Alex Crotts
acrotts2@uncc.edu
Luis Umana
lumana@uncc.edu

ECGR 4161/5196 Lab #6 Report

Due 03/21/21

Lab Objective:

The objective of the lab was to use ultrasonic sensor data to sweep left and right along a flat wall and determine when the TI-RSLK robot was facing perpendicular to it. The robot should then drive directly towards the wall until it is 30cm away, rotate in place 90 degrees, and drive 100cm parallel to the wall. The goal was to be as close to perpendicular as possible and as close to 30cm from the wall as possible before the turn. If those actions were performed correctly, the robot would remain 30cm from the wall at the end of the 100cm drive.

Commentary and Conclusion:

We encountered some difficulties storing ultrasonic distance values as neither lab partner was greatly experienced in creating arrays. There were occasional issues where the code would be stuck in an infinite loop depending on the angle the bot was placed. These issues typically required restarting the robot at which point it would perform the correct functions. The robot also tended to choose a slight angle to the left as the closest distance to the wall, leading to deviation from wall as it traveled 100cm. This issue did not always occur and could be solved by recording more distance values after each turn to get a better average distance from the wall.

Lab Code:

```
/*
Use an ultrasonic sensor to find the distance from a wall,
drive the RSLK robot straight toward the wall,
turn and drive parallel to the wall

Alex Crotts and Luis Umana - 3/17/2022
*/

#include <SimpleRSLK.h>

const int trigPin = 32;//This is Port Pin 3.5 on the MSP432 Launchpad
const int echoPin = 33; //This is Port Pin 5.1 on the MSP432 Launchpad

int MotorSpeed = 10;
float WheelDiameter = 6.985;     // In centimeters
float PulsePerRev = 360;           // Number of encoder pulses the
microcontroller reads per 1 wheel rotation
float WheelBase = 13.335;        // In centimeters

// Number of encoder pulses per 1 degree of rotation
double PulsePerDegree = WheelBase/WheelDiameter;

void setup() {
  // Initialization
  pinMode(trigPin, OUTPUT);    // Set trigPin as an output
  pinMode(echoPin, INPUT);     // Set echoPin as an input
  setupRSLK();
  resetLeftEncoderCnt();      // Reset encoder counts
  resetRightEncoderCnt();
  Serial.begin(9600);
  Serial.println("Beginning Sweep");
  delay(1000);      // Delay to allow the serial monitor to settle
}

void Drive_Straight(int y) {
  // Integer y allows for this function to be called for any distance
  // Function for driving straight for X centimeters
  resetLeftEncoderCnt();
  resetRightEncoderCnt();
  enableMotor(BOTH_MOTORS);
  // Set both motors to drive forward
  setMotorDirection(BOTH_MOTORS, MOTOR_DIR_FORWARD);
  // Set both motors to the same speed
  setMotorSpeed(BOTH_MOTORS, MotorSpeed);
  int L_Pulse_Count = 0;      // Zero the left encoder pulse count
  int R_Pulse_Count = 0;      // Zero the right encoder pulse count

  while((L_Pulse_Count < y) || (R_Pulse_Count < y)) {
    // Run until the pulses reach the value stated in the void loop
    L_Pulse_Count = getEncoderLeftCnt();      // Read the left encoder value
    R_Pulse_Count = getEncoderRightCnt();     // Read the right encoder value

    if((L_Pulse_Count + 1 < R_Pulse_Count)){
      // If left is driving slower than right, speed up left and slow down right
      // Speed up the left motor in increments of 1
      setMotorSpeed(LEFT_MOTOR, ++MotorSpeed);
      // Slow down the right motor in increments of 1
      setMotorSpeed(RIGHT_MOTOR, --MotorSpeed);
    }

    if((R_Pulse_Count + 1 < L_Pulse_Count)){
      // If right is slower than left, speed up right motor and slow down left
      // Speed up the right motor in incremements of 1
      setMotorSpeed(RIGHT_MOTOR, ++MotorSpeed);
      // Slow down the left motor in increments of 1
      setMotorSpeed(LEFT_MOTOR, --MotorSpeed);
    }

    if(L_Pulse_Count >= y){
      // If the pulses reach the specified value, turn off motors
      disableMotor(LEFT_MOTOR);     // Turn off the left motor
      disableMotor(RIGHT_MOTOR);    // Turn off the right motor
      }

      // Print encoder counts to the serial monitor for debugging
      Serial.print("Driving Straight Now");
      Serial.print("\t");
      Serial.print("Left Encoder: ");
      Serial.print(L_Pulse_Count);
      Serial.print("\t");
      Serial.print("Right Encoder: ");
      Serial.println(R_Pulse_Count);
      delay(100);
  }
}

void Rotate(int z, int L_Motor_Dir, int R_Motor_Dir) {
  // Integers allow the function to be called for CW or CCW and any degree
  // Function for rotating the RSLK robot in place
  resetLeftEncoderCnt();
  resetRightEncoderCnt();
  enableMotor(BOTH_MOTORS);
  // Set the left motor to drive in the specified direction
  setMotorDirection(LEFT_MOTOR, L_Motor_Dir);
  // Set the right motor to drive in the specified direction
  setMotorDirection(RIGHT_MOTOR, R_Motor_Dir);
  setMotorSpeed(BOTH_MOTORS, MotorSpeed);    // Set the motors to the same speed
  int L_CCW_Pulse_Count = 0;      // Zero the encoder count
  int R_CCW_Pulse_Count = 0;      // Zero the encoder count

    while(R_CCW_Pulse_Count < z) {
    // Run this loop until the pulses reach the specified value
    L_CCW_Pulse_Count = getEncoderLeftCnt();     // Read the left encoder value
    R_CCW_Pulse_Count = getEncoderRightCnt();    // Read the right encoder value

    if(R_CCW_Pulse_Count >= z) {
      // If the pulses reach the specified value, turn off the motors
      disableMotor(LEFT_MOTOR);       // Turn off the left motor
      disableMotor(RIGHT_MOTOR);      // Turn off the right motor
      delay(1000);
    }

    //Print encoder counts to the serial monitor for debugging
    Serial.print("Turning CCW Now");
    Serial.print("\t");
    Serial.print("Left Encoder CCW Turn: ");
    Serial.print(L_CCW_Pulse_Count);
    Serial.print("\t");
    Serial.print("Right Encoder CCW Turn: ");
    Serial.println(R_CCW_Pulse_Count);
    delay(100);
  }
}

long Read_Distance() {
  // This function reads the distance from the ultrasonic sensor
  byte Readings[7];   // Declare an array of readings
  int x = 0;          // Array indexed at zero
  long pulseLength;   // Length of the ultrasonic pulse
  long centimeters;   // Calculated distance
  long total = 0;     // Initially zero the total for averaging the array
  long average;       // Calculated average of the array

  // Sending the pulse to the ultrasonic sensor
  digitalWrite(trigPin, LOW);
  delayMicroseconds(10);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  delayMicroseconds(10);

  // Calculating the distance from the pulse length
  pulseLength = pulseIn(echoPin, HIGH);
  centimeters = pulseLength / 58;

  // Set up the loop to store values in an array
  for(Readings[x]; x < 7; x++) {
    // Read from the sensor:
    Readings[x] = centimeters;
    // Add the reading to the total:
    total = total + Readings[x];

    // If we're at the end of the array...
    if (x >= 7) {
      // ...wrap around to the beginning:
      x = 0;
      }
  }

  // Calculate the average of the array:
  average = total / 7;
  // send it to the computer as ASCII digits
  Serial.print("Average Distance: ");
  Serial.println(average);
  delay(100);        // delay in between reads for stability
  return(average);
    }

void loop() {
  long old_average = Read_Distance();   // Store an initial distance
  // Rotate CCW 5 degrees
  Rotate(5*PulsePerDegree, MOTOR_DIR_BACKWARD, MOTOR_DIR_FORWARD);
  long new_average = Read_Distance();   // Store a new distance value

  if(new_average < old_average) {
    // Compare the new and old distances
    // If the distance after turning is less than before, turn CCW 5 degrees
    Rotate(5*PulsePerDegree, MOTOR_DIR_BACKWARD, MOTOR_DIR_FORWARD);
    delay(500);
  }

  else {
    // If the distance after turning is greater than before, disable motors
    disableMotor(LEFT_MOTOR);       // Turn off the left motor
    disableMotor(RIGHT_MOTOR);      // Turn off the right motor
    delay(1000);
    old_average = Read_Distance();    // Store the distance value
    // Rotate CW 2 degrees
    Rotate(2*PulsePerDegree, MOTOR_DIR_FORWARD, MOTOR_DIR_BACKWARD);
    new_average = Read_Distance();    // Store the new distance value
    delay(1000);

    if(new_average < old_average + 1) {
      // If the distance after the turn is less than before, turn CW 2 degrees
      Rotate(2*PulsePerDegree, MOTOR_DIR_FORWARD, MOTOR_DIR_BACKWARD);
      delay(1000);
    }

    if(new_average < old_average) {
      // If the distance after turning is less than before, disable motors
      disableMotor(LEFT_MOTOR);       // Turn off the left motor
      disableMotor(RIGHT_MOTOR);      // Turn off the right motor
      delay(1000);
      // Drive straight until 30cm from the wall
      Drive_Straight((Read_Distance()-30)/((WheelDiameter * PI)/(PulsePerRev)));
      delay(500);
      // Rotate 90 degrees CW
      Rotate(90*PulsePerDegree, MOTOR_DIR_FORWARD, MOTOR_DIR_BACKWARD);
      delay(500);
      // Drive straight 100cm
      Drive_Straight((100)/((WheelDiameter * PI)/(PulsePerRev)));
      delay(10000);     // Pause for 10 seconds for measurement
    }
  }
}
```