**MECH 452 - Mechatronics Engineering**

**Department of Mechanical and Materials Engineering**

**Faculty of Engineering and Applied Science, Queen’s University, Kingston**

**Group #24**

**Laboratory #42 - Introduction to the Universe and Everything**

**Student Names:** Joshua Chan, Connor Russo, Matthew Birchard

**Instructor Name:** Dr. Li

**Date Report Submitted:** 2024/03/04

**Summary:**

A prewired USB-Boarduino and Sharp sensor was used to control a Lynxbot. The Sharp signal was first checked to make sure that it was working. Once confirming the correct workings of the Sharp sensor, skeleton code for the lynxbot to travel around a square. Taking readings from the Sharp sensor, if the reading read “too far” from the wall, the inner wheels would stop rotating while the outer wheels continued. This differential in wheel rotation would turn the lynxbot towards the square, back to within an acceptable tolerance. If the lynxbot was “too close” to the box, the outer wheels would stop spinning while the inner wheels would continue to spin. This turns the lynxbot away from the wall.

After making sure that the lynxbot made the right path around the square, a proportioanl controller was added to the code to prevent over correction and allow for a smoother run after turns. The proportioanl controller prevented the lynxbot from traveling in a zig-zag. Once the proportional constant was tuned, the speed of the lynxbot was increased until the bot completed the circuit in the required time.

**Program:**

A diagram of a process flow

Description automatically generated

Figure 1: Flowchart of the M452Lab4PROP-TEST code.

The flowchart in the figure above demonstrates the logic of the code in *M452Lab4PROP-TEST*. The code begins by checking to see if the bumper has collided with the wall before moving forward. Then the distance sensor will check if the robot is too far from the wall and correct the robot by turning the robot left before going to the beginning of the loop. The distance sensor will check if the robot is too close to the wall and correct the robot by turning the robot right before going to the beginning of the loop. If the robot collides with the wall at any point, detected by the bumpers, the robot is reversed slightly and then the program ends.

**Results***:*

Table 1 shows the impact of tuning delta with a set hysteresis value. For lower values of delta, the robot shows less oscillation and a smoother path of travel. This is due to the system having more time to respond to error and having less overshoot with each correction.

**Table 1**. Tuning of ***delta*** with ***HYS*** of 50.

|  |  |  |
| --- | --- | --- |
| ***delta*** | Lap time (sec) | comment |
| 15 | 26.04 | - slow with little oscillation |
| 25 | 15.79 | - looks good |
| 40 | 14.27 | - fast with excessive oscillation |

Table 2 shows the impact of tuning the hysteresis value. The table shows that a higher hysteresis value leads to less oscillation of the robot as it travels. This is because the range for error is increased, allowing the robot to deviate further from its path without corrective action.

**Table 2**. Tuning of ***HYS*** with ***delta*** of 25.

|  |  |  |
| --- | --- | --- |
| ***HYS*** | Lap time (sec) | comment |
| 25 | 18.28 | * A lot of oscillation |
| 50 | 15.62 | * Some oscillation |
| 100 | 14.75 | * Little oscillation |

Table 3 shows the impact of changing the motor speed value while maintaining a constant proportional gain of 0.5. A similar result to table 1 is observed where a higher speed yields more oscillation. This is due to the robot overcorrecting and the system having less time to respond to the correction.

**Table 3.** Tuning of ***delta*** with ***Kp*** of 0.5.

|  |  |  |
| --- | --- | --- |
| ***delta*** | Lap time (sec) | comment |
| 15 | 27.48 | Slow but no oscillation around corners |
| 25 | 13.78 | good pace with little oscillation around corners |
| 40 | 9.75 | Fast but lots of oscillation |

Table 4 shows the impact of changing the proportional gain value with a constant motor speed. The table shows that a higher KP value has more oscillation, particularly around the corners. This is due to the higher KP increasing the settling time.

**Table 4**. Tuning of ***KP*** with ***delta*** of 25.

|  |  |  |
| --- | --- | --- |
| ***KP*** | Lap time (sec) | comment |
| 0.25 | 14.75 | Slow with no oscillation |
| 0.50 | 14.78 | Faster but some oscillation |
| 0.75 | 15.95 | Most oscillation around corners |

***Placement of the Sensor***

The sensor was placed in a way that ensured it was within its useful operating range without obstructions from the robot chassis. The sensor was placed on the left side of the robot closest to the wall of the track. The sensor was placed at 20 cm which is in the middle of the steepest part of the curve on the data sheet. This yields the highest change in output for a change in input distance from the wall. The sensor was mounted on a stack of Lego blocks to elevate it above the wheels of the robot and ensure it was reading the distance to the wall.

**Appendix A - Program Listing***: (8 pitch font, 2 column, single spaced, changes highlighted EXAMPLE USING TUTORIAL 1 PROGRAM. REPLACE WITH LAB 1 WITH YOUR CHANGES HIGHLIGHTED)*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

M452Lab4PROP Tested code and graded (v2.0)

Original by H. Fernando, 24/09/2013

Revised by Matt, Josh, Connor 2024/03/04

This is an ON/OFF wall following algorithm for a robot equipped

with a single Sharp on the left front corner of the robot.

A proportional controller was added to smooth out the turns.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Pin Assignments

int RED = 4;

int GRN = 5;

int YLW = 6;

int BUTTON\_A = 7;

int BUTTON\_B = 8;

int BUTTON\_C = 9;

int MOTOR\_L = 10; // left motor signal

int MOTOR\_R = 11; // right motor signal

int BUMPER = 12;

int SHARP1 = A1; // sharp input pin

int sensor1; // sharp sensor reading

// recommended initial values

int delta = 32; // 25 as default for speed for speed test

int HYS = 100; // 50 as default for hysterisis

// adjust stop speed and target distance as appropriate

int WALL = 1400; // 700 (40cm) for corridor, 1200 (20cm) for wall

int STOP\_SPEED = 147; // 147 as default, change to match that in Lab #4

//Set proportional gain value

float KP = 0.15;

// Set-up Routine

void setup() {

// initialize the digital led pins as outputs.

pinMode(RED, OUTPUT);

pinMode(YLW, OUTPUT);

pinMode(GRN, OUTPUT);

//initialize buttons and bumper pins as inputs

pinMode(BUMPER, INPUT);

pinMode(BUTTON\_A, INPUT);

pinMode(BUTTON\_B, INPUT);

pinMode(BUTTON\_C, INPUT);

//initialize motor control pins as outputs

pinMode(MOTOR\_L, OUTPUT);

pinMode(MOTOR\_R, OUTPUT);

//setup serial debug

Serial.begin(9600);

runMotors(0, 0); // check motors are stopped

Serial.println("Press Button B to start.");

do {

toggleLED(GRN); //motors stopped, Green LED flashing

} while (digitalRead(BUTTON\_B) == HIGH);

sensor1 = map(analogRead(SHARP1), 0, 1023, 0, 5000); // initialize sensor

Serial.println("Program Running. Press bumper to stop");

Serial.println("Select speed");

}

// Main Routine

void loop() {

// Select motor speed

do {

toggleLED(YLW);

if (digitalRead(BUTTON\_A) == LOW) {

delta = 35;

Serial.println("Set delta = 15");

break;

}

else if (digitalRead(BUTTON\_B) == LOW) {

delta = 37;

Serial.println("Set delta = 25");

break;

}

else if (digitalRead(BUTTON\_C) == LOW) {

delta = 40;

Serial.println("Set delta = 45");

break;

}

}

while (true);

//// start of bumper check loop

do {

sensor1 = map(analogRead(SHARP1), 0, 1023, 0, 5000); // initialize sensor

int error = WALL - sensor1;

int dummy = min(abs(error) \* KP, 100);

int deltaV = delta \* (100 - dummy) / 100;

if (error > 0 ) { //too far from the wall

turnOnLED(YLW);

runMotors(deltaV, delta);

}

else if (error < 0){ //too close to the wall

turnOnLED(RED);

runMotors(delta, deltaV);

}

else { //desired distance

turnOnLED(GRN);

runMotors(delta, delta);

}

Serial.println(error);

Serial.println(dummy);

Serial.println(deltaV);

Serial.println(sensor1);

Serial.println(" ");

}

while (digitalRead(BUMPER) == LOW); // end of bumper loop

runMotors(0, 0);

delay(500);

runMotors(-delta, -delta);

delay(250);

runMotors(0, 0);

while (1) { // loop forever

flashAllLEDs();

}

}

//\*\*\*\*\*\*\*\*\*\*FUNCTIONS (subroutines)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// left and right motor commands

void runMotors(int delta\_L, int delta\_R) {

int pulse\_L = (STOP\_SPEED + delta\_L) \* 10; //determines length of pulse in microsec

int pulse\_R = (STOP\_SPEED + delta\_R) \* 10;

for (int i = 0; i < 3; i++) {

pulseOut(MOTOR\_L, pulse\_L); //send pulse to left motors

pulseOut(MOTOR\_R, pulse\_R); //send pulse to right motors

}

}

// single motor pulsewidth command

void pulseOut(int motor, int pulsewidth) {

digitalWrite(motor, HIGH);

delayMicroseconds(pulsewidth); //send pulse of desired pulsewidth

digitalWrite(motor, LOW);

}

//Turn on a single LED, and all other off

void turnOnLED(int colour) {

digitalWrite(GRN, LOW);

digitalWrite(YLW, LOW);

digitalWrite(RED, LOW);

digitalWrite(colour, HIGH);

}

//Toggle an LED on/off

void toggleLED(int colour) {

digitalWrite(colour, HIGH);

delay(250);

digitalWrite(colour, LOW);

delay(250);

}

//flash all LEDs

void flashAllLEDs() {

digitalWrite(GRN, LOW);

digitalWrite(YLW, LOW);

digitalWrite(RED, LOW);

delay(250);

digitalWrite(GRN, HIGH);

digitalWrite(YLW, HIGH);

digitalWrite(RED, HIGH);

delay(250);

}