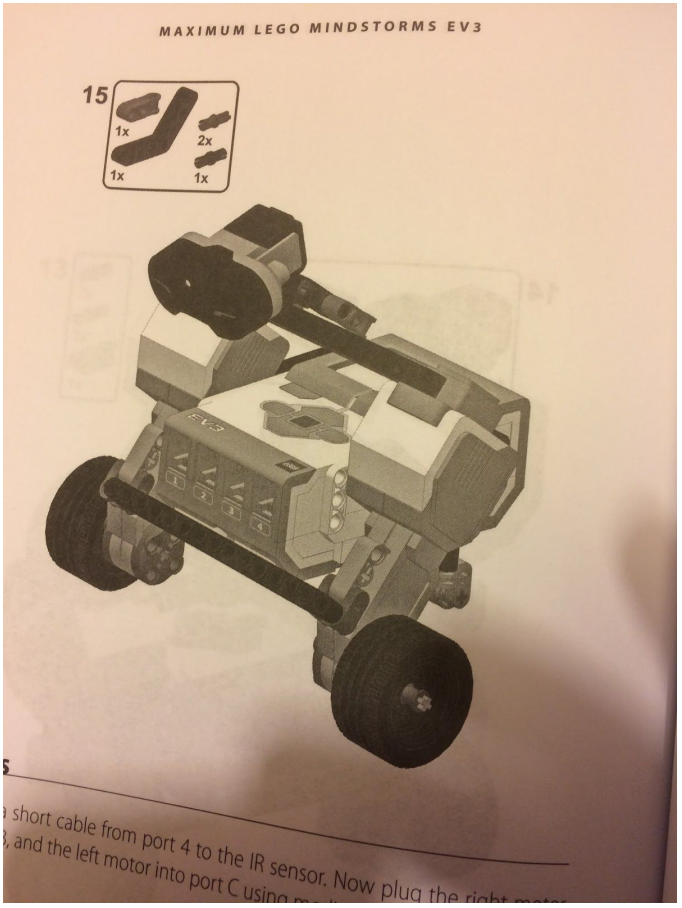
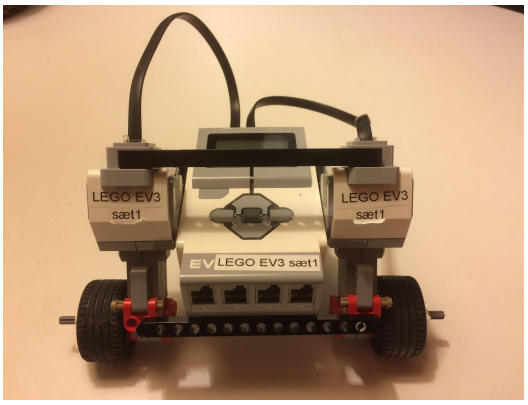
A picture of the robot and a discussions of its design:

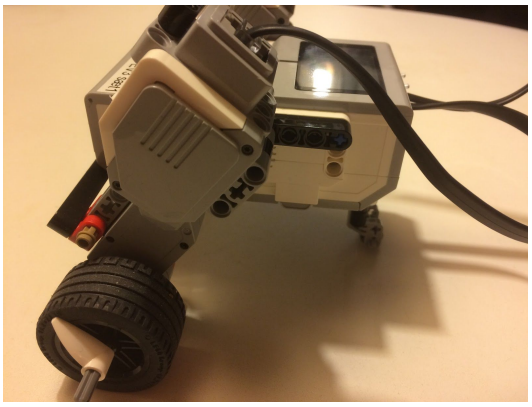
For the assignment I used the book Maximum Lego Mindstorms EV3 for inspiration in the design of the robot. As for the design I chose the robot from chapter 22 which chapter is writing about Moves. In that section they use a simple differential drive robot that is called Rov3r.
The original design is using an IR sensor but in this current design since the assignment explicitly states that we should not use sensors has no sensor on it.



The Rov3r robot as it is in the book with sensors

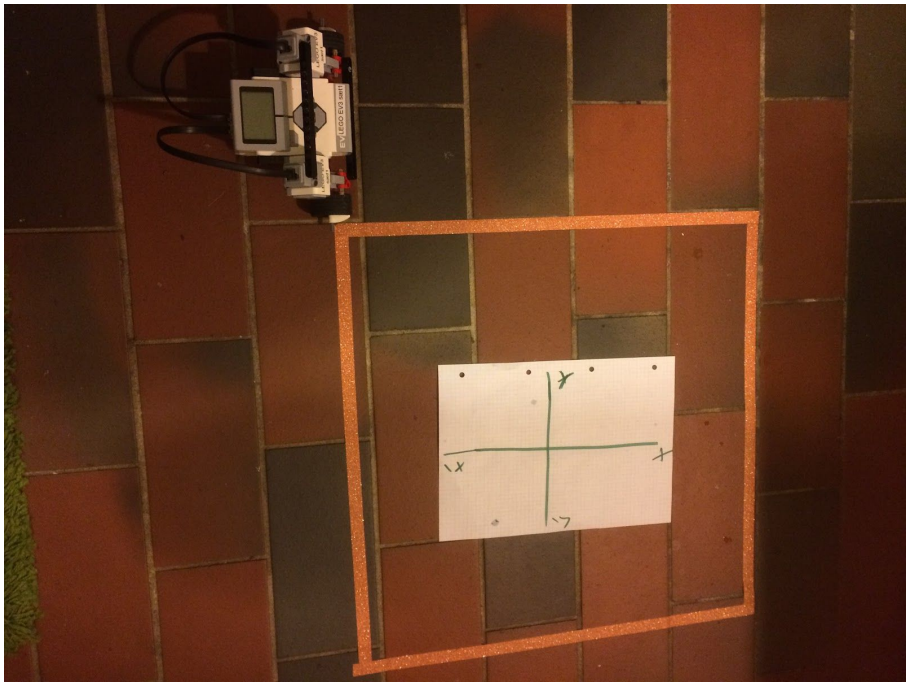
The design of my robot follows the design description that is presented in the book except the sensor. There are three basic parameters that are necessary in this case: the tire diameter, the tire width and the motors. The tires in the EV3 kit are 4.32cm The Rov3r robot has a track width of 15.2 cm but it is adjusted to 16 cm. The motors for the left and the right wheels are the motors that are plugged into the B and C ports.





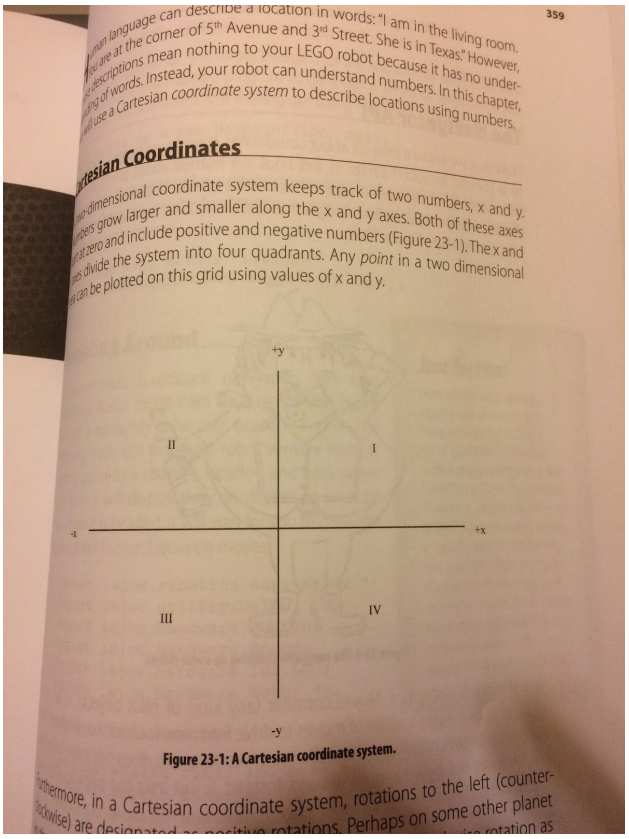
<u>A drawing showing the route, including also the selected points for version 3.:</u>

The route I chose to use is a square one. The length of the route is 2 meters so consequently each side is 50 cm long.



The 2 meter long square route

On the picture we can see the route and the starting point of the route is indicated by the robot and its direction.
In order to describe locations using numbers the book is using the Cartesian coordinate system. It is a two dimensional coordinate system that keeps track of two numbers which are x and y.



359

...human language can describe a location in words: "I am in the living room. You are at the corner of 5th Avenue and 3rd Street. She is in Texas." However, these descriptions mean nothing to your LEGO robot because it has no understanding of words. Instead, your robot can understand numbers. In this chapter, you will use a Cartesian *coordinate system* to describe locations using numbers.

...rtesian Coordinates

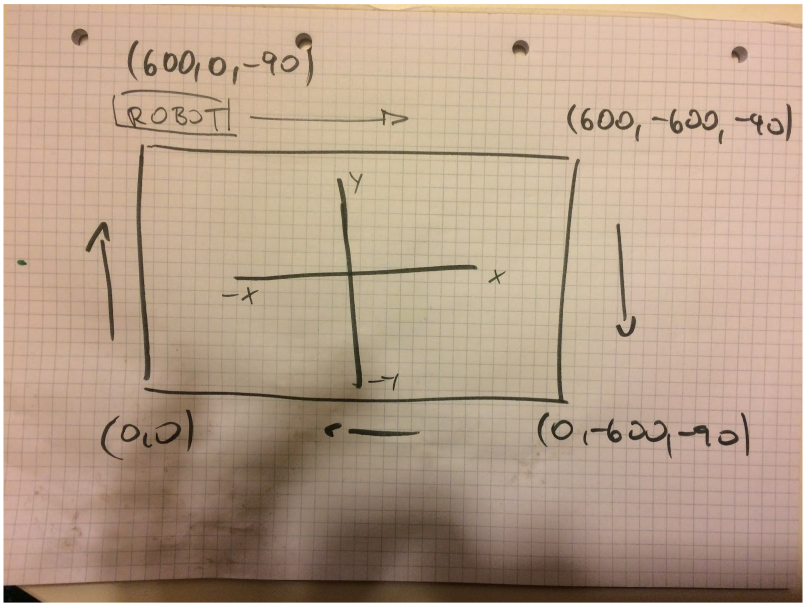
...o-dimensional coordinate system keeps track of two numbers, x and y. ...bers grow larger and smaller along the x and y axes. Both of these axes ...at zero and include positive and negative numbers (Figure 23-1). The x and ...es divide the system into four quadrants. Any *point* in a two dimensional ...n be plotted on this grid using values of x and y.

Figure 23-1: A Cartesian coordinate system.

...rthermore, in a Cartesian coordinate system, rotations to the left (counter-...ckwise) are desi... positive rotations. Perhaps on some other planet ... rotation as

The Cartesian coordinate system

The interesting thing to mention is that in this system the rotation to the left is considered as positive rotation. So if we rotate +90 degrees it means the robot will rotate counter-clockwise one-quarter and likewise a rotation of -90 degrees is equivalent to a clockwise one-quarter turn.



The selected points for version 3.

For making use of these coordinates the Navigator class is used that tells the pilot how to drive to a specific location.

The programs (with short explanation):
The motor class and the related code. For further information I used the official documentation. As it turns out the book and its code snippets are not up-to-date since the book was published in 2004.

```
public static void Motor_Class() {

    Motor.B.setSpeed(450);
    Motor.C.setSpeed(450);
    Motor.B.forward();
    Motor.C.forward();
    try{Thread.sleep(3000);}catch(Exception e){}
    Motor.B.rotate(90);
    try{Thread.sleep(1150);}catch(Exception e){}

    Motor.B.setSpeed(450);
    Motor.C.setSpeed(450);
    Motor.B.forward();
    Motor.C.forward();
    try{Thread.sleep(3000);}catch(Exception e){}
    Motor.B.rotate(90);
    try{Thread.sleep(1160);}catch(Exception e){}

    Motor.B.setSpeed(450);
    Motor.C.setSpeed(450);
    Motor.B.forward();
    Motor.C.forward();
    try{Thread.sleep(3000);}catch(Exception e){}
    Motor.B.rotate(90);
    try{Thread.sleep(1160);}catch(Exception e){}

    Motor.B.setSpeed(450);
    Motor.C.setSpeed(450);
    Motor.B.forward();
    Motor.C.forward();
    try{Thread.sleep(3000);}catch(Exception e){}
    Motor.B.rotate(90);
    try{Thread.sleep(1160);}catch(Exception e){}

    Motor.B.stop();
    Motor.C.stop();
}
```

Small adjustments were needed to be made since it happened during tests that the Pilot robot did not turn 90 degrees but little less and therefore it was not able to follow the path.

MovePilot class and the related code.

```
public static void Move_Pilot_Class() {

    Wheel wheelL = WheeledChassis.modelWheel(Motor.B, 43.2).offset(-80);
    Wheel wheelR = WheeledChassis.modelWheel(Motor.C, 43.2).offset(80);

    Chassis chassis = new WheeledChassis(new Wheel[] { wheelL, wheelR }, WheeledChassis.TYPE_DIFFERENTIAL);

    MovePilot pilot = new MovePilot(chassis);

    pilot.setLinearSpeed(100);
    pilot.travel(690);
    pilot.rotate(-90);

    pilot.setLinearSpeed(100);
    pilot.travel(750);
    pilot.rotate(-90);

    pilot.setLinearSpeed(100);
    pilot.travel(740);
    pilot.rotate(-90);

    pilot.setLinearSpeed(100);
    pilot.travel(700);
    pilot.rotate(-90);

    pilot.stop();
}
```

Small adjustments regarding the parameter to the travel() function was necessary to be made due to the rotation degree and due to the distance the Pilot robot needed to travel.

Navigator Class and the related code.

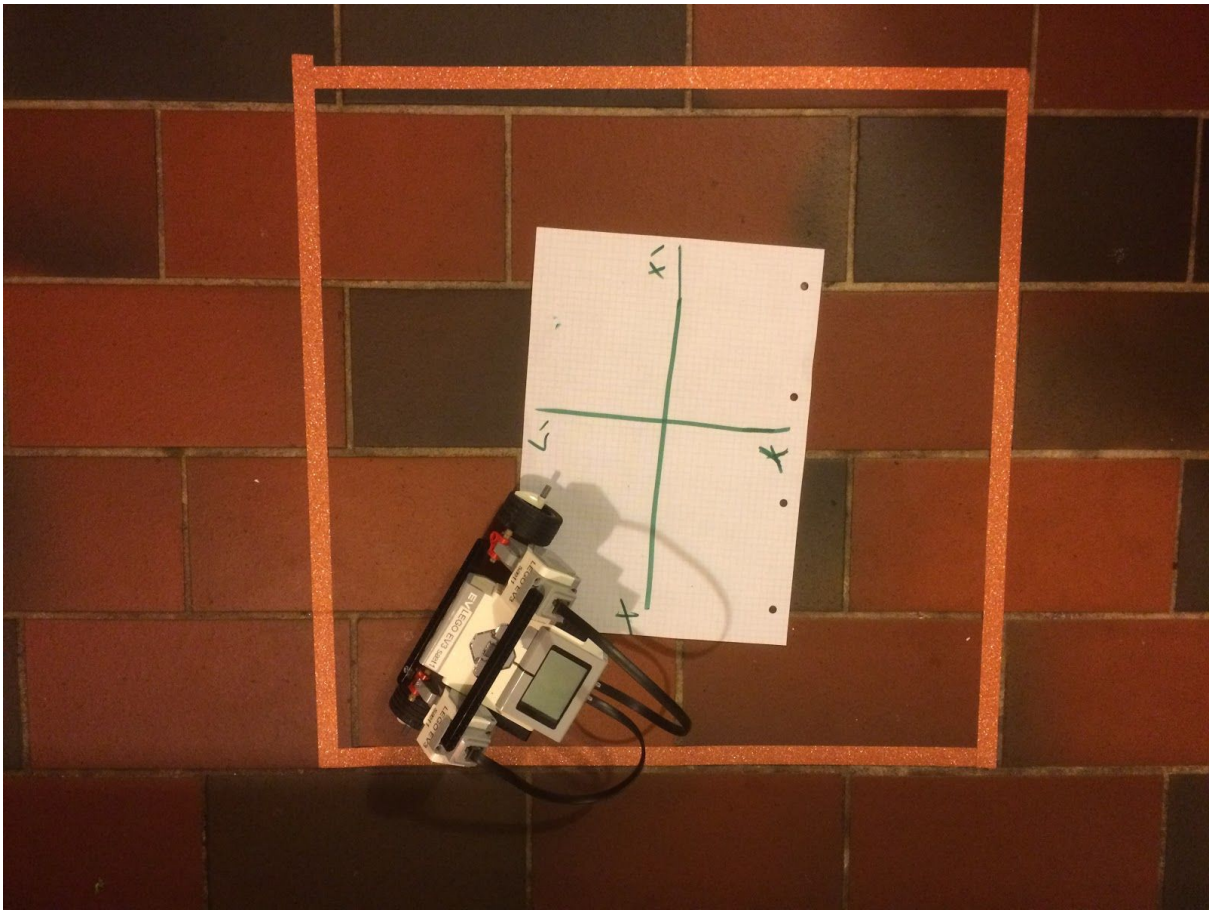
```
public static void Navigator_Class() {

    Wheel wheelL = WheeledChassis.modelWheel(Motor.B, 43.2).offset(-80);
    Wheel wheelR = WheeledChassis.modelWheel(Motor.C, 43.2).offset(80);

    Chassis chassis = new WheeledChassis(new Wheel[] { wheelL, wheelR }, WheeledChassis.TYPE_DIFFERENTIAL);

    MovePilot pilot = new MovePilot(chassis);
    pilot.setLinearSpeed(100);

    Navigator navigator = new Navigator(pilot);

    navigator.goTo(600, 0, -90);
    navigator.singleStep(true);
    navigator.followPath();
    navigator.waitForStop();

    navigator.goTo(600, -600, -90);
    navigator.singleStep(true);
    navigator.followPath();
    navigator.waitForStop();

    navigator.goTo(0, -600, 90);
    navigator.singleStep(true);
    navigator.followPath();
    navigator.waitForStop();

    navigator.goTo(0, 0);
    navigator.singleStep(true);
    navigator.followPath();
    navigator.waitForStop();
```

For writing the code I used the notes that were given regarding the assignments where the code snippet for the logic was given. The selected points can be seen above in the previous part of this document.

As an alternative I used the official documentation to write the code in another way.

```
/*navigator.addWaypoint(new Waypoint(650, 0));

navigator.singleStep(true);
navigator.followPath();
navigator.waitForStop();

navigator.addWaypoint(new Waypoint(700, -650));

navigator.singleStep(true);
navigator.followPath();
navigator.waitForStop();

navigator.addWaypoint(new Waypoint(0, -650));

navigator.singleStep(true);
navigator.followPath();
navigator.waitForStop();

navigator.addWaypoint(new Waypoint(0, 0));

navigator.singleStep(true);
navigator.followPath();
navigator.waitForStop();*/
```

In this code I used the Waypoint() where I added the selected the points that should be followed.



In several occasions during the test drives it was necessary to adjust the parameters and the code as well.

The code snippets in the book to demonstrate topics such as basic movement, navigation etc. were outdated due to the fact that the book was printed in 2004. The official documentation provided a more up-to-date version of the code.