ECGR 4161/5196 Lab 8 Report Due 08/08/21

Name 1: Joshua Ayers
Student email ID jayers18

Lab Objective:
The objective of the lab was to complete 2 tasks. The first was to follow a line and stop on top of a T. Next the robot is to navigate through a maze in the most efficient method given the knowledge that the longer distance at the first T junction leads to the exit.

Lab Questions or Figures:

Line following reaction table						
6800>	6000 > <5000	5000> 4000	>4000 and <3000	2000	1000<	200<
stop	hard left correction	light left correction	no correction	light right correction	hard right correction	stop

State table for the decision algorithm of the robot

Left scan	Front scan	action
yes	yes	Turn Right
yes	no	Follow wall
no	yes	Turn left
no	no	Drive forward and scan

Commentary and Conclusion:
This lab was the most challenging of the course to complete. The Line following segment was simple, only minimal adjustments were made to account for the sharp turn as well as implementing a region that would stop the robot when it encountered the T. However the navigation code was while algorithmically simple to design very difficult to implement functionally. Ultimately while my robot was able to avoid the walls it was not able to navigate through the maze because it needed extensive threshold tuning.

Lab Code:

```
#include <Servo.h>
```

```
#include "SimpleRSLK.h"

//ultrasonic sensor setup
const int trigPin = 32;          //connects to the trigger pin on the distance sensor
const int echoPin = 33;          //connects to the echo pin on the distance sensor
float distance = 0;         //connects to the echo pin on the distance sensor

//LED setup
#define RED 75
#define GREEN 76
#define BLUE  77

// Linefollower setup
uint16_t sensorVal[LS_NUM_SENSORS];
uint16_t sensorCalVal[LS_NUM_SENSORS];
uint16_t sensorMaxVal[LS_NUM_SENSORS];
uint16_t sensorMinVal[LS_NUM_SENSORS];

bool isCalibrationComplete = false;

// Buttion setup



// servo setup
Servo myservo;

void setup() {
  Serial.begin(9600);
  setupRSLK();
  pinMode(trigPin, OUTPUT);   //the trigger pin will output pulses of electricity
  pinMode(echoPin, INPUT);
  setupLed(RED_LED);
  pinMode(76, OUTPUT);  //RGB Green LED possible pinMode variables -> P2.1 -> 76 -> GREEN_LED
  pinMode(77, OUTPUT);  //RGB Blue LED possible pinMode variables -> P2.2 -> 77 -> BLUE_LED
  myservo.attach(38);   // attaches the servo on Port 2.4 (P2.4 or pin 38)to the servo object
  myservo.write(90);
  setupWaitBtn(LP_LEFT_BTN);

}

void floorCalibration() {
  /* Place Robot On Floor (no line) */
  delay(2000);
  String btnMsg = "Push left button on Launchpad to begin calibration.\n";
  btnMsg += "Make sure the robot is on the floor away from the line.\n";
  /* Wait until button is pressed to start robot */
  waitBtnPressed(LP_LEFT_BTN,btnMsg,RED_LED);

  delay(1000);

  Serial.println("Running calibration on floor");
  simpleCalibrate();
  Serial.println("Reading floor values complete");

  btnMsg = "Push left button on Launchpad to begin line following.\n";
  btnMsg += "Make sure the robot is on the line.\n";
  /* Wait until button is pressed to start robot */
  waitBtnPressed(LP_LEFT_BTN,btnMsg,RED_LED);
  delay(1000);

  enableMotor(BOTH_MOTORS);
}

void simpleCalibrate() {
  /* Set both motors direction forward */
  setMotorDirection(BOTH_MOTORS,MOTOR_DIR_FORWARD);
  /* Enable both motors */
  enableMotor(BOTH_MOTORS);
  /* Set both motors speed 20 */
  setMotorSpeed(BOTH_MOTORS,20);

  for(int x = 0;x<100;x++){
    readLineSensor(sensorVal);
    setSensorMinMax(sensorVal,sensorMinVal,sensorMaxVal);
  }

  /* Disable both motors */
  disableMotor(BOTH_MOTORS);
}

void line_follow()
{
  uint16_t normalSpeed = 10;
  uint16_t midspd = 25;
  uint16_t fastSpeed = 30;

  /* Valid values are either:
   *  DARK_LINE  if your floor is lighter than your line
   *  LIGHT_LINE if your floor is darker than your line
   */
  uint8_t lineColor = DARK_LINE;

```

```
/* Run this setup only once */
if(isCalibrationComplete == false) {
 floorCalibration();
 isCalibrationComplete = true;
}
 while(true){
 readLineSensor(sensorVal);
 readCalLineSensor(sensorVal, sensorCalVal,sensorMinVal,sensorMaxVal,lineColor);

 digitalWrite(GREEN, HIGH);
 uint32_t linePos = getLinePosition(sensorCalVal,lineColor);
 if(linePos > 1000 && linePos < 2000) {
  setMotorSpeed(LEFT_MOTOR,normalSpeed);
  setMotorSpeed(RIGHT_MOTOR,fastSpeed);
 } else if(linePos > 2000 && linePos < 3500) {
  setMotorSpeed(LEFT_MOTOR,normalSpeed);
  setMotorSpeed(RIGHT_MOTOR,midspd);}
 else if(linePos > 4500 && linePos < 5500)
 {
  setMotorSpeed(RIGHT_MOTOR,normalSpeed);
  setMotorSpeed(LEFT_MOTOR,midspd);
 }
 else if(linePos > 5500)
 {
  setMotorSpeed(RIGHT_MOTOR,normalSpeed);
  setMotorSpeed(LEFT_MOTOR,fastSpeed);

 }
 else if(linePos >= 6800 || linePos< 200)
 {
  disableMotor(BOTH_MOTORS);
  break;
 }
 else
 {
  setMotorSpeed(LEFT_MOTOR,normalSpeed);
  setMotorSpeed(RIGHT_MOTOR,normalSpeed);
 }
 }

 digitalWrite(GREEN,LOW);
}

void DRV_STR(int distance,int speed) // takes arbitrary distance in CM
{
  // this section will determine the number of encoder pulses for the
  int internal_ratio = 360; // number of encoder pulses per wheel rotation
  const float pi = 3.1415926572; // aproximation of pi
```

```
    int wheeldiam = 7; // aproximation of wheel diamiter
    float wheelCurcumfrence = 7*pi;
    float numRotations = distance/wheelCurcumfrence; // this number represents the wheel rotations that
    int finalPulses = numRotations*internal_ratio; // this represent the number of pulses

    int LMS = speed-2; // left motor speed
    int RMS = speed+1; // Right motor speed


    // motors setup
    resetLeftEncoderCnt();
    resetRightEncoderCnt();
    enableMotor(BOTH_MOTORS);
    setMotorDirection(LEFT_MOTOR,MOTOR_DIR_FORWARD);
    setMotorDirection(RIGHT_MOTOR,MOTOR_DIR_FORWARD);
    //acivate motor with ramp up to aleveate jolt
    int encoderL = getEncoderLeftCnt();
    int encoderR = getEncoderRightCnt();
    setMotorSpeed(LEFT_MOTOR,1);
    setMotorSpeed(RIGHT_MOTOR,1);
    delay(15);

    while((((encoderL = getEncoderLeftCnt()) + (encoderR = getEncoderRightCnt()))/2)<finalPulses)// compaire the average of the encoder values on each wheel with the required number of pulses
    {


      if(encoderL>(encoderR+1))
      {
        setMotorSpeed(LEFT_MOTOR,LMS-2);
        setMotorSpeed(RIGHT_MOTOR,RMS+2);
      }
     else if(encoderL<(encoderR+1)) // comao
      {
        setMotorSpeed(LEFT_MOTOR,LMS+2);
        setMotorSpeed(RIGHT_MOTOR,RMS-2);
      }
      else
      {
        setMotorSpeed(LEFT_MOTOR,LMS);
        setMotorSpeed(RIGHT_MOTOR,RMS);
      }

    }
    disableMotor(BOTH_MOTORS);
```

```

}

void SATIC_TURN_DEG(int TFLG,int deg, int speed) // REMEMBER IT IS THE EXTERNAL ANGLE NOT INTERNAL ANGLE
{
   // TFLAG indicates which turn direction relative to the robot the user wishes to pivot 0=L 1=R (defalts = L)
   // DFLAG indicates which direction the robot is driving the user wishes to pivot 0=F 1=B (defalts = F)
   // DEG indicates the number of degrees that the robot turns


   // BELOW IS BASIC SETUP
   disableMotor(BOTH_MOTORS); // just makes sure that the robot is stoped
   delay(50); // stop alows robot to come to a halt
   resetLeftEncoderCnt();
   resetRightEncoderCnt();
   enableMotor(BOTH_MOTORS);

  int currentDeg = 0;

  switch (TFLG){  // SWITCH STATEMENT FOR SETTING THE MOTOR TO BEING TURNING
  case 1:
    setMotorDirection(LEFT_MOTOR,MOTOR_DIR_FORWARD);
    setMotorDirection(RIGHT_MOTOR,MOTOR_DIR_BACKWARD);
    setMotorSpeed(BOTH_MOTORS,speed);
    currentDeg = (getEncoderLeftCnt()/6)+(getEncoderRightCnt()/6)/2;
    break;
  case 0:
   setMotorDirection(LEFT_MOTOR,MOTOR_DIR_BACKWARD);
   setMotorDirection(RIGHT_MOTOR,MOTOR_DIR_FORWARD);
   setMotorSpeed(BOTH_MOTORS,speed);
   currentDeg = (getEncoderLeftCnt()/6)+(getEncoderRightCnt()/6)/2;
    break;
  default:
    setMotorSpeed(BOTH_MOTORS,speed);
    currentDeg = getEncoderRightCnt()/3;
    break;
  }

  //the encoder pulses 360 times for a 90 degree turn
  //Thus we will need to have 4*360 pulses or 1440
  while(currentDeg<deg) // LOOP POLLS MOTOR ENCODERS (NEED TO LEARN HOW TO TRIGGER INTERUPS)
  {
      currentDeg = (getEncoderLeftCnt()/6)+(getEncoderRightCnt()/6)/2;
```

```
     }
     // DISSABLES ENCODERS AND MOTORS FOR SETUP FOR NEXT ACTION
     disableMotor(BOTH_MOTORS);

     enableMotor(BOTH_MOTORS);
    }
    float GET_US_DISTANCE_CM()
    {
     float echoTime;                   //variable to store the time it takes for a ping to bounce off an object
     float calculatedDistance;         //variable to store the distance calculated from the echo time

     //send out an ultrasonic pulse that's 10ms long
     digitalWrite(trigPin, LOW); //ensures a clean pulse beforehand
     delayMicroseconds(2);
     digitalWrite(trigPin, HIGH);
     delayMicroseconds(10);
     digitalWrite(trigPin, LOW);

     echoTime = pulseIn(echoPin, HIGH);      //use the pulsein command to see how long it takes for the
                                      //pulse to bounce back to the sensor in microseconds

     calculatedDistance += echoTime / 148.0;  //calculate the distance of the object that reflected the pulse (half the bounce time multiplied by the speed of sound)


     return (calculatedDistance)*2.54;             //send back the distance that was calculated
    }


    void DRIVE_ALONG_WALL(int speed)
    {
        int F;
        int L;
        myservo.write(90);
        delay(300);
        F = GET_US_DISTANCE_CM();
        Serial.print("F: ");
        Serial.print(F);
        Serial.print("\n");
        myservo.write(0);
        delay(500);
        L = GET_US_DISTANCE_CM();
        Serial.print("L: ");
        Serial.print(L);
```

```
      Serial.print("\n");
      setMotorSpeed(BOTH_MOTORS,speed);
      while(F>=20)
      {
        enableMotor(BOTH_MOTORS);
        if(L<15)
        {
          setMotorSpeed(RIGHT_MOTOR,speed+3);
          setMotorSpeed(LEFT_MOTOR,speed-1);
        }
        if(L>20)
        {
          setMotorSpeed(RIGHT_MOTOR,speed-1);
          setMotorSpeed(LEFT_MOTOR,speed+3);
        }
        else
        {
          setMotorSpeed(BOTH_MOTORS,speed);
        }
      delay(1000);
      disableMotor(BOTH_MOTORS);
      delay(100);
      myservo.write(90);
      delay(200);
      F = GET_US_DISTANCE_CM();

      myservo.write(0);
      delay(500);
      L = GET_US_DISTANCE_CM();
      myservo.write(90);
      delay(100);
      }
}

void loop() {
  int F,L,R; // values for the front left and Right of the robot
  int detect_threshold =15;
  int dist;
  bool LINEFOLLOWED = false;
  int lastL;
  delay(1000);
  //floorCalibration();
  //delay(10000);

    // Part 2:
    // Follows the line for and stops at a T

   if(digitalRead(LP_RIGHT_BTN) == LOW && LINEFOLLOWED == false )
```

```
{
 line_follow(); // line follower auto matically stops at a T
 LINEFOLLOWED  = true;

delay(5000);

 // part 3:
   F = GET_US_DISTANCE_CM();
   Serial.print("F: ");
   Serial.print(F);
   Serial.print("\n");
   myservo.write(0);
   delay(500);

   L = GET_US_DISTANCE_CM();
   Serial.print("L: ");
   Serial.print(L);
   Serial.print("\n");
   myservo.write(180);
   delay(500);
   R = GET_US_DISTANCE_CM();
   Serial.print("R: ");
   Serial.print(R);
   Serial.print("\n");
   myservo.write(90);
 if(R>L)
 {
   SATIC_TURN_DEG(0,42,20);
   F = GET_US_DISTANCE_CM();
   DRV_STR(30,15);
 }
 else if(L>R)
 {
   SATIC_TURN_DEG(1,40,20);
   F = GET_US_DISTANCE_CM();
   DRV_STR(30,15);
 }

 DRIVE_ALONG_WALL(10);
 while(true)
   {
     myservo.write(90);
     delay(500);
     F = GET_US_DISTANCE_CM();

     myservo.write(0);
     delay(500);
     L = GET_US_DISTANCE_CM();
```

```
            if(L<=10&&F<=10)
            {
              SATIC_TURN_DEG(0,45,20);
            }
            else if(L>=10&&F<=10)
            {
              SATIC_TURN_DEG(1,45,20);
            }
            else
            {
              SATIC_TURN_DEG(1,90,20);
              DRV_STR(20,15);
              DRIVE_ALONG_WALL(10);
            }
          }
        }
    }

```