MAP DRAWER ROBOT

4TH SEMESTER

Melissa Ma

Adi Tanase

ERHVERVS AKADEMI SYDVEST

Table of Contents

Abstract

This report is for the fourth semester exam in Computer Science. Here we explain the whole process that stands behind developing an autonomous robot. The project is created with the purpose to proof methodologies within robotics.

Introduction

Problem Statement

Project Background

In order to develop the robot we are orientating on using the nxt brick provided by the Lego Mindstorms and theirs additional parts. To get it running we are going to develop in java using the Eclipse IDE and the necessary plugins.

Project Definition

At the moment the robot is just a bunch of ideas with spare parts ready to be assembled. The way we want to put these ideas is to have the robot travelling and monitoring obstacles. The results should be reported to a PC, and with the information received, a map shall be created.

Vision of the solution

Vision Statement

Creating a product which will behave as an autonomous robot within a basic environment.

List of features

- Ability to explore
- Detection of objects

- Inspect and retrieve itself to a free area
- Able to be switch off at any time

Project Planning

The project is developed in a group. This has two members, which will collaborate in order to achieve the desired product.

We decided to split the project into two main parts.

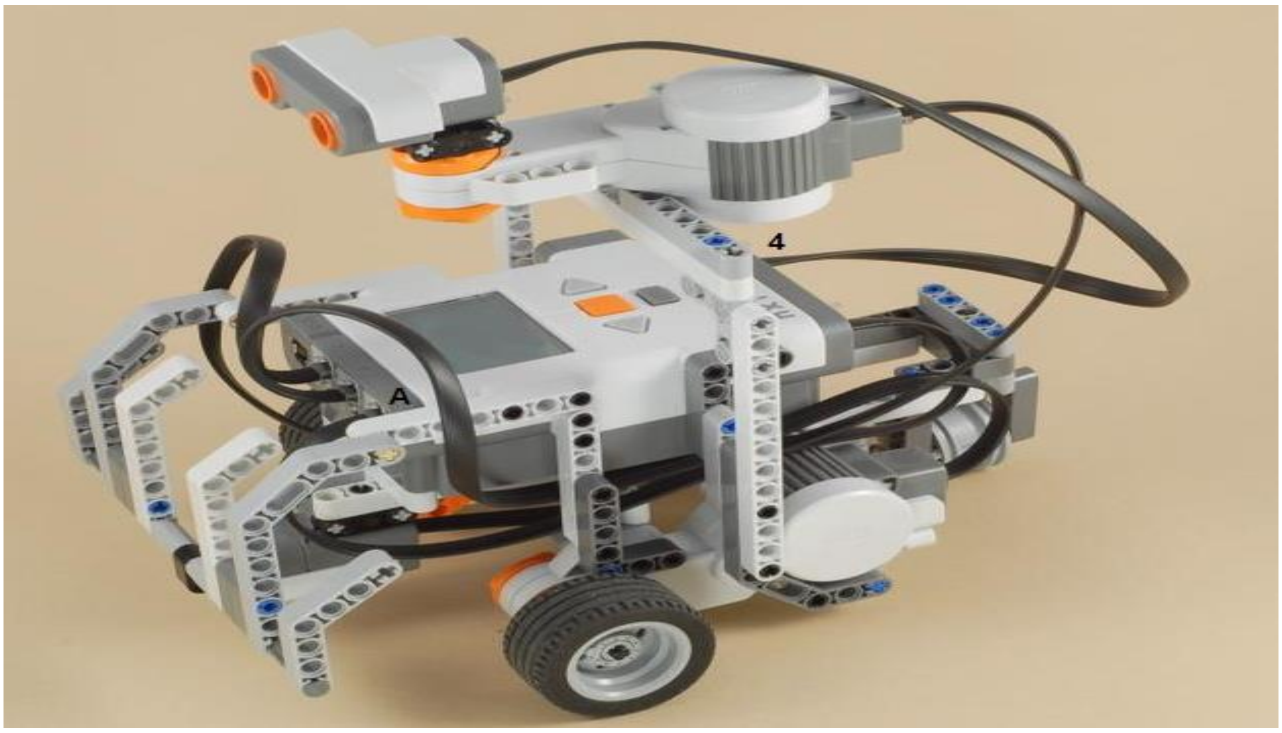
- **Building the robot**
  We had to prioritize this part in order to see how our code will affect the robot.

- **Developing the code**
  This part is also the longest one, it had been split it into few other tasks and spread them among us the team.

We worked at school and remotely from home and sharing the robot during the entire process.

Design

Construction of the Map Drawer robot is inspired from the Lego Mindstorms Projects. The one we decided to work with is the Explorer. We choose this structure because is fitting our requirements, meaning that it has the controllers for retrieving the particularly data we want. The picture below is illustrating our robot.



Design patterns

The final design pattern for our explorer robot it has been built by following two other projects. We started with the Castor design which is a blind robot an upgraded to Bumper Car.

The Bumper car is a bit more evolve meaning that it has a touch sensor in front so is not totally hopeless but that will not be enough in our case.

Next step, upgrade to the Explorer which is also our current final construction. It might seem a bit off a process but that engaged us to develop a better understanding of the structure.

Sensors

Map Drawer contains these sensors:

- Touch sensor : this will give a sense of a touch , it detects something when is pressed
- Ultrasonic Sensor: this will actually give vision and will enable later one to avoid obstacles and to determine their distances.
- Server Motor / rotation Sensor: this will enable to control the robot with precise rotations.

Architecture

We use the following construction for our map-drawer. We want to make our robot always go forward. When it hit any object, it will back up, then turn right and out put the position on the screen, also write the position into file. We can press escape button if we don't want our robot to continue work.

There are three behavior in our case, stored in an array of behavior. Inserted in this order ExploreBehavior , ObjectDetectBehavior and ExitBehavior. We named it as b1,b2,b3 . We pass this array as a parameter for Arbitrator object. After we call the start method of Arbitrator, the prioritized behavior will be active.

What should be the order of priority in our case?

The b3 is the highest priority, and b1 is the lowest one. For example, when our robot is start executing , the Arbitrator calls the takeControl() method in each behavior object, it will works way down through the array , till it finds a behavior that wants to take control. From b3 to b1, at the beginning, only the takeControl() method of b1 returns true , so the robot can move forward , until the next behavior is active.

Implementation

Here we are going to dissect our code, we will explain the process behind together with code examples. The coding part is divided l in two main parts, as following:

Client side

At this level we are having a LeJos NXT Project which is an option to do when the nxt plugins are installed within the Eclipse IDE.

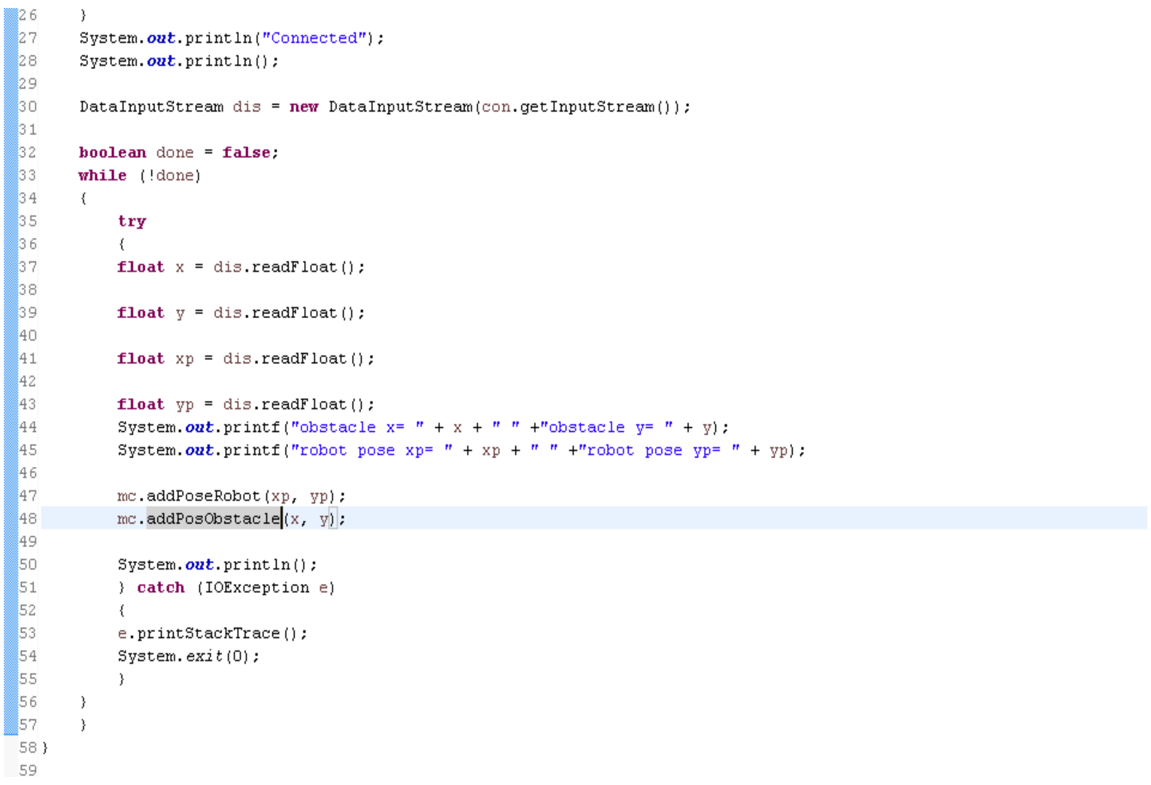
The project itself has two Java Classes which are called as the following:

- **ExplorerPCClient**

  This class is using the JFrame to create a 400*400 map after this we are establishing Bluetooth connection with the Server(robot).

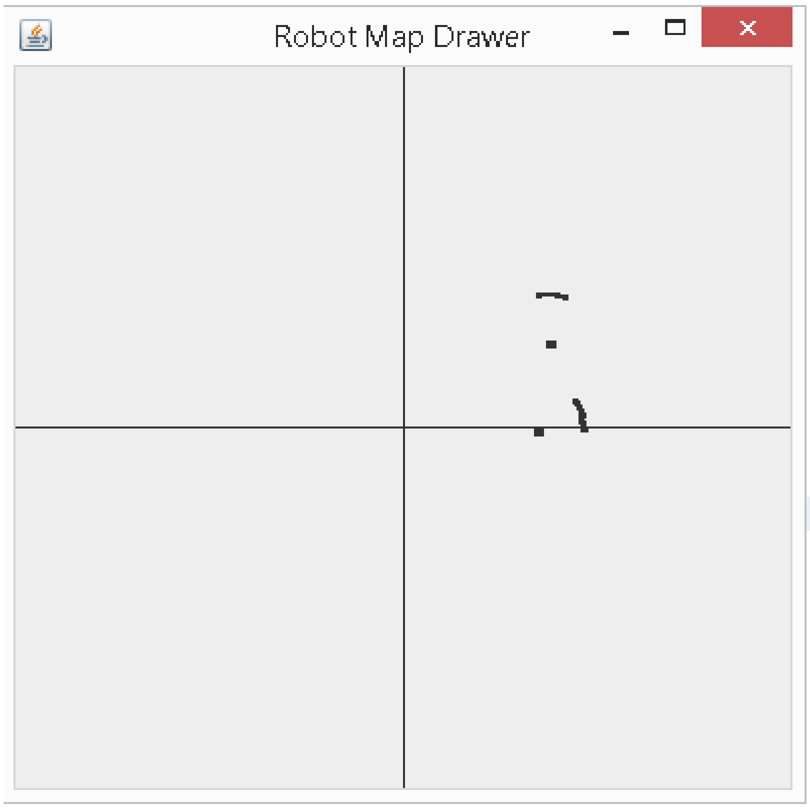
  By creating an instance of the DataInputStreamining class we are able to retrieve the coordinates for the robot and the obstacle.

  The code below is a sample of our class, the way we read the coordinates are in the same order flushed by the Server (robot), first the obstacle position and after the robot.



```java
    }
    System.out.println("Connected");
    System.out.println();

    DataInputStream dis = new DataInputStream(con.getInputStream());

    boolean done = false;
    while (!done)
    {
        try
        {
        float x = dis.readFloat();

        float y = dis.readFloat();

        float xp = dis.readFloat();

        float yp = dis.readFloat();
        System.out.printf("obstacle x= " + x + " " +"obstacle y= " + y);
        System.out.printf("robot pose xp= " + xp + " " +"robot pose yp= " + yp);

        mc.addPoseRobot(xp, yp);
        mc.addPosObstacle(x, y);

        System.out.println();
        } catch (IOException e)
        {
        e.printStackTrace();
        System.exit(0);
        }
    }
    }
}

```

The reason that we decided to get the coordinates for both items (robot, obstacle) is that we found it a bit tricky to see if correct are related to each other. In this manner we got a better visual of where the robot is and if the draws performed for the obstacle fits with the range we determine.
The figure below is illustrating how the drawing s looks like in the X, Y axe ,after the robot had met two obstacles.   The bigger dot is the robot position and the line is a partial shape of the obstacle encountered.



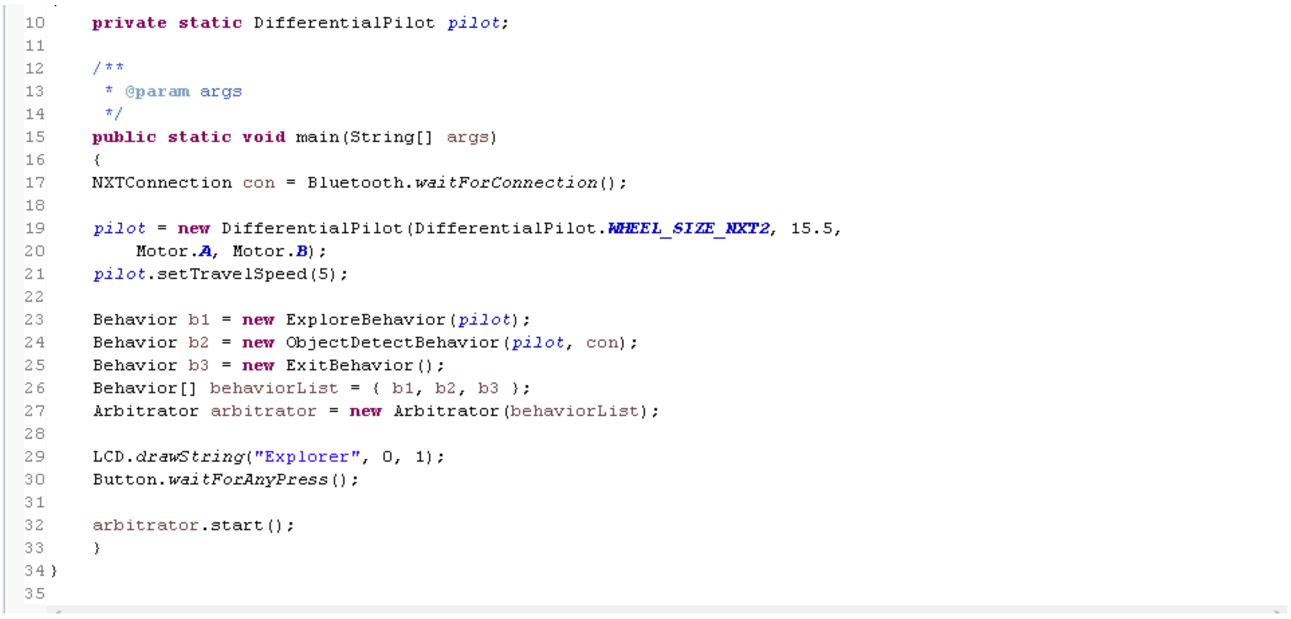
- **MapComponent**

  Here we have two Array List to store positions in coordinates for robot and another one for obstacles. We are iterating until we hit the array size and draw points. Here we faced some problems, we were using the foreach loop with it led to concurrency. Solution was to use the classic for loop which immediately mitigated the error.

Server side

In this part we invested more time than for the client side, that's because the behaviour architecture. Here we have to packages one for the main class and another one with three classes, each of them defining a behaviour. We are using the Differential Pilot to integrate the both motors, this will simplify the way we control the robot. The pilot instance is passed into the constructor of two of the classes a convenient way to give the behaviours control over the robot when occurs their turn. We are adding these three classes (behaviours) into a List called behaviourList and passed them within the Arbitrator constructor. Here the Exit Behaviour is store last, having the highest index, the Arbitrator Class will interpret this as a first priority.

The picture below is figuring the class we just described.



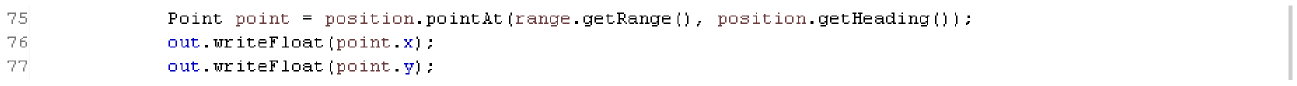
```
    private static DifferentialPilot pilot;

    /**
     * @param args
     */
    public static void main(String[] args)
    {
    NXTConnection con = Bluetooth.waitForConnection();

    pilot = new DifferentialPilot(DifferentialPilot.WHEEL_SIZE_NXT2, 15.5,
        Motor.A, Motor.B);
    pilot.setTravelSpeed(5);

    Behavior b1 = new ExploreBehavior(pilot);
    Behavior b2 = new ObjectDetectBehavior(pilot, con);
    Behavior b3 = new ExitBehavior();
    Behavior[] behaviorList = { b1, b2, b3 };
    Arbitrator arbitrator = new Arbitrator(behaviorList);

    LCD.drawString("Explorer", 0, 1);
    Button.waitForAnyPress();

    arbitrator.start();
    }
}
```

From the three Behaviours we nominated the ObjectDetector class to explain the process that is executed when is facing an obstacle. As an introduction we will mention that the Explorer behaviour is returning the value 'true' in the takeControl method that result in trying always to get the control to explore.

Well having the robot always curious at any time is great but if an obstacle occurs within the range 25 cm it will supress the Explorer Behavior and start flushing data.

We are using the OdometryPoseProvider to determine the position of our robot and their obstacles and to obtain the heading. For determine is we face some obstacle we are using the FeatureListener to get a feedback .

In the code below is presented a piece of the class ObjectDetector , we elected it because it has the most challenge full. Before having the knowledge of the features provided by the Class Pose, first we struggled on obtaining the coordinates for an obstacles with deep math calculations. The way we were trying to attack was to use the Pythagoras theorem assuming that we know the know the range. Second solution was to use the slope and to determine coordinates by assuming that the robot position is a point on the slope and the obstacle position is another point on it. A lot of calculations without an accurate result. Like always getting useful feedback, we approach our teacher and it led to a solution that is pictured below.



```
        Point point = position.pointAt(range.getRange(), position.getHeading());
        out.writeFloat(point.x);
        out.writeFloat(point.y);
```

Conclusion

It was a great experience to work with robots, we had the opportunity to observe how the code is behaving within the realistic environment. We would say that we were impressed on the adjustment that had to done in order to get the nxt brick up running with the expected result. That led us to increment our already java skills to a different level, where we have to understand and create code for a moving hardware the NXT brick. Lego Mindstorms comes with a lot of design patterns and explanatory documentation on how to develop a robot that we desire which it was a great feedback during this project. By far the most attractive concept about developing   software is the Behaviour based architecture. We've been fascinating on how easy becomes the code and later one to implement another behaviour act without getting all errors in world figure speaking. Great plus to avoid redundancy and a big load of if else statements which it may lead to have a spaghetti code. . As always there is place to improve but within the time range we had to document the work we've done. We are satisfy that the Map Drawing is design to evolve so ideas can be integrated.

Overall we are satisfy with the knowledge we achieved during developing this project, the way we see it, code has this tendency to be part of our daily lives more and more.