2020

Robotics – CE215

ASSIGNMENT 1

SHINSKIY, MAXIM 1804336

Contents

Introduction

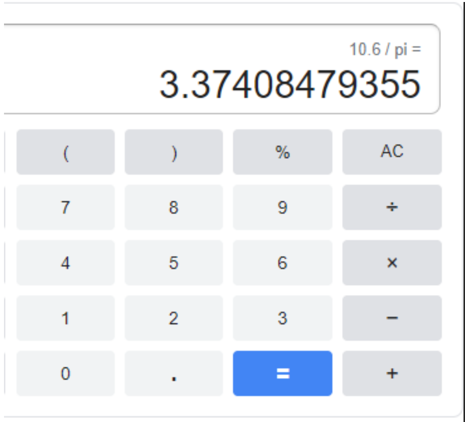
In this assignment we were required to build a robot to complete the task. One of the obligations was that it uses the Differential Pilot class provided by LeJos_NXJ library. This class provides methods to move the robot more precisely using predefined track width and wheel diameter. Both of these variables we have to calibrate manually. All the code was written in Java. The task was to make robot travel in a 1 meter square path and get the coordinates with Dead Reckoning. The coordinates then need to be written into the CSV file.

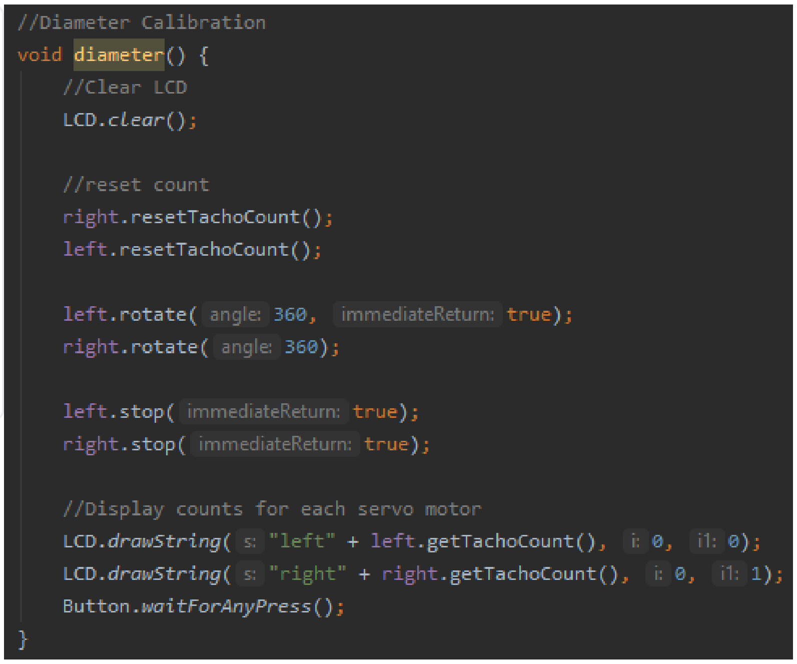
Differential Drive Robot

To use differential pilot class I first had to do the calibration, so the class can use the values to calculate the precise trajectory. Then more accurate the values are then more accurate the path will be.

Diameter

First to find diameter we need to use a measuring tape and make a robot move for certain angle of rotation of a servo motor (move in a straight line), then we can check whether the encoder counts match with the number we set in the code. The encoder counts are read by `.getTachoCount()` method. 1 full revolution of a servo motor will be equal to 360 counts, so one count per one degree. The number to be set can be anything but the more the better. I used 360. This value goes into `.rotate()` method for each motor. The statement that is first to be executed must have `immediate return: true` so the motors rotate at the same time. `.stop(true)` will let robot to stop almost instantly, without moving extra half a centimeter. Afterwards, put obtained distance into formula: $\Delta s = \frac{(n_r+n_l)}{2}\frac{\pi D}{c}$ where c is a constant and equals to 360 degrees. After rearranging the formula looks like this: $D = \frac{2c}{(n_r+n_l)}\frac{\Delta s}{\pi}$ The distance measured was 10.6 centimeters. The value I calculated for diameter is $\mathrm{D} = 3.37$ centimeters.

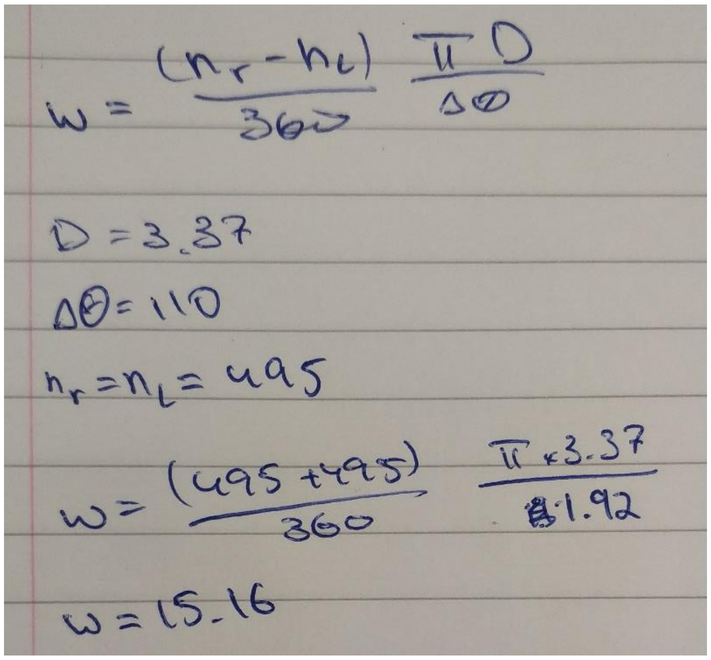




```
//Diameter Calibration
void diameter() {
    //Clear LCD
    LCD.clear();

    //reset count
    right.resetTachoCount();
    left.resetTachoCount();

    left.rotate( angle: 360,  immediateReturn: true);
    right.rotate( angle: 360);

    left.stop( immediateReturn: true);
    right.stop( immediateReturn: true);

    //Display counts for each servo motor
    LCD.drawString( s: "left" + left.getTachoCount(),  i: 0,  i1: 0);
    LCD.drawString( s: "right" + right.getTachoCount(),  i: 0,  i1: 1);
    Button.waitForAnyPress();
}
```

Track Width

To Calculate track width, I printed out a picture of a protractor on a paper and stick it to the floor with a tape. This lets me find the angle for which robot turns. The way to calibrate the track width is such that we create a Differential Pilot with diameter that we found and estimated track width. Then we can use method `dp.rotate(90)` of this class to cause the robot to turn for 90 degrees. The difference between the method used in diameter calibration is that this method applies to differential pilot class rather than motor class. Differential pilot takes the values of diameter of the wheel and track width into account to calculate the angle of rotation, also as an argument method from dp class takes actual angle, rather than angle of rotation of servo motor. After robot turned for what it thinks is 90 but actually not, I can measure the angle it turned for, using the protractor I printed. This value is then used in the formula: $W = \frac{(n_r - n_l)}{c}\frac{\pi D}{\Delta\theta}$. n_r and n_l are the encoder counts that I read off the screen of the robot. They are obtained by the same method as in diameter calibration, `.getTachoCount().` The values I measured were: $n_r = -n_l = 495$, $\Delta\theta = 110° = 1.91986^c$, $D = 3.37cm$. In the end the result value calculated for track width is 15.16 centimeters.



$$w = \frac{(n_r - n_l)}{360}\frac{\pi D}{\Delta\theta}$$

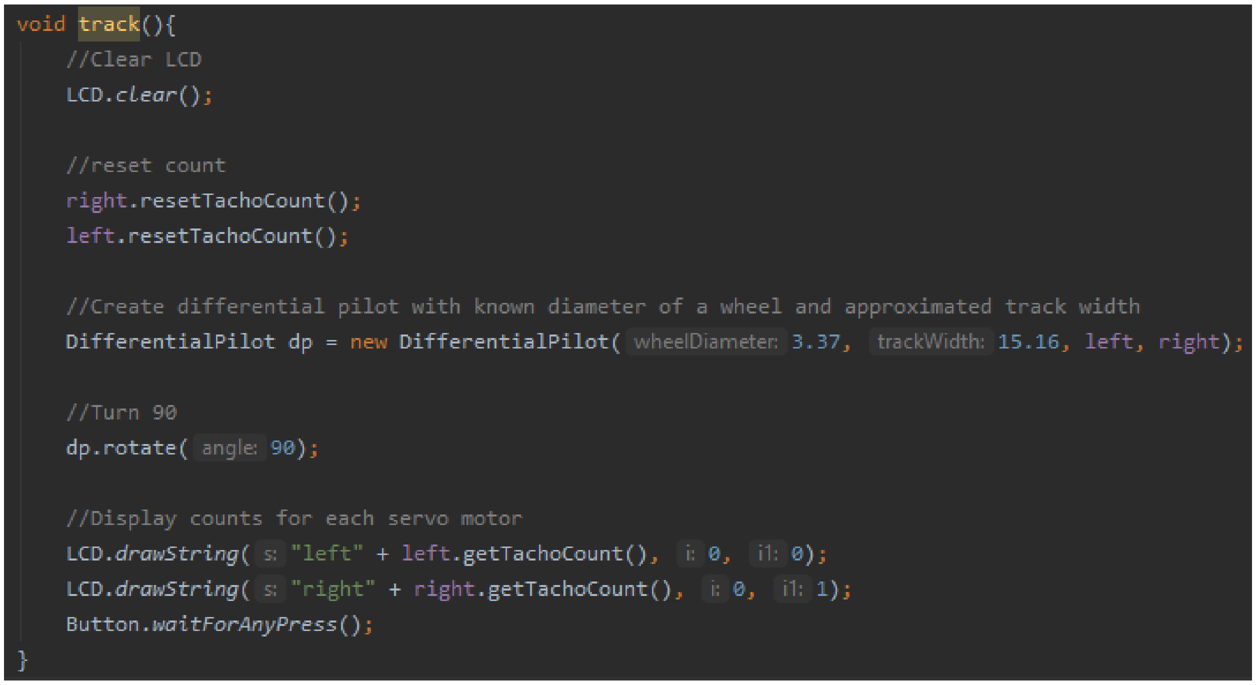
$D = 3.37$

$\Delta\theta = 110$

$n_r = n_L = 495$

$$w = \frac{(495 + 495)}{360}\frac{\pi \times 3.37}{1.92}$$

$w = 15.16$


```
void track(){
    //Clear LCD
    LCD.clear();

    //reset count
    right.resetTachoCount();
    left.resetTachoCount();

    //Create differential pilot with known diameter of a wheel and approximated track width
    DifferentialPilot dp = new DifferentialPilot( wheelDiameter: 3.37,  trackWidth: 15.16, left, right);

    //Turn 90
    dp.rotate( angle: 90);

    //Display counts for each servo motor
    LCD.drawString( s: "left" + left.getTachoCount(),  i: 0,  i1: 0);
    LCD.drawString( s: "right" + right.getTachoCount(),  i: 0,  i1: 1);
    Button.waitForAnyPress();
}
```

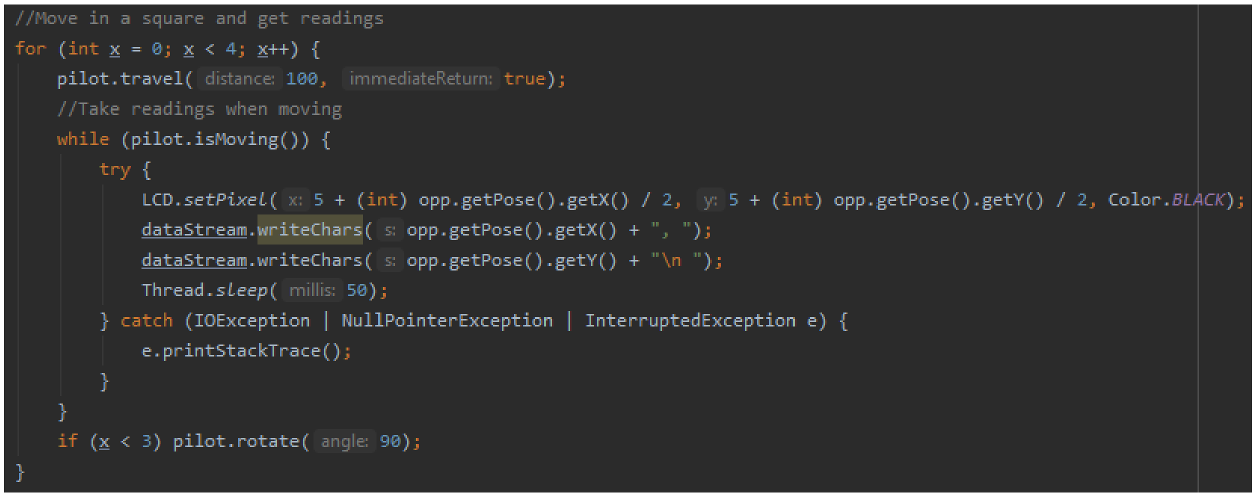
Software Code

To complete the task the robot supposed to move in the square and print its trajectory onto the LCD screen. We can split this into three sub tasks: movement, positioning and writing into file.

Movement is done inside a for loop that controls number of times that the robot goes straight and turns. I made it in a way that it performs only three turns, so after the final movement straight it won't turn. I believe this would give more accurate result and is completed using an if statement to check how many turns has been already made. The movement is done using methods from Differential pilot class with parameters obtained after calibration. `pilot.travel(100, true)` allows to move in the straight line for certain distance (100 centimeters in this case) and `pilot.rotate(90)` allows to turn for a certain angle (90 degrees).

Positioning and writing into file is done in the same while loop that is inside a for loop, which let us write and move simultaneously. It also put inside try-catch block to catch IO, Nullpointer and Interrupted exceptions that may be caused by some methods inside the loop. To track robots position I used OdometryPoseProvider class. `opp.getPose().getX()` and `opp.getPose().getY()` provide X and Y coordinates of the robot, these coordinates are then first printed onto LCD screen and the written into a file. LCD.setPixel() method allows to print just one pixel on the screen at a time. This method takes in two integer arguments of coordinates on the screen and the color object. To fit the coordinates onto the screen I divided them by 2 and added 5 to shift them form the very left top corner of the screen.

Afterwards the coordinates are written to CSV file. To write data we need to create a file and initialize FileOutput and DataOutput streams. After initializing them I input the names of the columns straight away to label them in the file. I used `dataStream.writeChars()` for it. To move to the next block I put `","` in the end of the string to be written and to jump to next line I used `"\n"` in the end. In the same way I inserted data into file inside the while loop. To avoid overfilling the file, the code didn't write the coordinates every millisecond but every after every time it wrote one it would wait for 50 milliseconds. This was done using `Thread.sleep(50)`. With this included I always had around 340-350 coordinate sets (x, y).

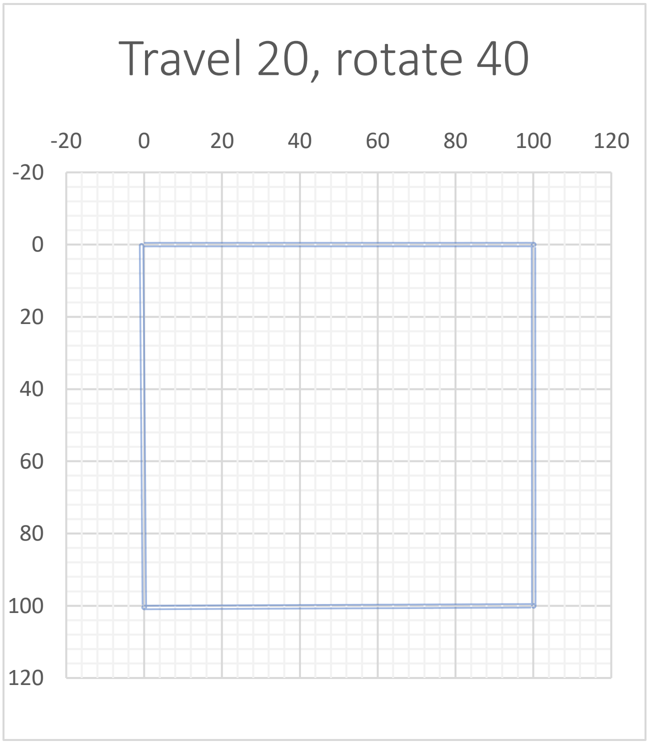


```
//Move in a square and get readings
for (int x = 0; x < 4; x++) {
    pilot.travel( distance: 100, immediateReturn: true);
    //Take readings when moving
    while (pilot.isMoving()) {
        try {
            LCD.setPixel( x: 5 + (int) opp.getPose().getX() / 2, y: 5 + (int) opp.getPose().getY() / 2, Color.BLACK);
            dataStream.writeChars( s: opp.getPose().getX() + ", ");
            dataStream.writeChars( s: opp.getPose().getY() + "\n ");
            Thread.sleep( millis: 50);
        } catch (IOException | NullPointerException | InterruptedException e) {
            e.printStackTrace();
        }
    }
    if (x < 3) pilot.rotate( angle: 90);
}
```

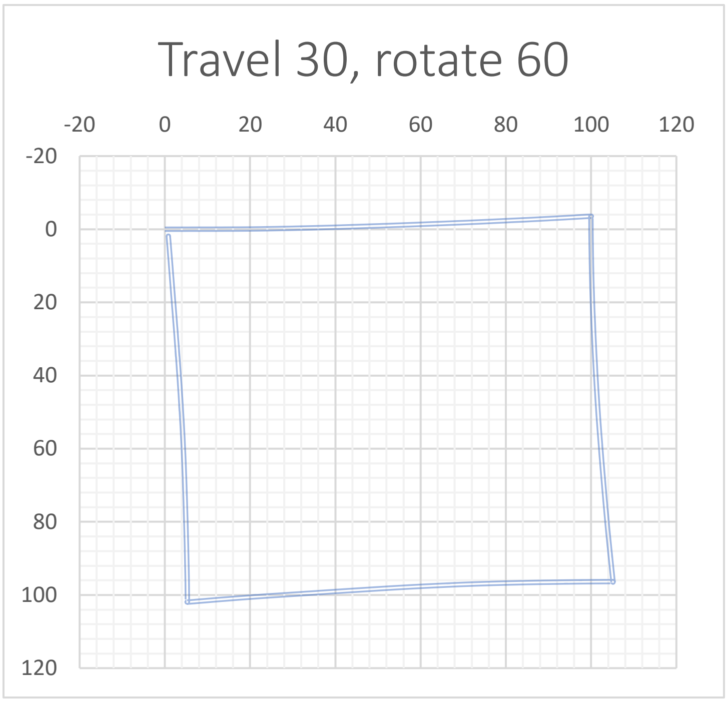
Evaluation

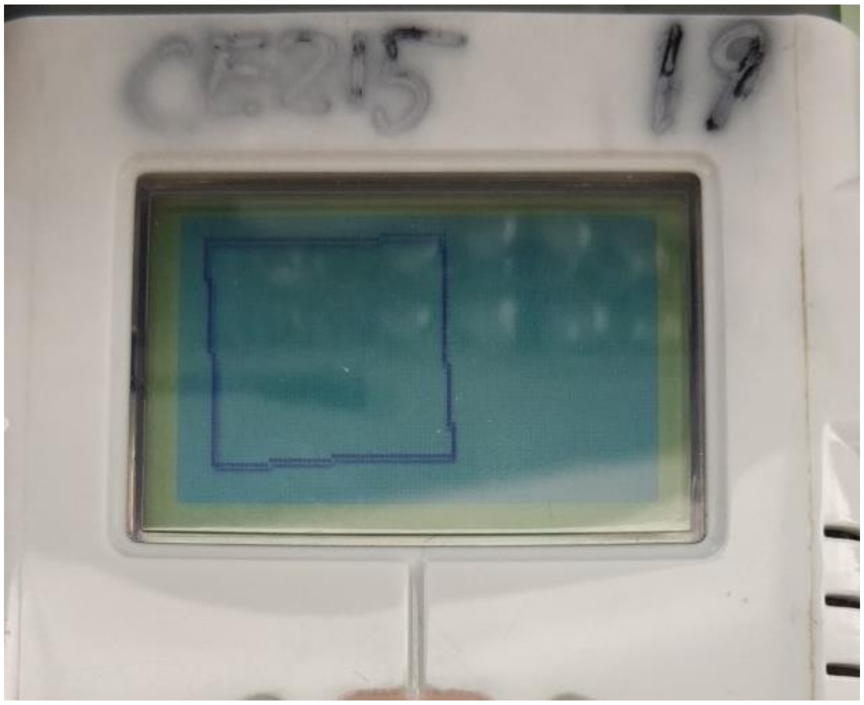
Even though the perfect square wasn't required, I could achieve it with one of the groups of speed. Overall I used three test sets: travel speed - 20 rotate speed - 40, travel speed - 30 rotate speed – 60 and travel speed - 40 rotate speed - 80. In all cases the car was moving in the clockwise direction.

This is the best result out of three runs. On the screen it built perfect square, the graph, more or less, is also perfect.

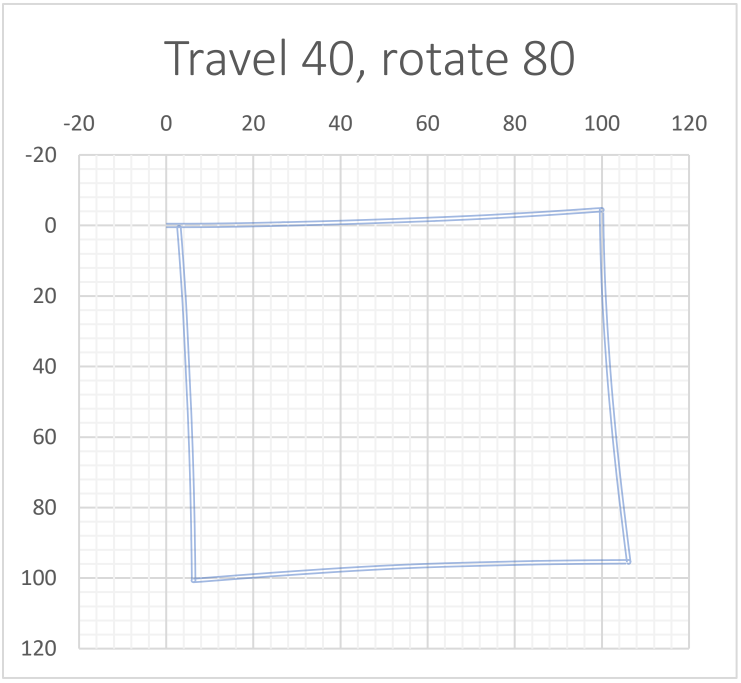


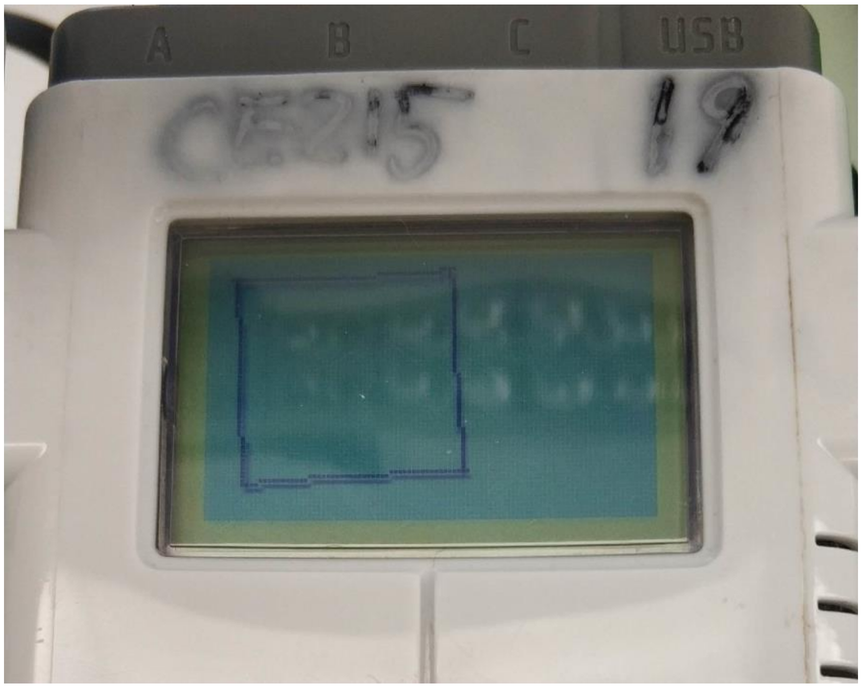
Here, with increased speed the trajectory changes when moving in a straight line. At turning angles are still stay at about 90 degrees. Robot seems to move little bit more to the left.





In the last run the path was the worst because not only it didn't move in the straight line but the turning got worse and now car turned for about 100 degrees each time. From the graph it can be observed that car didn't return to the same point as it left from.





Dead Reckoning Localization comments

As seen from experiments then greater the speed then worst the path gets. Same but to a lesser degree applies to the speed of rotation. In the case of travel speed, the path got worse after increasing the speed twice or by +10 units, for turnings, the change is visible after increasing it by also twice or +40 units.

On the graph it looked like trajectory was perfect, but in the reality and from observations it was still not that good. The problem may vary from the fact that robot used tracks that are not smooth to the difference in voltages supplied to the motors. The best way to increase the accuracy of the trajectory would be to get better calibration. The thing about calibration is that even though I did it precisely, it is still not ideal. In fact it won't be ever ideal as in the every experiment the path will differ slightly. Another problem could be that my robot had a handle on top of it that I built to make it hold easier (picture in the Track Width chapter), this changed the center of mass of the robot, which could shift the path. Or some more hardware problems, like as said earlier voltage supplied might vary because of the different charge in every battery cell.

Appendix

Main code

```
import lejos.nxt.*;
import lejos.robotics.Color;
import lejos.robotics.localization.OdometryPoseProvider;
import lejos.robotics.navigation.DifferentialPilot;
import lejos.robotics.navigation.Pose;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;

public class Assignment1 {
    public static void main(String[] args) {
        //Motors
        final NXTRegulatedMotor LEFT = Motor.C;
        final NXTRegulatedMotor RIGHT = Motor.A;
        final int trSpeed = 30;
        final int rtSpeed = 60;

        //File & Streams
        File file = new File("data.csv");
        FileOutputStream fileStream = null;
        DataOutputStream dataStream = null;

        //Creation and setup of file and streams
        try {
            file.createNewFile();
            fileStream = new FileOutputStream(file);
            dataStream = new DataOutputStream(fileStream);
            dataStream.writeChars("Travel speed: " + trSpeed + ",");
            dataStream.writeChars("Rotation speed: " + rtSpeed + "\n");
            dataStream.writeChars("x: " + ",");
            dataStream.writeChars("y: " + "\n");
        } catch (IOException e) {
            e.printStackTrace();
        }
        //Diff. pilot and odometry constructors
        DifferentialPilot pilot = new DifferentialPilot(3.37, 15.16, LEFT, RIGHT);
        OdometryPoseProvider opp = new OdometryPoseProvider(pilot);
        pilot.addMoveListener(opp);

        //Define speeds
        pilot.setTravelSpeed(trSpeed);
        pilot.setRotateSpeed(rtSpeed);

        //Reset position
        opp.setPose(new Pose(0, 0, 0));

        //Move in a square and get readings
        for (int x = 0; x < 4; x++) {
            pilot.travel(100, true);
            //Take readings when moving
            while (pilot.isMoving()) {
                try {
                    LCD.setPixel(5 + (int) opp.getPose().getX() / 2, 5 + (int)
opp.getPose().getY() / 2, Color.BLACK);
                    dataStream.writeChars(opp.getPose().getX() + ", ");
                    dataStream.writeChars(opp.getPose().getY() + "\n ");
                    Thread.sleep(50);
                } catch (IOException | NullPointerException | InterruptedException e)
{
                    e.printStackTrace();
                }
            }
            if (x < 3) pilot.rotate(90);
        }
        //close DataOutputStream and FileOutputStream
        try {
            fileStream.close();
            dataStream.close();
        } catch (IOException | NullPointerException e) {
            e.printStackTrace();
        }
        Button.waitForAnyPress();
    }
}
```

Calibration

```
import lejos.nxt.Button;
import lejos.nxt.LCD;
import lejos.nxt.Motor;
import lejos.nxt.NXTRegulatedMotor;
import lejos.robotics.navigation.DifferentialPilot;
import java.lang.*;

public class Calibration {
    final NXTRegulatedMotor right = Motor.A;
    final NXTRegulatedMotor left = Motor.C;

    Calibration() {
        diameter();
        track();
    }

    //Diameter Calibration
    void diameter() {
        //Clear LCD
        LCD.clear();

        //reset count
        right.resetTachoCount();
        left.resetTachoCount();

        left.rotate(360, true);
        right.rotate(360);

        left.stop(true);
        right.stop(true);

        //Display counts for each servo motor
        LCD.drawString("left" + left.getTachoCount(), 0, 0);
        LCD.drawString("right" + right.getTachoCount(), 0, 1);
        Button.waitForAnyPress();
    }

    void track(){
        //Clear LCD
        LCD.clear();

        //reset count
        right.resetTachoCount();
        left.resetTachoCount();

        //Create differential pilot with known diameter of a wheel and approximated
track width
        DifferentialPilot dp = new DifferentialPilot(3.37, 15.16, left, right);

        //Turn 90
        dp.rotate(90);

        //Display counts for each servo motor
        LCD.drawString("left" + left.getTachoCount(), 0, 0);
        LCD.drawString("right" + right.getTachoCount(), 0, 1);
        Button.waitForAnyPress();
    }

    public static void main(String[] args){
        new Calibration();
    }
}
```