Robotics Group Project Individual Report

Zehra Punjwani

1431800

King’s College London

Table of Contents

[Introduction 4](#_Toc476875344)

[Robotics Group Project Individual Report 5](#_Toc476875345)

[Team 1 5](#_Toc476875346)

[Team 2 5](#_Toc476875347)

[Conclusion and Discussion 6](#_Toc476875348)

[Limitations 6](#_Toc476875349)

[Future Work 6](#_Toc476875350)

[References 7](#_Toc476875351)

[Appendix 8](#_Toc476875352)

[Team 1 8](#_Toc476875353)

[Program Code – Java 8](#_Toc476875354)

[Team 2 14](#_Toc476875355)

Introduction

Robotics Group Project Individual Report

## Team 1

## Team 2

Conclusion and Discussion

## Limitations

## Future Work

References

Appendix

## Team 1

### Program Code – Java

package leJOSEV3;

import lejos.hardware.sensor.EV3UltrasonicSensor;

import lejos.hardware.port.Port;

import lejos.hardware.port.SensorPort;

import lejos.hardware.port.MotorPort;

import lejos.robotics.SampleProvider;

import lejos.utility.Delay;

import lejos.hardware.motor.EV3LargeRegulatedMotor;

import lejos.hardware.Button;

import lejos.hardware.lcd.LCD;

import lejos.hardware.sensor.EV3ColorSensor;

import lejos.robotics.Color;

public class Team1Final {

static Port ultraSonicSensorPort;

static Port colorSensorPort;

static EV3UltrasonicSensor ultraSonicSensor;

static EV3ColorSensor colorSensor;

static EV3LargeRegulatedMotor motorLeft;

static EV3LargeRegulatedMotor motorRight;

static EV3LargeRegulatedMotor motorSpin;

static SampleProvider sampleProviderSound;

static SampleProvider sampleProviderLight;

static int sampleSoundSize;

static int sampleSizeLight;

static float[] sampleSound;

static float[] samplesLight;

static float valueSound;

static float valueLight;

static float errorSound;

static float errorLight;

static float derivativeSound;

static float integralSound;

static float lasterrorSound;

static float derivativeLight;

static float integralLight;

static float lasterrorLight;

static float obstacleDistance;

static float correctionSound;

static float kpSound;

static float kiSound;

static float kdSound;

static float tpSound;

static float black;

static float white;

static float midPoint;

static float kpLight;

static float kiLight;

static float kdLight;

static float tpLight;

static int stateTracker;

static int tmp = 0;

public static void initialise() {

ultraSonicSensorPort = SensorPort.S4;

ultraSonicSensor = new EV3UltrasonicSensor(ultraSonicSensorPort);

sampleProviderSound = ultraSonicSensor.getMode("Distance");

sampleSoundSize = sampleProviderSound.sampleSize();

sampleSound = new float[sampleSoundSize];

colorSensorPort = SensorPort.S1;

colorSensor = new EV3ColorSensor(colorSensorPort);

sampleProviderLight = colorSensor.getRedMode();

colorSensor.setFloodlight(Color.RED);

sampleSizeLight = sampleProviderLight.sampleSize();

samplesLight = new float[sampleSizeLight];

motorRight = new EV3LargeRegulatedMotor(MotorPort.D);

motorLeft = new EV3LargeRegulatedMotor(MotorPort.A);

motorSpin = new EV3LargeRegulatedMotor(MotorPort.C);

lasterrorSound = 0;

integralSound = 0;

derivativeSound = 0;

correctionSound = 0;

lasterrorLight = 0;

integralLight = 0;

derivativeLight = 0;

black = 0;

white = 0;

//pid controls for ultrasound

obstacleDistance = (float) 0.02;

tpSound = 100;

kpSound = 200;

kiSound = 0;

kdSound = 10;

//pid controls for light

tpLight = 120;

kpLight = 1000;

kiLight = (float) 0;

kdLight = 120;

motorLeft.setSpeed(0);

motorRight.setSpeed(0);

motorLeft.forward();

motorRight.forward();

stateTracker = 0;

}

public static float readSound(float[] sampleArray){

sampleProviderSound.fetchSample(sampleArray, 0);

return sampleArray[0];

}

public static float readLight(float[] sampleArray){

sampleProviderLight.fetchSample(sampleArray, 0);

return sampleArray[0];

}

static void lineFollower() {

valueLight = readLight(samplesLight);

errorLight = (midPoint - valueLight);

integralLight = integralLight + errorLight; // Integral...

derivativeLight = errorLight - lasterrorLight; // Derivative...

float correction = kpLight\*errorLight + kiLight\*integralLight + kdLight \* derivativeLight;

motorLeft.setSpeed(tpLight + correction);

motorRight.setSpeed(tpLight - correction);

motorLeft.forward();

motorRight.forward();

lasterrorLight = errorLight;

obstacleDetect();

}

static void obstacleDetect(){

float object = readSound(sampleSound);

if(object < 0.18){

motorLeft.setSpeed(0);

motorLeft.setSpeed(0);

motorLeft.forward();

motorRight.forward();

stateTracker++;

}

}

static void obstacleAvoid() {

valueSound = readSound(sampleSound);

if(valueSound < obstacleDistance + 0.2 || valueSound > obstacleDistance - 0.2 ) {

errorSound = obstacleDistance - valueSound; //if neg need to turn left

correctionSound = errorSound \* kpSound;

motorLeft.setSpeed(tpSound + correctionSound);

motorRight.setSpeed(tpSound - correctionSound);

motorLeft.forward();

motorRight.forward();

Delay.msDelay(500);

detectLine();

}

}

static void calibrate(float[] sampleArray) {

LCD.drawString("(Calibrate White)", 1, 1);

Button.waitForAnyPress();

white = readLight(sampleArray);

LCD.drawString("(Calibrate Black)", 1, 1);

Button.waitForAnyPress();

black = readLight(sampleArray);

midPoint = (white - black)/2 +black;

LCD.drawString("(Press To START)", 1, 1);

Button.waitForAnyPress();

}

static void turnRobot90(){

motorLeft.stop();

motorRight.stop();

boolean b = true;

float current;

float previous = 0;

float delta;

while(b) {

current = readSound(sampleSound);

if(Float.isInfinite(current)){

motorLeft.setSpeed(40);

motorRight.setSpeed(40);

motorLeft.forward();

motorRight.backward();

Delay.msDelay(300);

}

else{

delta = current - previous;

if(delta > 0 || Float.isInfinite(current)){

motorLeft.setSpeed(40);

motorRight.setSpeed(40);

motorLeft.forward();

motorRight.backward();

Delay.msDelay(300);

LCD.drawString(Float.toString(delta), 1, 1);

previous = current;

}

else if(delta < -0.025 && current < 0.2){

motorLeft.setSpeed(40);

motorRight.setSpeed(40);

motorLeft.forward();

motorRight.backward();

Delay.msDelay(1500);

motorLeft.setSpeed(0);

motorRight.setSpeed(0);

motorLeft.forward();

motorRight.forward();

LCD.drawString(Float.toString(delta), 1, 1);

b = false;

}

}

}

stateTracker++;

}

static void turnSensor90(boolean direction) { //True for anti-clockwise...

if(direction == true) {

//Anti-clockwise

motorSpin.rotate(-110);

} else {

//Clockwise

motorSpin.rotate(110);

}

}

static void waitForCurtain() {

boolean curtainUp = false;

while(curtainUp == false){

float curtainDistance = readSound(sampleSound);

if(curtainDistance > 0.12){

curtainUp = true;

stateTracker++;

}

}

}

static void detectLine() {

float detectionValue = readLight(samplesLight);

System.out.println(Float.toString(detectionValue));

if(detectionValue <= white - (0.8\*white)) {

motorLeft.setSpeed(45);

motorRight.setSpeed(45);

motorLeft.forward();

motorRight.backward();

Delay.msDelay(9000);

motorLeft.setSpeed(70);

motorRight.setSpeed(70);

Delay.msDelay(2000);

turnSensor90(false);

motorLeft.setSpeed(0);

motorRight.setSpeed(0);

motorLeft.forward();

motorRight.forward();

stateTracker++;

}

}

public static void main(String[] args) {

initialise();

calibrate(samplesLight);

while(true){

if(stateTracker == 0){

waitForCurtain();

}

else if(stateTracker == 1){

lineFollower();

}

else if(stateTracker == 2){

turnSensor90(true);

turnRobot90();

}

else if(stateTracker == 3){

obstacleAvoid();

}

else if(stateTracker == 4){

lineFollower();

}

}

}

}

## Team 2