Robot Programming Project Course

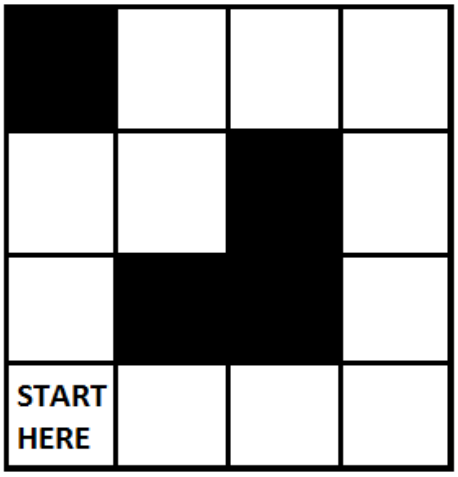
Mapping Robot

Fiyinfoluwa Soyoye
014528002
Fiyinfoluwa.soyoye@helsinki.fi

Mapping Robot Description

The mapping robot is a robot which can explore a grid-like environment and create a map of roads in this environment. It then displays this map on the screen of the robot. The maze is first arranged by the robot operator into any arrangement as long as the arrangement defines a continuous path through the grid.

An example of such an arrangement is below: the black squares represent places blocked off and the white squares are the path that the robot can follow and map.



The robot then explores the maze for about 5 minutes and marks off in its memory the path way. When the escape button is pressed, it stops and prints the map on the screen.

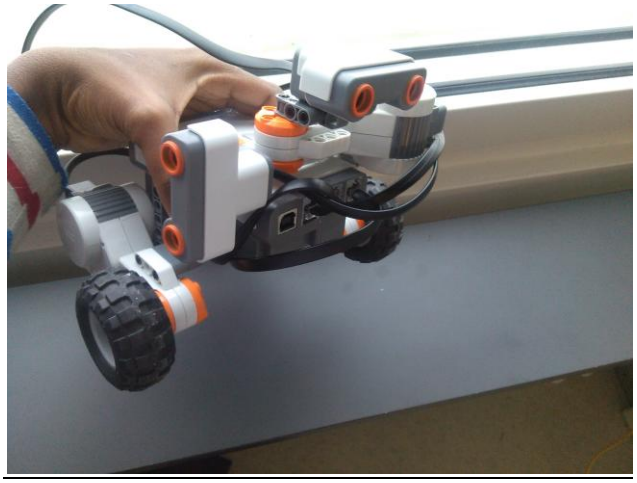
Robot build

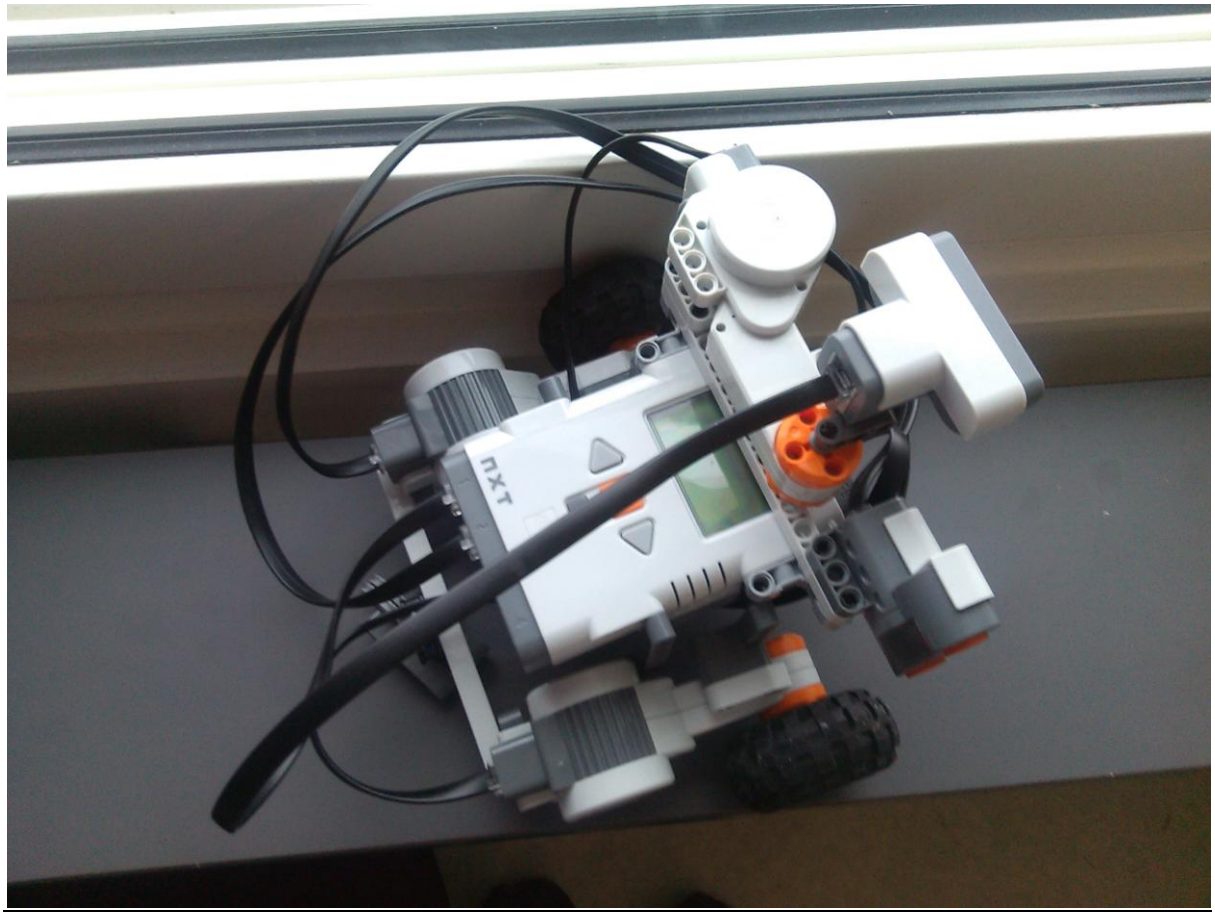
The robot is a simple car with two motors driving the two front tires. The 3rd tire at the back is a caster wheel that is attached to the back of the robot. There is a third motor attached sideways to the top of the robot. On it is attached an ultrasonic sensor which is set to rotate 180 degrees. There is another ultrasonic sensor facing the right hand side of the robot.

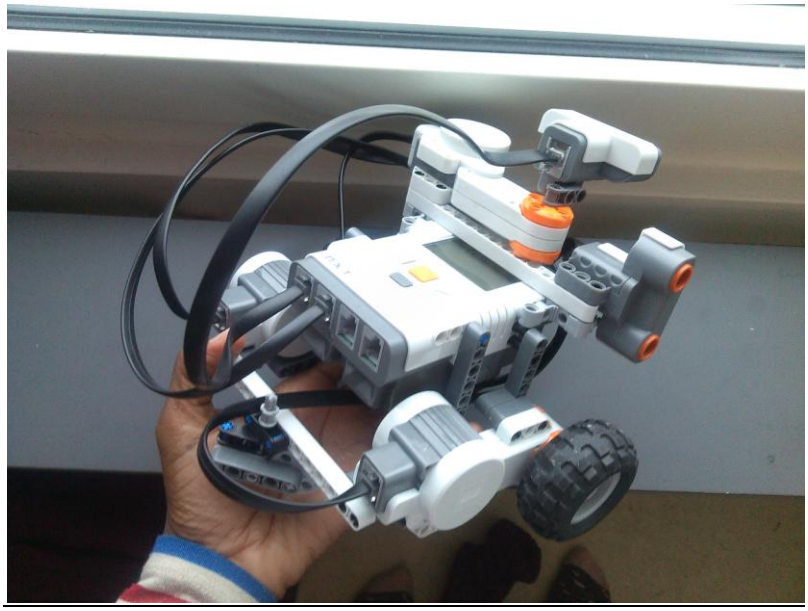
Wheel diameter: 5.6cm

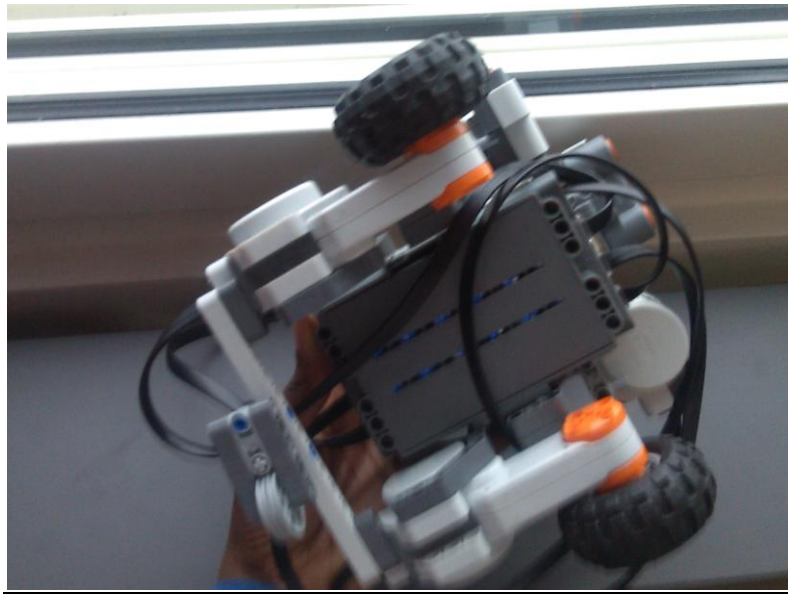
Track diameter: 16.5cm

Pictures




NXT





Code Structure

There are five classes in this code:

The HelloWorld class only holds the main method.

The Map class handles all the mapping functionality of the robot including printing the map on the screen after the robot runs.

The MoveDecider class decides what next move the robot should take depending on the values from the Ultrasonic sensors and the limits of the maze.

The Positions class is there to organize all the X and Y coordinates and make the code easier to write and understand.

The Robot class handles all the properties of the robot including its current position, its next position, its sensors and its actual movements.

How the code works: The robot has a 50 by 50 array initialized to 0 to help with the mapping. When the robot starts, it looks towards the left, right and forward positions and then

stores the values it gets from the ultrasonic sensors at those positions. It checks the values to determine what directions are free and each free grid is incremented by 1 on the array. The blocked directions are decremented. The robot then randomly picks a direction to go from the list of free directions and moves there. Then it repeats the process until the escape button is held down. After it stops, the printMap() method is called and the map is printed on the screen of the LCD.

Tests

Test case 1: Distance from each direction

This was to make sure that the robot ultrasonic sensors recognized when walls are present and when they are not. The robot was placed in an enclosure and the program to scan the forward, left and right positions was run. If there was a wall present at the forward position, a specific type of noise is emitted. If there was no wall, another type of beep was emitted. And so on for the other two positions.
The test case ran perfectly.

Test case 2: Choosing where to go

This test case is an extension of the first test case. After it was determined that the robot could recognize walls, there was a test to make sure that it could pick a direction in which there was no a wall and then turn towards it. The code for this is such that if there was more than one direction free, the robot would randomly pick one and turn. If there was just one then it turned towards it and if there were none, it would reverse.
Problems: there was an exception for when there was no free direction to turn to because the robot would try to randomly generate a direction out of no possible directions.
Solution: a simple if-else structure in the code solved this.

Test case 3: Does the robot know where it is?

In this test case, the robot was allowed to run along a maze arrangement for a few turns and when a button was pressed, it should stop and print the specific grid location it was on at the moment.
Problems: At this point of the programming, it was only possible to start the robot from the possible (0, 0) in the maze. It turned out that I forgot to prevent the robot from being able to turn and move towards the location (-1, 0). This generated an exception immediately with the array in the code.
Solution: structures in the code to make sure that the robot can never go out of bounds of 0 to 50 in both the x and the y axes. After this, it ran properly

Test case 4: What are the best grid dimensions for the robot to run?

Since the robot was already set to go 40cm with each move, it was obvious that that would be the ideal size each grid so that the robot could always be in the middle of the grids. The program was also set to accommodate the robot running on a maximum of 50 by 50 grids. It turns out however that the LCD screen will only display up to 16 by 8 grids so that is the actual maximum.

Test case 5: Did the robot work? On the first try of the entire program, it did pretty well until the end when it got stuck moving back and forth between only two grids. To fix this I create a new method checkIfAgain() in the MoveDecider class which would check the value of every possible

position around the robot and then eliminate the positions that had been visited already a lot of times. That way, the chances of visiting a grid position that had never been previously visited was greatly increased. This fixed the problem. The robot ran and the map was printed successfully.

Limitations and Future:

Robot turns are not perfect. One wrong turn leads to a really big angle deviations and so a future improvement is a line follower mechanism. Also in the future the robot should be able to print the map to a file. One of the goals of this project was to be able to give the robot a location to go to and then it goes there without having to navigate with sensors but only with the map. There was no time to implement this so this can be done in the future.

Operating Instructions:

1. Set up your maze of not more than 16 by 8 grids. A lot less would be ideal though: about 4 by 3. It can be made of boxes, books and other solid objects. The ideal size of each grid is 40cm by 40cm.
2. In the main method inside the HelloWorld class of code, enter where specified the position that you plan start your robot in. You can also enter tire and track diameters if there are from 5.6cm and 16.5cm respectively.
3. Place your robot in the middle of the chosen start grid and run the program downloaded. Leave to run until you are satisfied and then hold down the escape button until it stops.
4. Then look at the map on the LCD.