NXT MapBot

Dan Lapp
COSC 3P78 Project

April 21, 2014

Contents

1 Instructions

1.1 Compiling Sources

The source files have been included in the electronic submission of the code. The main class is MapBot.java. I have included batch scripts for compiling and uploading the code to the NXT. All required classes should automatically be compiled and uploaded just by running the scripts on MapBot.java. To compile the sources and upload to a brick, open a command interface and navigate to the source folder. Enter *build MapBot* to compile. You can then upload the files to the brick by inputting *upload MapBot* while the NXJ is plugged in and turned on.

This should upload all required files, but all of the java classes included in the submission are required by the NXT to run the MapBot program. If for some reason this does not compile and upload all of them, you can repeat the above commands for each file.

1.2 Running MapBot

To start the program, run MapBot.nxj. The robot will wait for you to prepare before it starts. Pressing ENTER will start mapping and ESCAPE will quit the program.

The MapBot program assumes that everything is square, based on the fact that it only uses a single ultrasonic sensor for navigation. Place the robot on the floor perpendicular to a wall. The robot tries to maintain it's relative positioning based on moving up, down, left and right, keeping track of how far it has gone. Therefore you will get a better mapping when the robot is directly facing a wall. When you are ready to start, press ENTER, and the robot will begin mapping the area.

Press ESCAPE at any time during mapping to halt the program and return to the menu. When you are done mapping, press ENTER. The robot will finish it's current iteration in the mapping loop, and you will be presented with a map on the screen.

The map is displayed relative to the robot's starting position. Up on the screen is the direction the robot was first facing. You can navigate through the other screens by pressing LEFT, RIGHT, ENTER to go up, and ESCAPE to go down.

To quit the program and return to the menu, press ENTER and ESCAPE together.

2 Autonomous Mapping

2.1 Abstract

The problem of autonomous mapping deals with a robot autonomously navigating a space, while maintaining position, and mapping obstacles and traversable terrain. Using positioning of obstacles or recognizing landmarks are methods often used to keep the robot aware of its surroundings and its position. Intelligent mapping requires several computing disciplines to work together. Computer vision and pattern recognition, navigation and graphing, and machine learning, to name a few.

In a nutshell, an autonomous mapping robot needs to navigate a space, and keep track of its direction and position according to some scale. It needs to identify obstacles and differentiate them from navigable space. Obstacles are then drawn to the map keeping with the scale and location of the object in the real world. This produces a properly scaled map that the robot can use to navigate the space, and plan paths to and from arbitrary areas.

2.2 Autonomous Mapping

There are three integral parts to even the simplest of map making automatons. Navigation, localization, and mapping are the three essential parts to think about. These work together to provide the robot with the tools it requires to create a map of its environment. This problem is commonly referred to as Simultaneous Location and Mapping (SLAM).

2.3 Navigation

Navigation and path planning are difficult to implement on a robot consistently. One of the biggest problems is real world environments are not consistent. It is incredibly hard to drop a robot into an arbitrary real world area and have it navigate effectively. This is something humans can, for the most part, do effortlessly, due to our rational thinking and observatory skills. The robot needs to make sure it can find its goal from any position for it to be effective at navigation. It needs to have an efficient means of searching the environment as well, whether it has a map or not.

A map making robot will have to have one or more path planning strategies that are affected by traversable terrain and impassable obstacles. Heuristics can help the robot make informed decisions about where it wants to go, such as moving to the most open space, or moving toward the nearest wall that it hasnt seen before. This allows the robot to move throughout the area and not collide with obstacles. The robot will need to keep track of which direction it is going, and how far it has gone from the starting point for localization and mapping purposes.

2.4 Localization

Localization has to do with identifying where in the space the robot is. There are several methods commonly used in map making robots, including coordinate mapping, GPS, magnetometer, and landmark. Each of these methods has their own advantages and disadvantages, and will depend on the type and scale of your mapping project. For example, GPS localization would be much better for something like a Mars rover than something that will map your kitchen.

Coordinate mapping keeps track of the distance the robot has travelled from a given point, often the starting point. The robot marks the coordinates of obstacles and builds a map of coordinates that outlines the space. This is a good method for smaller scale mapping bots, but it can be inaccurate. The robot only knows how far the motors have moved. So if it gets stuck, or slips, or its direction is affected in any way, that will throw off the mapping. As distance increases, these errors will accumulate.

GPS on the other hand is able to tell if the robot gets stuck, or changes direction unexpectedly. This is because it gives a coordinate based on the physical location of the robot, not based on how far it has moved. GPS can be inaccurate, however and unreliable in some environments. It is also generally unavailable indoors. Also, the scope of the robot project might not allow for it.

Magnetometer sensors use the earths magnetic fields to let a robot know which direction it is heading, in degrees from due north. These, like GPS, dont rely on the supposed direction of the robot based on how far it has moved. These will only be able to tell you what direction the robot is heading in, so they will need to use other methods to calculate where the robot actually is. Many buildings can also cause significant interference with magnetometer sensors, causing wildly inaccurate readings.

Landmark detection is arguably the most reliable method of localization. It relies on the robot recognising where it is in relation to a distinct object in the world. This way it can know where it is in the environment without having to rely on sensors or motors that are prone to drift. This is by far the hardest of the methods mentioned to implement. It requires some knowledge of the world, and knowledge of the landmark and where it is. The robot will need to have a certain intelligence about it to recognise and react to landmarks when it sees them.

2.5 Mapping

The representation of the map is important for navigation and localization. If the robot needs to build a map for navigation, or to figure out where it is in the world, then this is very important. This is the notion of spatial memory. There are two forms of spatial memory, qualitative and metric.

Qualitative connects certain areas or landmarks known to the robot together. The map forms a sort of graph with weighted distances that the robot can use to traverse the world. Metric is more like a traditional map, where obstacles and traversable spaces are represented by mapping their actual layout in a two dimensional space. The result is a birds eye map of the world.

3 MapBot

MapBot uses a simple room navigation and mapping technique to map obstacles and distinguish them from open floor space. It locates an object or a wall and follows it, recording the location on the map. When it detects it is in a cycle it will move on to find another object. It maps only in four directions, based on its starting position: up, down, left and right. It uses an ultrasonic rangefinder to determine how far away the robot is from a given object.

3.1 Navigation

I made the robot as a compact tracked robot. This gives the robot the ability to turn 90 degrees on the spot, to make the wall following algorithm fairly straightforward. I initially wanted to have two ultrasonic rangefinders on the robot, one on the front and another on the side to follow along with the wall. After trying to poll each one individually, I found that there was quite a bit of interference and the robot was detecting non-existent objects and turning sporadically. Simply removing the second sensor proved that it was interference from using two sensors at once. Because of this, to follow a wall, MapBot moves forward alongside a wall, and will turn periodically to see if it is still there. If so, it turns back and continues.

3.2 Localization

MapBot uses simple coordinate localization. The location of the robot and all detected objects are represented in the world as points of collision. The robot itself has a point, which moves as the motors move. One pixel on the screen is represented by what I called a tick in the program. For each tick the robot moves, they advance one pixel, and one map coordinate in the localization system. The x and y values of the robot are adjusted accordingly as the robot moves. The location of objects in the world are represented as a list of points, and their distance from the robot are easily calculated.

3.3 Mapping

MapBot uses the NXT screen to display the map of the area after the search is stopped. The map can span multiple screen widths and heights, and the user can navigate through them. The map is represented by a series of Nodes. Each Node is a quadrupally linked list,

with references to the next screen up, down, left and right. The data for a current Node is a list of points. When the map is drawn, the points are connected together with line segments.

As the robot moves through the world, it has a world coordinate and a current node coordinate. So when the robot moves off of one node to the next, the world coordinate is adjusted normally, and the current node coordinate is set to the robots position on the new node. This allows us to draw obstacles on the current node, and still maintain the robots absolute position in the world.

After the mapping routine has been stopped, the user can see the map. It shows the current screen where the robot ended its search. The user can use the NXT buttons to navigate up, down, left and right from that position to view the rest of the map.

4 Problems

MapBot navigates areas quite well, and almost never collides with a wall or object. It maps areas best that have square corners. It will still map a room with an odd angled wall, or round objects, however they will be jagged as a result of only having one sensor. I couldnt make it follow a wall on say, a 45° angle and have it show an angled line. It therefore assumes everything has square corners. Also because of the single sensor, it made it hard to detect very small objects, as it would require the robot to turn and look constantly.

I tested the code for making 90 degree turns quite extensively, and seemed to get an optimal number. After letting MapBot go for a while though, it can tend to veer a little bit off, which affects the mapping. This was worse on my tile floor, and way worse on carpet. The tank tracks arent the most accurate, but to try and stay straight on carpet or an uneven tile floor requires a bit of a nudge from time to time.