        // Turn left 90
        turnLeft(leftMotor, rightMotor);
      } else {
        // Go straight
        goStraightMav(leftMotor, rightMotor, 500, 500);
      }
  }
}
```

The code for exercise 3 does exactly what was discussed in parts 1 and 2 of this section. If the left touch sensor is hit, it moves back and rotates clockwise making a right turn. Likewise, if the right touch sensor is hit then the Demobot will back up and turn counterclockwise which is a left turn.

Exercise 4

When the Demobot was put into an open environment full of chairs, desks, and backpacks, it performed well enough depending on what it ran into. If it ran into a wall or cabinet, the Demobot would back up and turn to a new direction and continue its path. If it ran into something like a backpack, depending on the shape of the bag, it might run into it again even after backing up and trying to change direction. The Demobot rarely got stuck in an open environment which was due to touch sensors and obstacle avoidance code that was implemented into its microcontroller.

Part 2: Line Following

Part 2 of the lab's goal was to develop a line following program. To accomplish this, we affixed two IR proximity sensors to the front of the robot. These sensors are positioned as closely together as possible to simplify the algorithm's implementation. For reference of the IR proximity sensors used, please see **Figure 3** below.

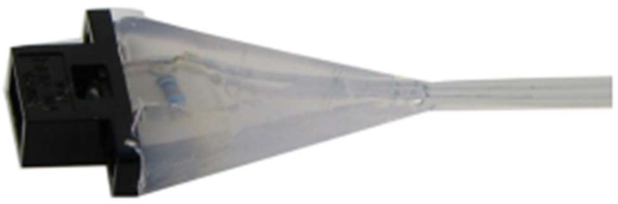
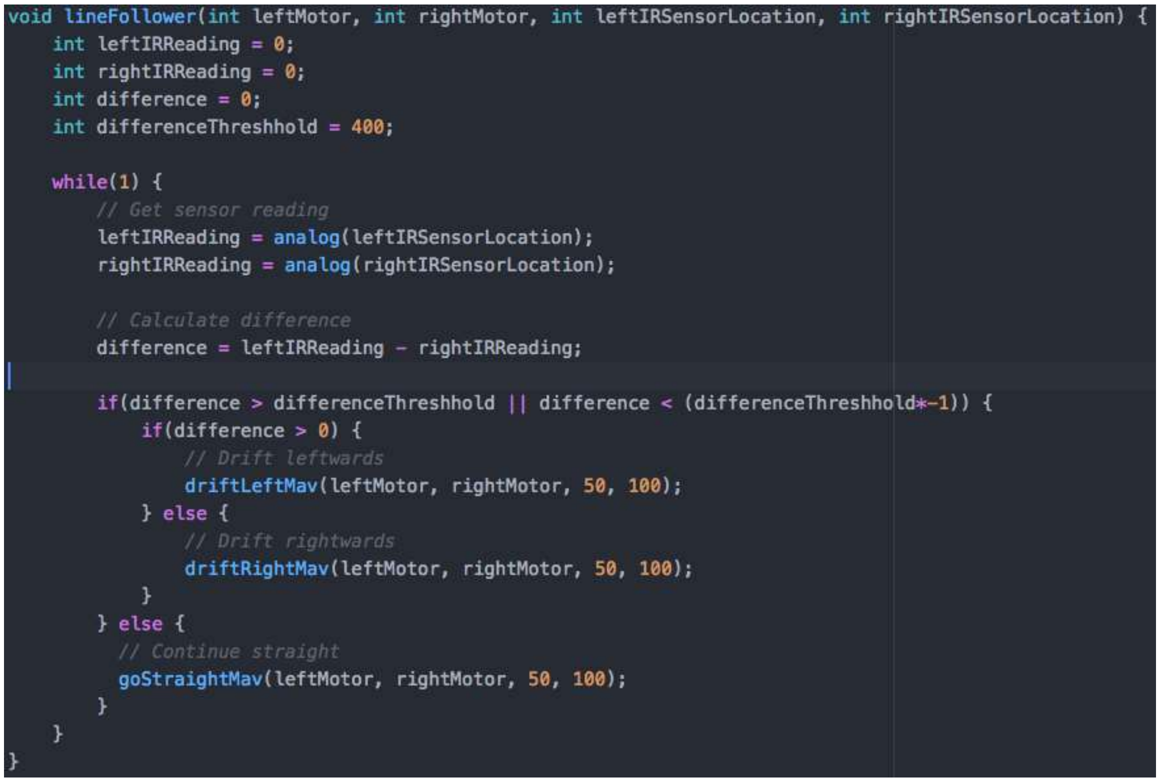


Figure 3 : IR Proximity Sensor

The line follower algorithm was implemented with a similar constant while loop. The algorithm uses a difference threshold to determine which direction is darker and moves towards that direction. Additionally, this difference threshold can be met from flat and shiny surfaces with shiny surfaces producing values of larger magnitude than flat surfaces.

The difference threshold approach was chosen because the line to be followed is said to be black while the rest of the scene is said to be white. If the value read on the left IR sensor is greater than that of the right IR sensor, it assumed the left side is viewing a more white canvas and the right side is viewing a more black canvas and vice versa if the readings are swapped. This difference is ignored until it is greater than 400, causing the robot to drift in the direction of the more black canvas. If the difference threshold hasn't been met, the robot continues forward. Please see **Figure 4** for reference of the algorithm's exact implementation.



```
void lineFollower(int leftMotor, int rightMotor, int leftIRSensorLocation, int rightIRSensorLocation) {
    int leftIRReading = 0;
    int rightIRReading = 0;
    int difference = 0;
    int differenceThreshhold = 400;

    while(1) {
        // Get sensor reading
        leftIRReading = analog(leftIRSensorLocation);
        rightIRReading = analog(rightIRSensorLocation);

        // Calculate difference
        difference = leftIRReading - rightIRReading;

        if(difference > differenceThreshhold || difference < (differenceThreshhold*-1)) {
            if(difference > 0) {
                // Drift leftwards
                driftLeftMav(leftMotor, rightMotor, 50, 100);
            } else {
                // Drift rightwards
                driftRightMav(leftMotor, rightMotor, 50, 100);
            }
        } else {
          // Continue straight
          goStraightMav(leftMotor, rightMotor, 50, 100);
        }
    }
}
```

Figure 4: Line Follower Algorithm

A working implementation could be done using a single IR proximity sensor instead of two. This would require the robot "panning" whenever it left a black region to find the direction back to the black region from its current location. This algorithmic approach was not chosen because it is less time efficient.

The robot was able to successfully traverse the track displayed in **Figure 5**. It was unable to pick up the edge though when approaching it from a perpendicular angle. This is because the sensors do not provide the precision to create the difference threshold and once the robot reaches a fully white surface again, it has an amicable difference threshold. On the bright side, the robot was able to follower lines of all colors using the difference threshold. This included the outside of the track's paper.

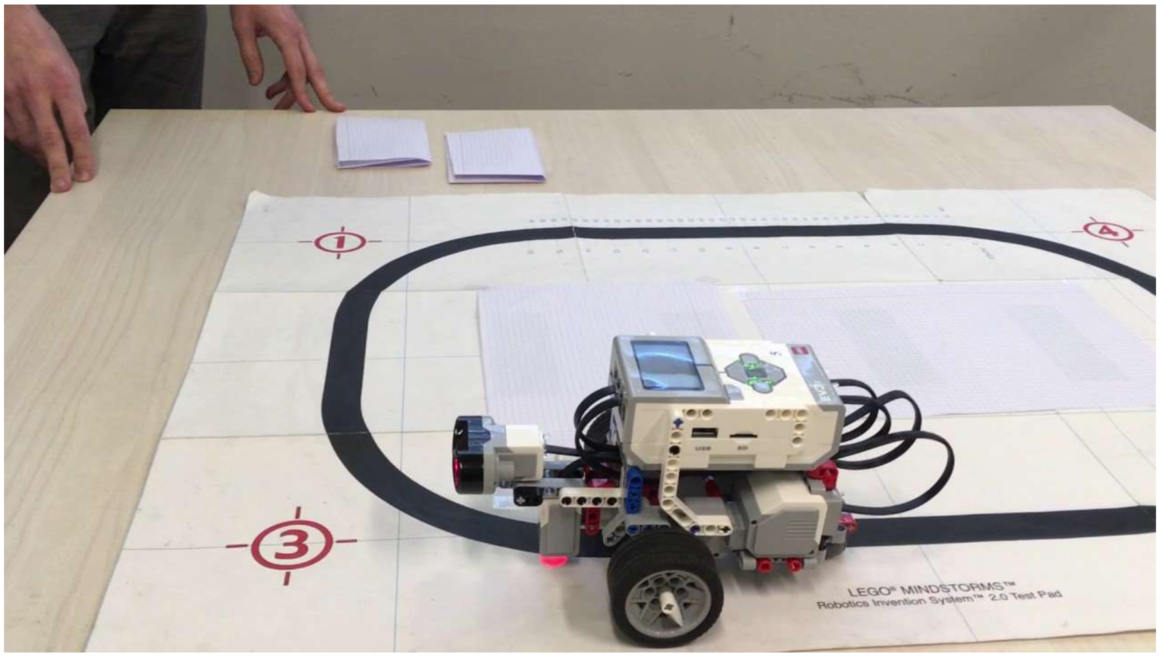


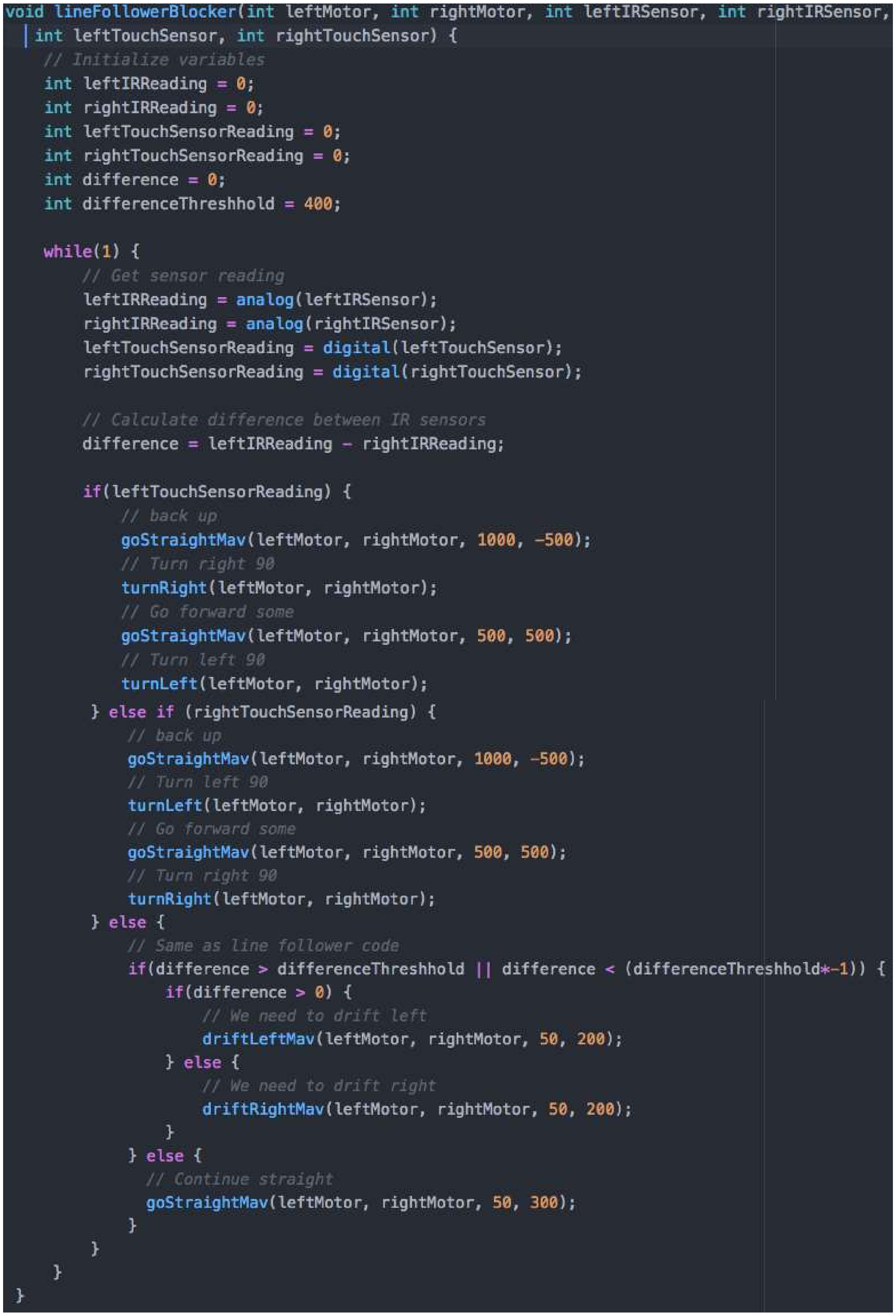
Figure 5: Line Follower Track

Part 3: Combine Obstacle Avoidance and Line Following Behaviors

The third part of the lab is to combine the obstacle avoidance and line following behaviors. To do this, no new hardware needed to be added. The control flows from both parts were combined into one with some tweaks to the line follower to improve precision.

The algorithm utilizes a constant while loop like the previous. It also, begins with getting data readings from its pertinent sensors. A difference is calculated from the IR proximity sensors similar to Part 2. Next, the touch sensors values are checked to see if they are 1. If they are, they perform the same routine described in Part 1 for their respective sides. If both are low, the algorithm continues with the algorithm described in Part 2 of this document. To refind the line once leaving, the robot continues moving forward indefinitely as it always turns inside of the closed track. For the exact implementation of obstacle avoidance/line follower algorithm please reference **Figure 6**.

The main differences between the algorithm in Part 3 and those in Parts 1 and 2 is the frequency at which commands are being sent to the robot as well as the speed of the robot when in the line follower section of the algorithm. Both of these changes were made to increase the precision of the line follower. Giving more frequent, lower in magnitude, drive commands to the robot allows for more sensor readings and therefore more chances for the robot to find the edge without running past it.



```
void lineFollowerBlocker(int leftMotor, int rightMotor, int leftIRSensor, int rightIRSensor,
  int leftTouchSensor, int rightTouchSensor) {
    // Initialize variables
    int leftIRReading = 0;
    int rightIRReading = 0;
    int leftTouchSensorReading = 0;
    int rightTouchSensorReading = 0;
    int difference = 0;
    int differenceThreshhold = 400;

    while(1) {
        // Get sensor reading
        leftIRReading = analog(leftIRSensor);
        rightIRReading = analog(rightIRSensor);
        leftTouchSensorReading = digital(leftTouchSensor);
        rightTouchSensorReading = digital(rightTouchSensor);

        // Calculate difference between IR sensors
        difference = leftIRReading - rightIRReading;

        if(leftTouchSensorReading) {
            // back up
            goStraightMav(leftMotor, rightMotor, 1000, -500);
            // Turn right 90
            turnRight(leftMotor, rightMotor);
            // Go forward some
            goStraightMav(leftMotor, rightMotor, 500, 500);
            // Turn left 90
            turnLeft(leftMotor, rightMotor);
         } else if (rightTouchSensorReading) {
             // back up
             goStraightMav(leftMotor, rightMotor, 1000, -500);
             // Turn left 90
             turnLeft(leftMotor, rightMotor);
             // Go forward some
             goStraightMav(leftMotor, rightMotor, 500, 500);
             // Turn right 90
             turnRight(leftMotor, rightMotor);
         } else {
             // Same as line follower code
             if(difference > differenceThreshhold || difference < (differenceThreshhold*-1)) {
                 if(difference > 0) {
                     // We need to drift left
                     driftLeftMav(leftMotor, rightMotor, 50, 200);
                 } else {
                     // We need to drift right
                     driftRightMav(leftMotor, rightMotor, 50, 200);
                 }
             } else {
               // Continue straight
               goStraightMav(leftMotor, rightMotor, 50, 300);
             }
         }
     }
 }
```

Figure 6: Line Follower & Obstacle Avoidance Algorithm

Conclusion

In conclusion, obstacle avoidance behavior can be accomplished with the use of touch sensors and line following behavior can be accomplished with the use of IR proximity sensors. Combination of both of the behaviors can be easily done as long as the frequency at which the sensors are polled is sufficient. Furthermore, when using a difference threshold as a point of importance in a control system, the closer the sensors are to each other allows for more precision.

Suggestions

Several students were placing their IR proximity sensors far apart from each other to look cool. A discussion on sensor placement could be helpful before starting this lab.

Appendix

Main.c

```
#include <kipr/botball.h>
#include "robo_move.h"

void driveForward(int leftMotorLocation, int rightMotorLocation, int velocity);
void turtleMode(int leftMotor, int rightMotor, int leftTouchSensor, int
rightTouchSensor);
void lineFollower(int leftMotor, int rightMotor, int leftIRSensorLocation, int
rightIRSensorLocation);
void lineFollowerBlocker(int leftMotor, int rightMotor, int leftIRSensor, int
rightIRSensor, int leftTouchSensor, int rightTouchSensor);

int main() {
   int leftMotor = 0;
   int rightMotor = 3;
   int leftIRSensorLocation = 0;
   int rightIRSensorLocation = 1;
   int leftTouchSensor = 0;
   int rightTouchSensor = 1;

   lineFollowerBlocker(leftMotor, rightMotor, leftIRSensorLocation,
rightIRSensorLocation, leftTouchSensor, rightTouchSensor);
   return 0;
}

void driveForward(int leftMotorLocation, int rightMotorLocation, int velocity) {
   mav(leftMotorLocation, velocity);
   mav(rightMotorLocation, velocity);
}

void turtleMode(int leftMotor, int rightMotor, int leftTouchSensor, int
rightTouchSensor) {
   // Initialize variables
   int leftSensorReading = 0;
   int rightSensorReading = 0;
   while(1) {
     // Get sensor data
     leftSensorReading = digital(leftTouchSensor);
     rightSensorReading = digital(rightTouchSensor);
     if(leftTouchSensorReading) {
      // Back up
      goStraightMav(leftMotor, rightMotor, 1000, -500);
      // Turn Right 90
      turnRight(leftMotor, rightMotor);
      // Go straight
      goStraightMav(leftMotor, rightMotor, 500, 500);
      // Turn left
      turnLeft(leftMotor, rightMotor);
     } else if (rightTouchSensorReading) {
      // Back up
      goStraightMav(leftMotor, rightMotor, 1000, -500);
      // Turn left 90
      turnLeft(leftMotor, rightMotor);
      // Go straight
      goStraightMav(leftMotor, rightMotor, 500, 500);
      // Back up
      turnRight(leftMotor, rightMotor);
     } else {
      // Go straight
      goStraightMav(leftMotor, rightMotor, 500, 500);
     }
   }
}

void lineFollower(int leftMotor, int rightMotor, int leftIRSensorLocation, int
rightIRSensorLocation) {
   int leftIRReading = 0;
   int rightIRReading = 0;
   int difference = 0;
   int differenceThreshhold = 400;

   while(1) {
     // Get sensor reading
     leftIRReading = analog(leftIRSensorLocation);
     rightIRReading = analog(rightIRSensorLocation);

     // Calculate difference
     difference = leftIRReading - rightIRReading;

     if(difference > differenceThreshhold || difference < (differenceThreshhold*-1)) {
         if(difference > 0) {
             // Drift leftwards
             driftLeftMav(leftMotor, rightMotor, 50, 100);
         } else {
             // Drift rightwards
             driftRightMav(leftMotor, rightMotor, 50, 100);
         }
     } else {
      // Continue straight
      goStraightMav(leftMotor, rightMotor, 50, 100);
     }
   }
}

void lineFollowerBlocker(int leftMotor, int rightMotor, int leftIRSensor, int
rightIRSensor,
   int leftTouchSensor, int rightTouchSensor) {
   // Initialize variables
   int leftIRReading = 0;
   int rightIRReading = 0;
   int leftTouchSensorReading = 0;
   int rightTouchSensorReading = 0;
   int difference = 0;
   int differenceThreshhold = 400;

   while(1) {
     // Get sensor reading
     leftIRReading = analog(leftIRSensor);
     rightIRReading = analog(rightIRSensor);
     leftTouchSensorReading = digital(leftTouchSensor);
     rightTouchSensorReading = digital(rightTouchSensor);

     // Calculate difference between IR sensors
     difference = leftIRReading - rightIRReading;

     if(leftTouchSensorReading) {
         // back up
         goStraightMav(leftMotor, rightMotor, 1000, -500);
         // Turn right 90
         turnRight(leftMotor, rightMotor);
         // Go forward some
         goStraightMav(leftMotor, rightMotor, 500, 500);
         // Turn left 90
         turnLeft(leftMotor, rightMotor);
     } else if (rightTouchSensorReading) {
         // back up
         goStraightMav(leftMotor, rightMotor, 1000, -500);
         // Turn left 90
         turnLeft(leftMotor, rightMotor);
         // Go forward some
         goStraightMav(leftMotor, rightMotor, 500, 500);
         // Turn right 90
         turnRight(leftMotor, rightMotor);
     } else {
         // Same as line follower code
         if(difference > differenceThreshhold || difference < (differenceThreshhold*-
1)) {
             if(difference > 0) {
                 // We need to drift left
                 driftLeftMav(leftMotor, rightMotor, 50, 200);
             } else {
                 // We need to drift right
                 driftRightMav(leftMotor, rightMotor, 50, 200);
             }
         } else {
           // Continue straight
           goStraightMav(leftMotor, rightMotor, 50, 300);
         }
     }
   }
}
```

Robo_move.h

```
#ifndef ROBO_MOVE_H
#define ROBO_MOVE_H

#include <kipr/botball.h>

void goDemobot(int leftMotor, int rightMotor, int millisecond_t, int powerLevel_l, int
powerLevel_r); // function prototype
void goDemobotMav(int leftMotor, int rightMotor, int millisecond_t, int leftVelocity,
int rightVelocity);
void goStraight(int leftMotor, int rightMotor, int millisecond_t, int powerLevel);
void goStraightMrp(int leftMotor, int rightMotor, int velocity, int numberOfTicks);
void goStraightMav(int leftMotor, int rightMotor, int millisecond_t, int velocity);
void make90Turn(int leftMotor, int rightMotor, int leftTurn, int powerLevel);
void turnLeft(int leftMotor, int rightMotor);
void turnRight(int leftMotor, int rightMotor);
void driftLeftMav(int leftMotor, int rightMotor, int millisecond_t, int velocity);
void driftRightMav(int leftMotor, int rightMotor, int millisecond_t, int velocity);
void makeCircle(int leftMotor, int rightMotor);
void figureEight(int leftMotor, int rightMotor);

#endif
```

Robot_move.c

```
#include "robo_move.h"

void goDemobot(int leftMotor, int rightMotor, int millisecond_t, int powerLevelLeft,
int powerLevelRight) {
   motor(leftMotor, powerLevelLeft * 0.925);
   motor(rightMotor, powerLevelRight);
   wait_for_milliseconds(millisecond_t);
   ao();
}
// end goDemobot

void goDemobotMav(int leftMotor, int rightMotor, int millisecond_t, int leftVelocity,
int rightVelocity) {
  mav(leftMotor, leftVelocity*0.944);
  mav(rightMotor, rightVelocity);
  msleep(millisecond_t);
  ao();
}

void goStraight(int leftMotor, int rightMotor, int millisecond_t, int powerLevel) {
   goDemobot(leftMotor, rightMotor, millisecond_t, powerLevel, powerLevel);
}

void goStraightMrp(int leftMotor, int rightMotor, int velocity, int numberOfTicks) {
   mrp(leftMotor, velocity*0.944, numberOfTicks);
   mrp(rightMotor, velocity, numberOfTicks);
   msleep(velocity*numberOfTicks);
}

void goStraightMav(int leftMotor, int rightMotor, int millisecond_t, int velocity) {
   mav(leftMotor, velocity*0.944);
   mav(rightMotor, velocity);
   msleep(millisecond_t);
   ao();
}


void make90Turn(int leftMotor, int rightMotor, int leftTurn, int powerLevel) {
   if(leftTurn) {
      /// Turn 90 degress Left
      goDemobot(leftMotor, rightMotor, 2000 , 0, powerLevel);
   } else {
             /// Turn 90 degrees Right
      goDemobot(leftMotor, rightMotor, 2000, powerLevel, 0);
   }
}

void turnLeft(int leftMotor, int rightMotor) {
   // Turn left
          make90Turn(leftMotor, rightMotor, 1, 63);
}

void turnRight(int leftMotor, int rightMotor) {
   // Turn right
          make90Turn(leftMotor, rightMotor, 0, 63);
}

void driftLeftMav(int leftMotor, int rightMotor, int millisecond_t, int velocity) {
  goDemobotMav(leftMotor, rightMotor, millisecond_t, velocity/30, velocity);
}


void driftRightMav(int leftMotor, int rightMotor, int millisecond_t, int velocity) {
  goDemobotMav(leftMotor, rightMotor, millisecond_t, velocity, velocity/30);
}

void makeCircle(int leftMotor, int rightMotor) {
   goDemobot(leftMotor, rightMotor, 10370, 60, 10);
}

void figureEight(int leftMotor, int rightMotor) {
   goDemobot(leftMotor, rightMotor, 5185, 60, 10);
   goDemobot(leftMotor, rightMotor, 10170, 10, 60);
   goDemobot(leftMotor, rightMotor, 5185, 60, 10);
}
```