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

CE215 Robotics

Trajectory Tracking Robot

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

Contents

1 ABSTRACT

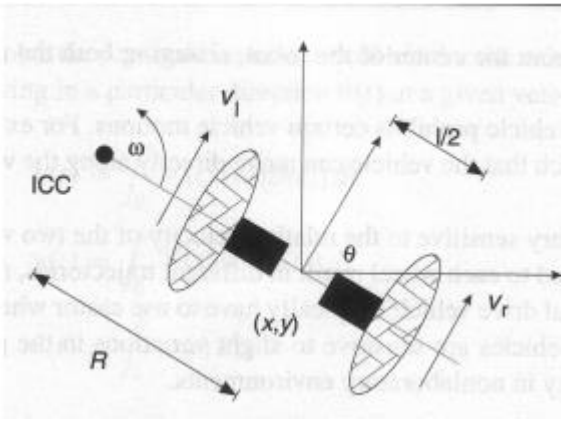
As we know that robots are artificial device that can sense its environment and purposefully act on or in that environment, a machine that can autonomously carry out useful work. Robots have few Programming features such as they are Real time, Robustness, Modularity, Intelligence. Robotics are the future of robotic science in space exploration and other areas of everyday life. Nowadays they have introduced Lego Robots (such as Lego Mindstroms RCX) that we just stick Lego bits together like sensors, motors Lego parts and can program robot to do anything as we wish. It has No mechanical design, electronic feature, system integration. It can easily be programmed using any Language C, Java etc.

2 INTRODUCTION

This report is about the Trajectory Tracking Robot, where we have to program a Lego robot and analyse the different driving mechanism. The program is done using the language LEJOS NXJ and it needs to be constructed in a way such that it travels in a square of length 1 meter. While it is following the path, its trajectory coordinates must be recorded in a CSV file. The sampled robot position should also be displayed on the LCD when the robot is moving. To make this possible we had to calibrate the robot to find its wheel diameter and the track width. Apart from this we had to implement more than one trial at different speeds of the robot, to monitor its positon. More description about its differential drive (how the robot was calibrated, classes that were used), dead reckoning localisation (geometry behind how the robot keeps track of its position) will be explained later in the report.

3 DIFFERENTIAL DRIVE

Many mobile robots use a drive mechanism known as differential drive. It consists of 2 drive wheels mounted on a common axis, and each wheel can independently being driven either forward or backward.



If both of the wheels are programmed to move in forward direction then it moves.

If the wheels programmed are in opposite direction then it either turns clockwise or antic-clockwise.

Knowing how the differential drive mechanism works, there are other parts inside the robot included to control its movement.

The two servo motors on LEGO NXT uses Direct Current motos. Depending on the voltage applied across the brushes, it produces torque. This torque allows it to control the speed of the motor, depending on the current that's being generated.

So when calibrating the **track width** and **wheel diameter.** I had to acquire the Tachometer value (which is an instrument that measures the rotation speed of a shaft, as in a motor).The Incremental encoder produces pulses and these allow us to know the speed, direction and position of the motor drive.

To start of with I first designed a piece of code for a straight line. I used these constructors below:

```
DifferentialPilot straight = new DifferentialPilot(); // to give actions to robot
straight.travel (); // To allow the robot travel a specific distance
Motor.A.getTachoCount (); // returns motor angle degrees/second
Motor.C.getTachoCount (); // returns motor angle degrees/second
```

I set the travel speed to a distance of 10cm first. Using the Vernier caliper I measured the diameter of the wheel which was around **3.35cm**. Then displayed the Tachometer value on the LCD. Using the equation to measure the distance each wheel travelled

n = tachometer value

D = wheel diameter

c = 360 (counts per revolution)

$$\Delta s_l = n_l \frac{\pi D}{c}, \quad \Delta s_r = n_r \frac{\pi D}{c}$$

Test for 10cm distance.

Trial	n_l, n_r	D	c	ΔS_{left}	ΔS_{right}
1	333,333	3.351	**360**	9.58	9.58
2	333,334	3.422		9.93	9.96
3	333,334	3.481		10.17	10.20
Wheel Diameter (D): (**3.35+3.42+3.48**)**/3** = **3.442**					

After the getting the wheel diameter I tested it for 1m length and it gave me around 100.01cm, which was almost accurate.

Initially when finding out the track width I measured it using the ruler which is inaccurate and it gave me **13.5cm**. When I tested it for 90 degrees turn, it wouldn't draw a square so I increased my values and applied trial and error method.

The same concept is used when turning 90 degrees, but we use a different equation. I used this method for rotation control:

```
straight.rotate (); // to allow robot to rotate at a specific angle (eg: 90)
```

n = tachometer value

D = wheel diameter

W = track width

$$\Delta\theta = \frac{(\Delta s_r - \Delta s_l)}{W} = \frac{(n_r - n_l)}{c}\frac{\pi D}{W}$$

Trial	n_l,	n_r	ΔS_{left}	ΔS_{right}	ΔS_{r-l}	D	c	Δθ	W
1	3062	4028	91.97	122.98	30.01	**3.442**	**360**	**π/2**	19.128
2	2815	3844	84.55	115.46	30.91				19.878
3	3090	4026	92.81	123.92	31.11				19.805
Track Width (W): (**19.128+19.878+19.805**)**/3** = **19.654**									

My final testing of the track width and the diameter I found were almost accurate as it gave a nearly a perfect square.

4 SOFTWARE CODE

This the method I used for creating a csv file called rdata. The method DataOutputStream allows the final output of the odometry pose to be written.

```
FileOutputStream out = null;
FileR datawritten = new File("rdata.csv");
try{
    out = new FileOutputStream(datawritten);
} catch (FileNotFoundException u) {
    System.err.println("Failed to open");}
DataOutputStream dataout = new DataOutputStream(out);
```

The Odometry pose is used to keep track of the robots pose. The differential pilot will keep the odometry pose provider informed about motor movements etc. There are different methods used to get the poses.

Firstly I had to add the odometry constructor which would then allow me to get the robot poses.

```
OdometryPoseProvider straightopp = new OdometryPoseProvider(straight);
straight.addMoveListener(straightopp); // We add a listener so that the
    differential pilot can inform the odometry about its position
```

The odometry methods I used to get the X and Y pose of the robot are.

```
straightopp.getPose().getX()
straightopp.getPose().getY()
```

This is the code I implemented to allow the robot move in a square and using the dataoutputstream to write the the values of X and Y coordinates of the robot in the excel file.

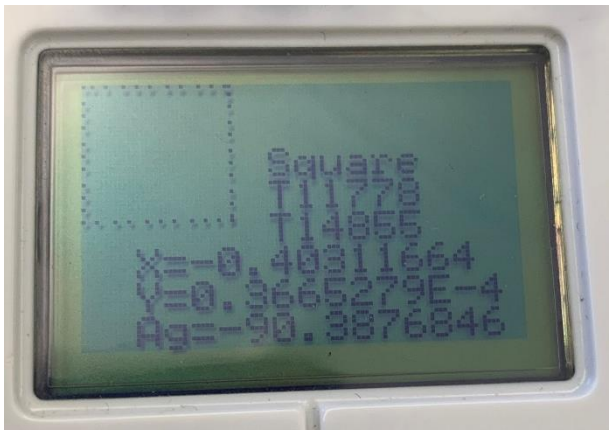
```
for (int i = 0; i < 4; i++) {
    straight.travel(100, true);
    while (straight.isMoving()) {
        try {
            dataout.writeChars(straightopp.getPose().getX()
         + "," + straightopp.getPose().getY() + ", " + "\n");
            Thread.sleep(500);
        } catch (IOException | InterruptedException e) {
          System.err.println("Failed to write the values");
        }
    }
```

5 EVALUATION OF ROBOT

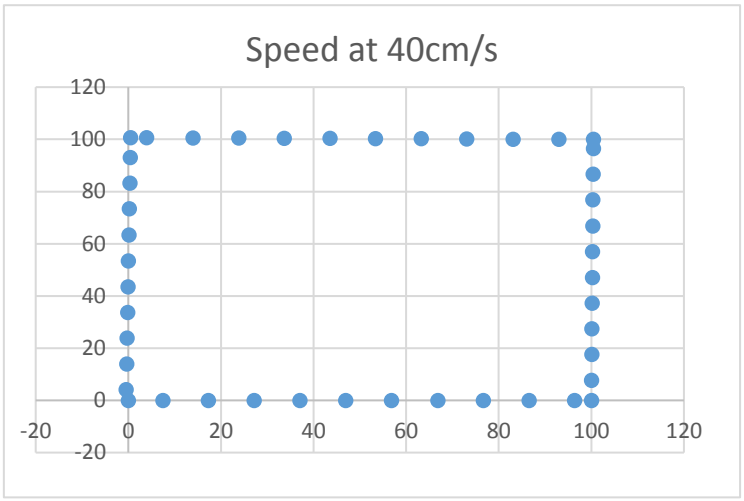
To draw the trajectory displayed on the LCD I used these method which allowed me to display to display a square.

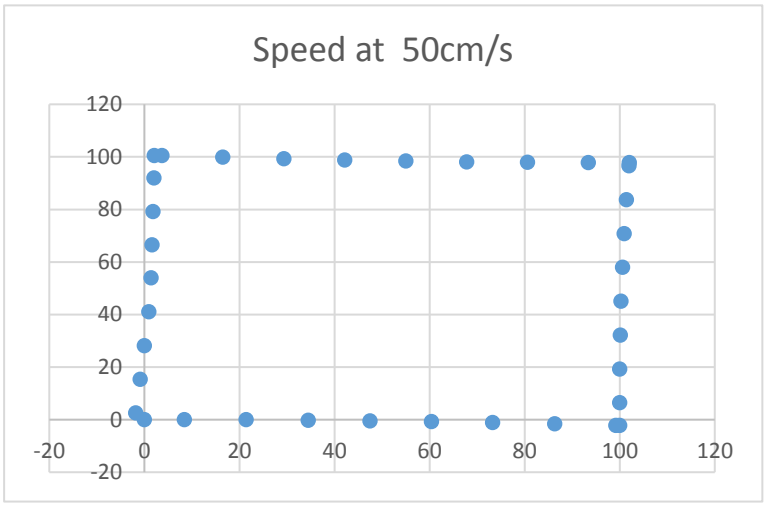
```
int x_coordinate = (int) straightopp.getPose().getX() / 3;
int y_coordinate = (int) straightopp.getPose().getY() / 3;
LCD.setPixel(x_coordinate, y_coordinate, 1);
```

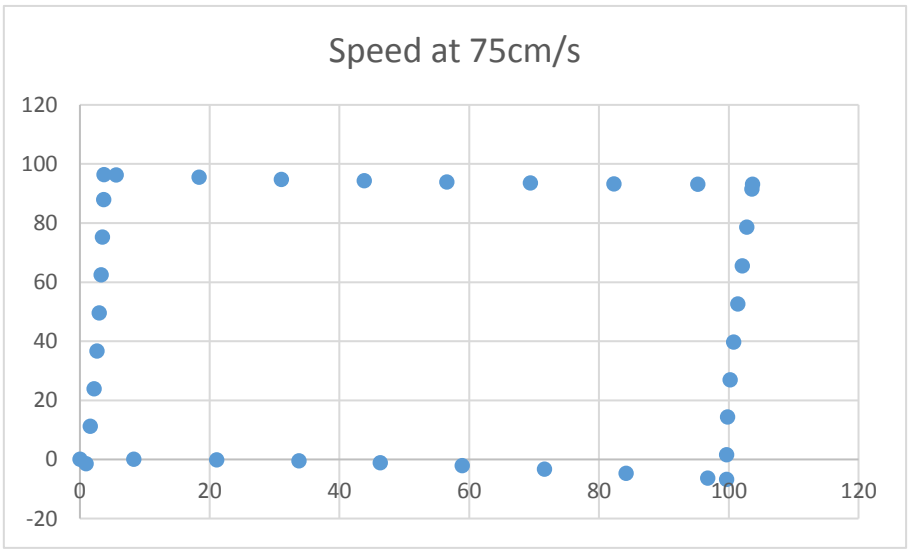
This is the final output of the trajectory displayed.



To carry on with the perfect square I got, I tested my robot at various speeds like (**50cm/s, 75cm/s, 0cm/s)** to see how to robot behaves. The values of the poses under various speeds were recorded in the CSV file. Below are the graphs trajectory:







Based on the trajectories that I drew in excel, states that the robot flickers at various points which lets it not have a proper square built. The reason it haves in that way it is because the motor speed is controlled by pulse-width modulation (cycle square wave), this effectively turns motor on and off fast, the longer it is on it produces more torque and faster it will go. The rotation speed is fast as well which results in a different behavior of the robot. When the speed is set high the robot moves faster and it produces greater uncertainties when it takes turn. But when kept at low speed the odometry pose is accurate there are lower uncertainties and this results in almost a perfect trajectory.

6 DEAD RECKONING LOCALISATION

DRL is a technique that calculates position change of the robot based on the robot position. From this we know that when the accurate absolute positioning is not possible DRL plays and important role in it. There couple of reasons why the robot doesn't do a perfect trajectory square. The main reasons are:

- Odometry pose is different compared to ground tools, meaning that it is only used for calibration purpose. It actually doesn't draw a trajectory square based on the movement of poses when the robot is kept on the ground.

Based of the (unknown disturbances), which works well controlling the quadrotor close but reduces its trajectory tracking accuracy progressively with increasing speed.

To improve absolute positioning (keeping track of its pose when on ground) of the robot we can use:

- Adding Visual Odometry with mono-camera, to achieve high accuracy at reasonable cost.
- Using external sensors such as Ultrasonic, Laser scanning to keep track of the moving object detection. It gives high accuracy and resolution
- The stereo camera or LIDAR can perform robust and precise moving object detection.

7 FULL CODE

```
package Assignment;

import lejos.nxt.Button;

import lejos.nxt.LCD;

import lejos.nxt.Motor;

import java.io.*;

import lejos.robotics.localization.OdometryPoseProvider;

import lejos.robotics.navigation.DifferentialPilot;

import lejos.util.Delay;

import java.io.FileOutputStream;


public class assignment1 {

    public static void main(String[] args) throws InterruptedException {

        // Display initial text in between the square path

        LCD.drawString("Square",7,2 );

        try{

            Thread.sleep(200);

        }catch (InterruptedException e){ }

        Delay.msDelay(300);

        // Button press

        Button.ENTER.waitForPress();


        // Creating file

        FileOutputStream out = null;

        File datawritten = new File("rdata.csv");
```

```
        try{
            out = new FileOutputStream(datawritten);
        } catch (FileNotFoundException u) {
            System.err.println("Failed to open");
        }
        DataOutputStream dataout = new DataOutputStream(out);
        // using differential pilot to give actions to robot
        DifferentialPilot straight = new DifferentialPilot(3.442f, 19.654f,
Motor.A, Motor.C);
        // To get robot pose
        OdometryPoseProvider straightopp = new
OdometryPoseProvider(straight);
        straight.addMoveListener(straightopp);
        //this sets position of the starting point of the square
        LCD.setPixel(0,0,1);
        // Loop that does the square
        for (int i = 0; i < 4; i++) {
            straight.travel(100, true);
            while (straight.isMoving()) {
                try {
                    dataout.writeChars(straightopp.getPose().getX() + "," +
straightopp.getPose().getY() + ", " + "\n");
                    Thread.sleep(500);

                } catch (IOException | InterruptedException e) {
```

```
                    System.err.println("Failed to write the values");
                }
                // set pixel draws the square based on the x and y coordinates
                int x_coordinate = (int) straightopp.getPose().getX() / 3;
                int y_coordinate = (int) straightopp.getPose().getY() / 3;
                LCD.setPixel(x_coordinate, y_coordinate, 1);
                LCD.drawString("T" + Motor.A.getTachoCount(),7,3);
                LCD.drawString("T" + Motor.C.getTachoCount(),7,4);
                LCD.drawString("X="+ straightopp.getPose().getX(),2,5);
                LCD.drawString("Y=" + straightopp.getPose().getY(),2,6);
                LCD.drawString("Ag="                                    +
straightopp.getPose().getHeading(),2,7);}
            straight.rotate(90); }
        Thread.sleep(5000);
        //closing DATAOUTPUT FILE
        try{
            dataout.close();
        }catch (IOException e) {
            System.err.println("Failed to close");
        }
        try{
            Thread.sleep(5000);
        }catch (InterruptedException e){ }
    }
}
```