University of Essex
School of Computer Science and Electronic Engineering
Department of Computer Systems Engineering

CE215 Robotics

Robot Navigation

Prepared by
Waleed Bin Asad
1805967
wa18382@essex.ac.uk
Computer Systems Engineering

Contents

1 INTRODUCTION

This report is going to discuss about robot navigation, where we have to test our differential drive robot in a given *(200cm by 200cm)* arena. There are two locations marked on it one is the starting point and the other ending point. We have to program our robot in a way where it controls its behaviour passing from the starting point and on the way there are obstacles *(cylinder with a 10cm diameter)* going to be placed which the robot detects using its ultrasonic sensor and once it reaches its end point *(detect colour red using the light sensor)* it has to return back to its starting point. Apart from this we had to implement more than one trial while it is moving in the arena, its trajectory must be displayed on the LCD panel and apart from this the trajectory coordinate must be recorded in the CSV file. More descriptions about obstacle avoidance behaviours and the equations that were used will be explained late in the report.

2 BEHAVIOURS INVOLVED

As we know that to make the robot navigate in a given area, it has to follow different behaviours which will make it possible for it to go according to our expectations. The LeJos Behaviour class which offers support from implementing behaviour-based systems. There are three summed up categories and are explained in the below units.

2.1 GOAL BEHAVIOUR

The robot was provided with the metric map, its current position and the goal position. Its main waypoint aspect was to plan a path from the current position (0,0) to the goal position which is set at (150,150) coordinates and return back to the current position. Two things required for this to make possible:

Light Sensor: This was placed at the bottom of the robot and it only responds to light coming from the front. The reason it is essential to use is so that when it detects colour red on the circles during its waypoint it will send the information that it has detected the colour and needs to return back to its starting point.

Localisation: There are various algorithms applied to plan a path from the current location to the destination. PID was used to control the robot to track the path. The navigator class is used an Odometry pose provider to keep tracking of the local information of the robot.

The path tracking controller is aimed to track the exact number of lines with the aid of localisation. It gives a specific angle for the robot to rotate and to execute the actual performance.

These equations denote the position of the goal. The ‘d’ is the distance between to goal and the robot while the other theta is the angle between the robot and the goal

$$d = \sqrt{(x_G - x)^2 + (y_G - y)^2}$$

$$\theta = \tan^{-1}\left(\frac{y_G - y}{x_G - x}\right)$$

d	Angle(degrees)
$\sqrt{(150\text{-}0)^2 + (150\text{-}0)^2}$ = 212.132	$\text{Tan}^{-1}(150\text{-}0/150\text{-}0)$ = 45

The robot rotates 45 degrees towards its goal position.

So in the code below there is a constructor class for the light sensor which is used to detect the given colour. The navigator is used to traverse a path, a sequence of waypoints. It uses the pose provide which calls its navigator listener when a waypoint is reached.

```
/* Constructors that are sensor port and the navigator */

 public detectcolor(SensorPort sen, Navigator navi) {
        this.cS = new ColorSensor(sen);
        this.Nav = navi;
    }
```

In the action there is a method used by the navigator followpath. What it does is that from the starting point (0,0) it maps its route to the destination at (150,150). The supress condition is an exception that is thrown from the method. Only one exception will be thrown by the compiler at a time.

```
/* Methods that will perform some actions when it detects colour */
@Override
public void action() {
    suppressed = false;
    if(!suppressed){
        Nav.addWaypoint(0, 0);
        Nav.followPath();
    }
}
```

Finally the take control method is matched with the colour red using the light sensor mode and now when it reaches the destination the robot triggers itself and returns back to the starting point. There is exploring behaviour involved which will be discussed in the unit 2.2

```
@Override
public void suppress() {
    suppressed = true;
};

@Override
public boolean takeControl() {
    return (cS.getColorID() == ColorSensor.Color.RED);
```

2.2 EXPLORING BEHAVIOUR

This is a situation when the red circles are quite small and the robot needs to detect the colour Red when it is at the destination coordinates. So using one of the behaver based systems, there is a class for search colour which does the following:

I have set a boundary of 40cm square around the destination which allows the robot to search for the colour red. Without this we would never be able to find the exact location of the destination point. Because every time the robot moves it loses its track of the previous position and the uncertainty around it gets higher and higher.

```
/* when the robot reaches the end point, it will search for colour red by moving
within the given range*/
private static int FIND_L = 40; // 40cm sweep
```

In this action method the robot will perform to go forward, then backwards and finally rotate at a given angle. This is done when the robot reaches the destination and is trying to search for the red circle.

```
@Override
public void action() {
    suppressed = false;
    difp.setTravelSpeed(30);
    difp.setRotateSpeed(40);
    while(!suppressed){

        difp.rotate(55);
        if(suppressed){
            break;
        }
        difp.travel(FIND_L /4 );
        if(suppressed){
            break;
        }
        difp.travel(FIND_L / 4);
        if(suppressed){
            break;
        }
        difp.travel(FIND_L / 4);
        if(suppressed){
            break;
        }
        difp.travel(-FIND_L);
        if(suppressed){
            break;
        }
    }
}
```

Finally will the takeControl method which gives information to the navigator that the path is completed. Then it rotates back to its back to the starting point.

```
@Override
public boolean takeControl() {

    return Nav.pathCompleted();
}
```

2.3 OBSTACLE AVOIDANCE BEHAVIOUR

So during the tracjetory there are obstacles placed inside the arena which the robot needs to detect and take some action. There are multiple tasks require in our experiment. Moving to the goal while avoiding obstacles. We are trying to coordinate two different behaviours. Around is a potential field concept which states that if the robot is a bit far away from the obstacle, the robot will follow the original direction and speed to move. There is a force applied which is inversely proportional to the distance with direction away from the obstacle.

There is an ultrasonic sensor *(produces an ultrasonic pulse wave and listen to the echo)* placed at the front of the robot which has an accuracy of +/-3 cm. This was used to report the distance and to detect obstacle upfront.

Here are few equations that were used for obstacle avoidance.

$$d = \sqrt{(x_O - x)^2 + (y_O - y)^2} \qquad \theta = \tan^{-1}\left(\frac{y_O - y}{x_O - x}\right)$$

This was for manual clarification purpose to check if the robot turns at the given angle when it is heading near the obstacle.

The method that was implemented for doing multiple tasks were that there is one coordination methods meaning two different potential fields that are added in a separate behaviour based system. One where it avoids obstacle and the other one where it follows the path after it.

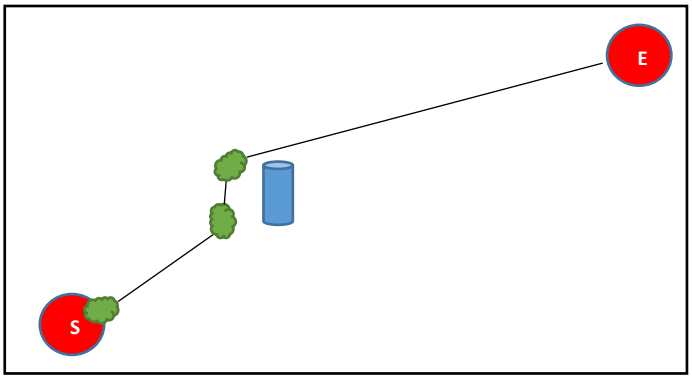
Below in my code I've set a distance of 17 cm. So the ultrasonic sensor is going to detect and obstacle within the given distance and perform some action.

```
/* Constants value for avoiding distance */
private static int OBSTACLE_DISTANCE = 17; // When sensor detects obstacle within
17cm
```

Therefore when the obstacle is detected the robot rotates at a given angle and travel a short distance further away from the obstacle. It then rotates automatically to moved forward and head to the destination point

```
public void action() {
    suppressed = false;
    if(!suppressed){
        Nav.stop();
        // what it does
        difp.setTravelSpeed(30);
        difp.setRotateSpeed(30);
        difp.rotate(35);
        difp.travel(35);

    }
}
```



The take control method is compulsory because it gives a condition to the robot that when the obstacle is less than the specified distance it should do some action.

```
@Override
public boolean takeControl() {
    return (sonarSensor.getDistance() <= OBSTACLE_DISTANCE);

}
```

3 BEHAVIOURS COORDINATION

This is a behaviour modules that encapsulate rules and specify how the robot should behave in the environment. The vital feature is to add new behaviour that can manage and simplify the control problem. So in our case from [*section 2.3*] is one of the behaviour coordination's stage. The robot has to detect an obstacle and at the same time it needs to take a different action which is heading to the destination point.

One way is that the coordination methods meaning two different potential fields that are added in a separate behaviour based system. One where it avoids obstacle and the other one where it follows the head forwards and completes its trajectory. There is a vector summation method which states that the robot will move following the combined potential field with coordinated behaviour. There are potential fields across the robot:

1. One when the robot is heading towards the obstacle in the given distance time it generate a repulsive force that makes the robot move away from the obstacle.
2. Around is an attractive force between the goal position and the robot because the distance between them is much greater and now the robot is attracting towards it, to complete its goal.

The code below is the form of behaviour that is coordinated with the obstacle detection in [*section 2.3*]. It is in a class called "forward".

This is the action performed by the robot after it had detected the obstacle. It has set rotation speed and travel speed which will cover the distance to the goal as fast as possible,

```
/* Action that would be performed when the robot moves forward after detecting
obstacle*/
@Override
public void action() {
    suppressed = false;
    difp.setRotateSpeed(30);
    difp.setTravelSpeed(50);
    while(!suppressed){
        //Nav.goTo(150,150);
        Nav.followPath();
    }
}
```

There is a take control method set to Boolean True which means that the robot has to move forward after it has detected the obstacle. Without this method the robot won't be able to take control over to its goal position.

```
@Override
public boolean takeControl()
{
    return true;
}
```

4 TRACJECTORY FIGURES

In this stage due to lack of time and the corana which had happened in the last week I wasn't able to get my figures done completely. My code for CSV worked but I unfortunately forgot to save the CSV files. I was only lucky to get my LCD screen picture partially.

The code below is for storing data in the CSV file and drawing a trajectory. It is a thread that I created.

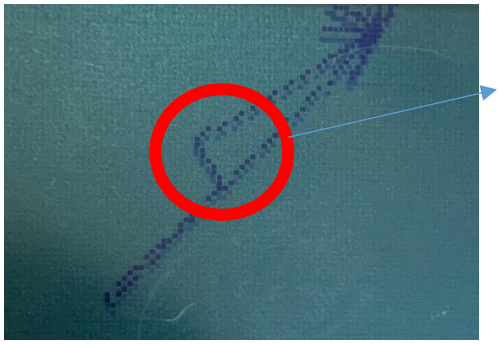
```
// writing current coordinates and other details
try {
    this.DOSOut.writeChars(odom.getPose().getX() + ", " + odom.getPose().getY() +
", " + odom.getPose().getHeading() + "\n");
} catch (IOException e) {
    e.printStackTrace();
}
```

This bit is the code for drawing trajectory on the LCD. The coordinates are divided because the size of LCD is small and the x and y coordinates are really long float numbers. To make them fit on the size of LCD that was the appropriate approach to it.

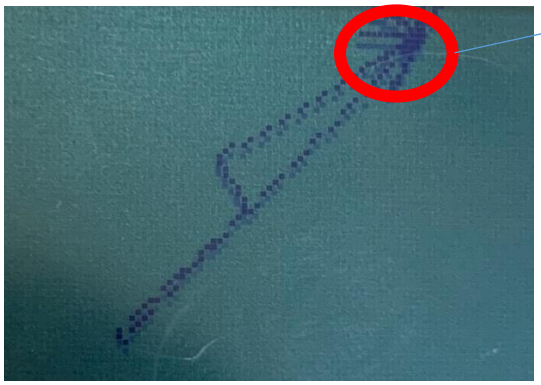
```
// for the drawing of the trajectory to the lcd
int x_coordinate = (int) odom.getPose().getX() / 3 + 10;
int y_coordinate = (int) odom.getPose().getY() / 3 + 10;

LCD.setPixel(x_coordinate, y_coordinate, 1);
```

Results:



This is how my robots trajectory was displayed on the LCD. As you can see the robot detects the obstacle and then rotates accordingly to head to its destination point.



As you see in this the robot is exploring behaviour it has reached the destination and now it's trying to find the red circle in a given distance. It rotates, moves forwards and then heads back. It does that in a loop. Once it has detected the red colour the robot rotates back to its starting point, which is clear on the LCD.

I had attempted two different trials and it had the same trajectory on the LCD screen. While it was heading back to its starting point I didn't place any obstacle in front so my trajectory is a kind of straight line. My testing was still pending for the obstacle which was supposed to be placed when the robot heads back to the starting point. But due to closure of labs I wasn't able to test it again.

5 CONCLUSION

This experiment was one of the most interesting ones I have done in robotics. In the beginning understanding the behaviour of the robot was a bit tough and proceeding with multiple tasks was challenging. But reading through the lectures and doing some research on the behaviour, I was able to code it up to the standard where my results were not that much accurate as we faced many problems because of corona that took place during the last week of university. But it was almost completed. If I had to improve on my work then I'd do the followings.

- Add one more ultrasonic sensor with the one already present. So making it a total of two. The reason it would be a good approach is because then we can have a wider range to detect the obstacle. The uncertainty is going to be much higher and it would be a bit challenging to control two same sensors at the same time.
- Adding Visual Odometer with mono-camera, to achieve high accuracy at reasonable cost. This is to improve the absolute positioning with the ground.
- Using external sensors such as Ultrasonic, Laser scanning to keep track of the moving object detection. It gives high accuracy and resolution. This is to improve the absolute positioning with the ground.
- Placing one more light sensor at the back so that when the robot is in a loop, it heads forward then backwards so there is a high chance for it to detect the red circle.

6 APPENDIX

Movements also known as behaviors. There are five different movements that was coded.

(1) DetectColor

```
package assignmentTwo.movements;

import lejos.nxt.*;
import lejos.robotics.navigation.Navigator;
import lejos.robotics.subsumption.Behavior;

public class detectcolor implements Behavior {

    /*Creating my variables instances*/
    private ColorSensor cS;
    private boolean suppressed = false;
    private Navigator Nav;

    /* Constructors that are sensor port and the navigator */
    public detectcolor(SensorPort sen, Navigator navi) {
        this.cS = new ColorSensor(sen);
        this.Nav = navi;
    }

    /* Methods that will perform some actions when it detects color */
    @Override
    public void action() {
        suppressed = false;
        if(!suppressed){
            Nav.addWaypoint(0, 0);
            Nav.followPath();
        }
    }

    @Override
    public void suppress() {
        suppressed = true;
    };

    @Override
    public boolean takeControl() {
        return (cS.getColorID() == ColorSensor.Color.RED);
    }
}
```

(2) DetectObstacle

```
package assignmentTwo.movements;

import lejos.nxt.*;
import lejos.robotics.navigation.DifferentialPilot;
import lejos.robotics.navigation.Navigator;
import lejos.robotics.subsumption.Behavior;

public class detectObstacle implements Behavior{

    /*These are the constant Instance variables */
    private UltrasonicSensor sonarSensor;
    private DifferentialPilot difp;
    private Navigator Nav;
    private boolean suppressed = false;

    /* Constants value for avoiding distance */
    private static int OBSTACLE_DISTANCE = 17; // When sensor detects obstacle
within 15cm

    /* Constructor */
    public detectObstacle(SensorPort s, DifferentialPilot difp, Navigator navi) {
        this.sonarSensor = new UltrasonicSensor(s);
        this.difp = difp;
        this.Nav = navi;
    }

    /* These are the method performed by the robot when it detects and obstacle */
    @Override
    public void action() {
        suppressed = false;
        if(!suppressed){
            Nav.stop();
            // for rationale on this method, see the writeup/notes
            difp.setTravelSpeed(30);
            difp.setRotateSpeed(30);
            difp.rotate(35);
            difp.travel(35);

        }
    }

    /* when more than one exception would be thrown from the method,
    one of them will be suppressed and not thrown by the compiler.
    This exception is known as the suppressed exceptions*/
    @Override
    public void suppress() {
        suppressed = true;

    }

    @Override
    public boolean takeControl() {
        return (sonarSensor.getDistance() <= OBSTACLE_DISTANCE); }
}
```

(3) Forward

```
package assignmentTwo.movements;
import lejos.robotics.navigation.DifferentialPilot;
import lejos.robotics.navigation.Navigator;
import lejos.robotics.subsumption.Behavior;

public class forward implements Behavior {

    /*These are the constant Instance variables */
    private boolean suppressed = false;
    private DifferentialPilot difp;
    private Navigator Nav;

    /* Constructor */
    public forward(DifferentialPilot difp, Navigator navi) {
        this.difp = difp;
        this.Nav = navi;
    }

    /* Action that would be performed when the robot moves forward after detecting
obstacle*/
    @Override
    public void action() {
        suppressed = false;
        difp.setRotateSpeed(30);
        difp.setTravelSpeed(50);
        while(!suppressed){
            //Nav.goTo(150,150);
            Nav.followPath();
        }
    }

    @Override
    public void suppress()
    {
        suppressed = true;
    }

    @Override
    public boolean takeControl()
    {
        return true;
    }
}
```

(4) SearchColor

```
package assignmentTwo.movements;
import lejos.robotics.navigation.DifferentialPilot;
import lejos.robotics.navigation.Navigator;
import lejos.robotics.subsumption.Behavior;

public class searchcolor implements Behavior {
    /*Creating my variables instances*/
    private boolean suppressed = false;
    private DifferentialPilot difp;
    private Navigator Nav;
```

```
    /* when the robot reaches the end point, it will search for color red by
moving within the given range*/
    private static int FIND_L = 40; // 30cm sweep

    /* Constructors */

    public searchcolor(DifferentialPilot difp, Navigator navi) {
        this.difp = difp;
        this.Nav = navi;
    }

    /* This will perform the robot to go forward, backwards and rotate when it is
at the end point seacrhing for color red*/
    @Override
    public void action() {
        suppressed = false;
        difp.setTravelSpeed(30);
        difp.setRotateSpeed(40);
        while(!suppressed){

            difp.rotate(55);
            if(suppressed){
                break;
            }
            difp.travel(FIND_L /4 );
            if(suppressed){
                break;
            }
            difp.travel(FIND_L / 4);
            if(suppressed){
                break;
            }
            difp.travel(FIND_L / 4);
            if(suppressed){
                break;
            }
            difp.travel(-FIND_L);
            if(suppressed){
                break;
            }
        }
    }

    @Override
    public void suppress() {

        suppressed = true;
    }

    @Override
    public boolean takeControl() {

        return Nav.pathCompleted();
    }
}
```

(5) StoringData

```java
package assignmentTwo.movements;
import lejos.nxt.*;
import lejos.robotics.subsumption.*;

import assignmentTwo.dataoutput;

public class storingdata implements Behavior {

    public boolean suppressed;
    private dataoutput dt;

    public storingdata(dataoutput dt) {
        this.dt = dt;
    }

    @Override
    public void action() {
        suppressed = false;
        dt.setFin(true);
        try {
            Thread.sleep(5000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.exit(0);
    }

    @Override
    public void suppress() {
        suppressed = true;
    };

    @Override
    public boolean takeControl() {
        return Button.RIGHT.isDown();
    }
}
```

This is a Thread for dataouput

```java
package assignmentTwo;
import java.io.*;

import assignment2.DataThread;
import lejos.nxt.*;
import lejos.robotics.localization.OdometryPoseProvider;

public class dataoutput extends Thread {
    /* Instance vars */
    private boolean fin;
    private OdometryPoseProvider odom;
    private FileOutputStream FOSOut;
```

```
    private File data;
    private DataOutputStream DOSOut;




    /* Constant */
    private static int DELTA_T = 250;

    /* Constructor */
    public dataoutput(OdometryPoseProvider odom){
        this.fin = false;
        this.data = new File("datalog.csv");

        try {
            this.FOSOut = new FileOutputStream(data);
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }

        this.odom = odom;
        this.DOSOut = new DataOutputStream(this.FOSOut);
    }

    /* Setters and getters */
    public boolean isFin() {
        return fin;
    }

    public void setFin(boolean fin) {
        this.fin = fin;
    }

    /* The core of the thread */
    public void run(){
        while(true){
            // waiting: so that we don't end up with the mythical 1gb csv file
that breaks excel
            //LCD.drawString("aaaaaaaa",1,1);
            try {
                Thread.sleep(DELTA_T);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

            // writing current coordinates and other details
            try {
                this.DOSOut.writeChars(odom.getPose().getX() + ", " +
odom.getPose().getY() + ", " + odom.getPose().getHeading() + "\n");
            } catch (IOException e) {
                e.printStackTrace();
            }

            // for the drawing of the trajectory to the lcd
            int x_coordinate = (int) odom.getPose().getX() / 3 + 10;
            int y_coordinate = (int) odom.getPose().getY() / 3 + 10;

            LCD.setPixel(x_coordinate, y_coordinate, 1);
```

```
            if(this.isFin()){
                break;
            }
        }

        // closing the file
        try {
            this.DOSOut.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

This is my main file called Final Navigation, where all the movements ar being called in an array.

```
package assignmentTwo;

import assignmentTwo.movements.*;
import lejos.nxt.Button;
import lejos.nxt.LCD;
import lejos.nxt.Motor;
import lejos.nxt.SensorPort;
import lejos.robotics.localization.OdometryPoseProvider;
import lejos.robotics.navigation.DifferentialPilot;
import lejos.robotics.navigation.Navigator;
import lejos.robotics.subsumption.Arbitrator;
import lejos.robotics.subsumption.Behavior;

public class Finalnavigation {

    /* These are the Instance variables */
    private DifferentialPilot difp;

    private Navigator Nav;

    private Arbitrator ab;

    public dataoutput dataoutput;

    private Behavior[] movementAr;



    private OdometryPoseProvider odm;


    /* Calibiration values of the robot */

    private static float WHEEL_DIAMETER = 3.442f;
    private static float TRACK_WIDTH = 19.654f;
```

```
    /* Constructor */
    public Finalnavigation(){
        /*
         * Motors: Ports 1 (left) and 3 (right)
         * Sensors:
         *       Sonar Sensor: port S1
         *       Colour Sensor: port S3
         */
        this.difp = new DifferentialPilot(WHEEL_DIAMETER, TRACK_WIDTH, Motor.A,
Motor.C);
        difp.setRotateSpeed(30);
        //difp.rotate(20);
        this.odm = new OdometryPoseProvider(this.difp);

        this.Nav = new Navigator(this.difp);
        this.dataoutput = new dataoutput(this.getOdom());
        Nav.singleStep(true);
        //LCD.drawString("Ag=" + odm.getPose().getHeading(),2,7);



        /* Behaviour array: this gets passed into the arbitrator.*/
        movementAr= new Behavior[]{
                new forward(this.difp, this.Nav),
                new detectObstacle(SensorPort.S1, this.difp, this.Nav),
                new searchcolor(this.difp, this.Nav),
                new detectcolor(SensorPort.S4, this.Nav),
                new storingdata(this.dataoutput),
        };

        ab = new Arbitrator(movementAr, true);
    }

    /* these are the methods robot will do when it reaches the given coordinates
*/
    public void start(){
        dataoutput.start();
        ab.start();
    }

    public void stop(){
        dataoutput.setFin(true);
        this.Nav.stop();
    }

    public OdometryPoseProvider getOdom(){
        return odm;
    }

    public void addWaypoint(float x, float y){

        this.Nav.addWaypoint(x, y);
    }

    /* Main */
    public static void main(String[] args) throws InterruptedException {
        Button.ENTER.waitForPress();

        // coordinates set to the end point
        nav.addWaypoint(150, 150);
        nav.start();
        nav.stop();
    }
}
```