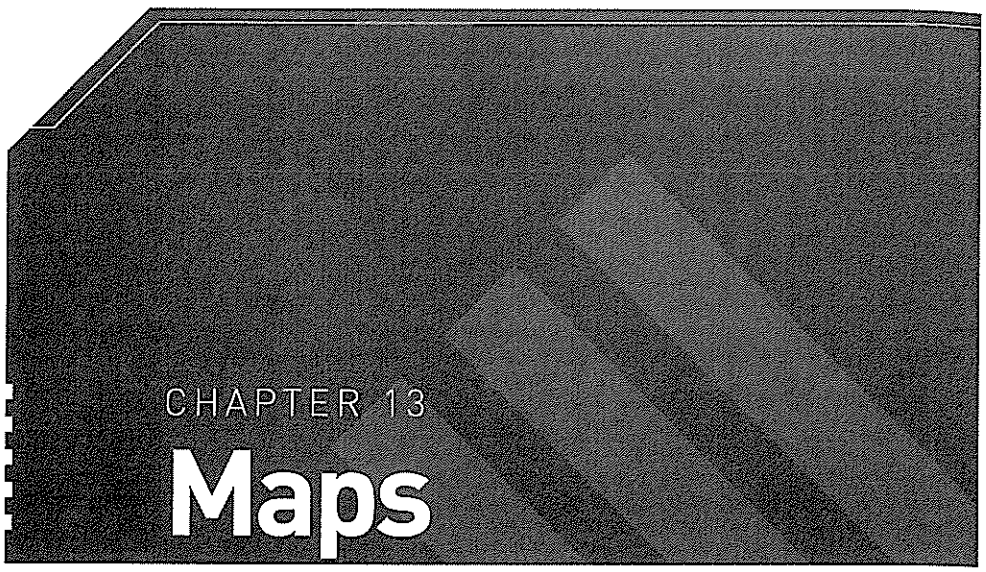


CHAPTER 13

Maps

TOPICS IN THIS CHAPTER

- Maps
- Editing a map
- Controlling a robot using a map
- Whole home mapping

A map is a symbolic representation of an environment. It shows the relationship between objects on the map. The goal of this chapter is to create an accurate map of a test area, or perhaps even a large portion of your home. Once this map is complete, you can use the map to help guide your robot to specific locations. But first, let's look at how leJOS NXJ represents a map and the flexibility it provides in navigation projects.

Maps

As we learned earlier in the book, maps are an important tool for navigation (see Figure 13-1). They are a record of previous exploration, and can help you get to where you want to go. Maps are an important component of human navigation, such as with ships in the Navy, and they are important for navigational robots.

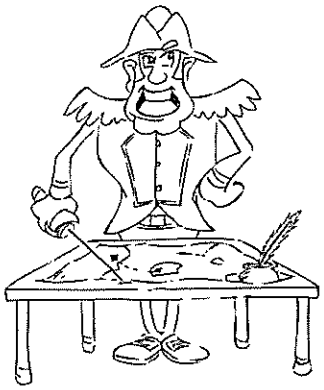


Figure 13-1: Using map data to navigate

Why create a map of the environment for your robot? There are two very good reasons, both of which are well worth the effort on their own. The first is that it is difficult to tell your robot where you want it to go. You can't just point with your finger to a spot on the floor and command it to go there. Re-

where you wanted it to go using only coordinates? To get it to move to a specific point required taking out a ruler and measuring centimeters along the x and y coordinates. By viewing the map on a computer screen, you can very easily command the robot by clicking on different locations.

The second reason for making a map is that your robot can store the map in its memory and use the map geometry to plan a series of moves to a new location. The next chapter will demonstrate this concept in more detail. Even later, in chapter 26, your robot will use a map to find its location within an environment using a technique called Monte Carlo Localization. But for this chapter, we will deal with the first concept of telling a robot where to go on a map.

Autonomous navigation is often more complex than it first appears. It is one thing to program a robot to figure out how to physically get from one location to another, but sometimes the robot needs to navigate only where it is socially acceptable. For instance, the shortest route for a vehicle might be to cut through private property, but it is not socially acceptable (or legal) to do so. Maps can help to determine socially acceptable routes of navigation, not just physically possible ones.

The map classes in the leJOS API are mainly indoor specific. Outdoors, you'll find that detecting objects becomes difficult because terrain is uneven. It is a challenge for sensors to distinguish an impassable object from one that the robot can navigate over. Indoors is much simpler because the floor is a flat plane and all the walls are perpendicular to the floor, providing clear and unambiguous targets for the ultrasonic sensor to detect.

There are several ways a cartographer (ether amateur or professional) can choose to map an environment. We could use an occupancy grid map, which consists of a regular grid of squares (see Figure 13-2). But a simpler way to create a map is with a set of lines that represent walls. Most rooms have straight-line walls, so the leJOS API does not allow curved walls. You can still use diagonal lines for walls

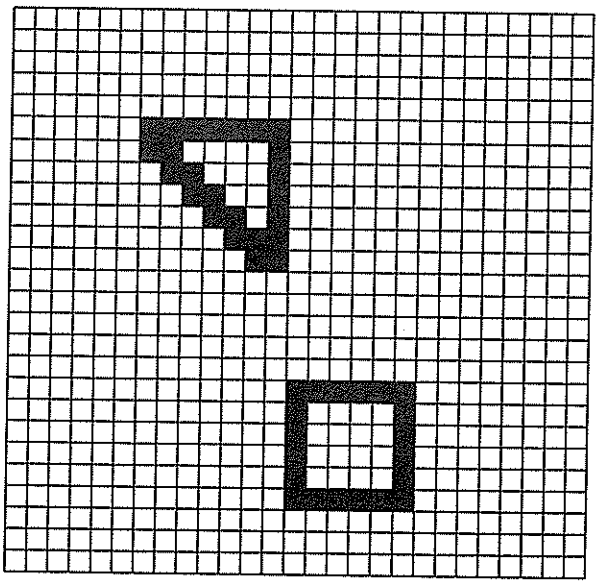


Figure 13-2: Viewing an occupancy grip map

The mapping API is located in lejos.robotics.mapping. Data is stored in a class called LineMap, and it simply consists of an array of lines. Each line consists of two coordinates—x1, y1, x2, y2. But how do we generate map data?

Editing a Map

The leJOS developers wanted to give users the ability to design a map using a dedicated map editor. Rather than including a proprietary map editor with leJOS, we decided to support standard file types for representing map geometry. That way, you can use the latest available map editing tools to create, save and edit your own maps.

It can take a long time to measure a room and enter it into an editor. With a good editor, lines and polygons can be moved, rotated, and named. For example, the living room might have furniture in it, such as a couch. If you rearrange your living room, it should be easy to load the map file into

where you wanted it to go using only coordinates? To get it to move to a specific point required taking out a ruler and measuring centimeters along the x and y coordinates. By viewing the map on a computer screen, you can very easily command the robot by clicking on different locations.

The second reason for making a map is that your robot can store the map in its memory and use the map geometry to plan a series of moves to a new location. The next chapter will demonstrate this concept in more detail. Even later, in chapter 26, your robot will use a map to find its location within an environment using a technique called Monte Carlo Localization. But for this chapter, we will deal with the first concept of telling a robot where to go on a map.

Autonomous navigation is often more complex than it first appears. It is one thing to program a robot to figure out how to physically get from one location to another, but sometimes the robot needs to navigate only where it is socially acceptable. For instance, the shortest route for a vehicle might be to cut through private property, but it is not socially acceptable (or legal) to do so. Maps can help to determine socially acceptable routes of navigation, not just physically possible ones.

The map classes in the leJOS API are mainly indoor specific. Outdoors, you'll find that detecting objects becomes difficult because terrain is uneven. It is a challenge for sensors to distinguish an impassable object from one that the robot can navigate over. Indoors is much simpler because the floor is a flat plane and all the walls are perpendicular to the floor, providing clear and unambiguous targets for the ultrasonic sensor to detect.

There are several ways a cartographer (ether amateur or professional) can choose to map an environment. We could use an occupancy grid map, which consists of a regular grid of squares (see Figure 13-2). But a simpler way to create a map is with a set of lines that represent walls. Most rooms have straight-line walls, so the leJOS API does not allow curved walls. You can still use diagonal lines for walls

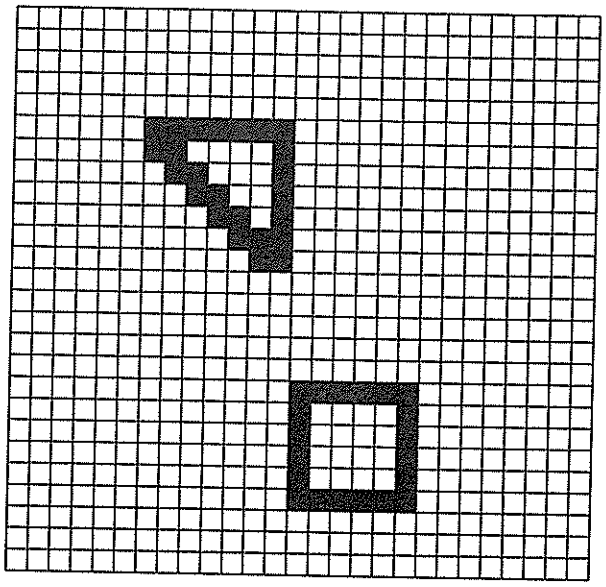


Figure 13-2: Viewing an occupancy grip map

The mapping API is located in lejos.robotics.mapping. Data is stored in a class called LineMap, and it simply consists of an array of lines. Each line consists of two coordinates—x1, y1, x2, y2. But how do we generate map data?

Editing a Map

The leJOS developers wanted to give users the ability to design a map using a dedicated map editor. Rather than including a proprietary map editor with leJOS, we decided to support standard file types for representing map geometry. That way, you can use the latest available map editing tools to create, save and edit your own maps.

It can take a long time to measure a room and enter it into an editor. With a good editor, lines and polygons can be moved, rotated, and named. For example, the living room might have furniture in it, such as a couch. If you rearrange your living room, it should be easy to load the map file into

As of this writing, leJOS NXJ supports a few file types. The most important file format is called SVG, which stands for *Scalable Vector Graphics*. This is a popular open source standard for creating line geometry such as maps. This file format uses XML code and there are many editors available. Let's get started.

Creating a Test Area

Before you can begin creating a map, you need to decide on an area to map. We'll start small, because you will have to measure each wall of your test area. Robots are in their infancy still, so we want a simple, uncluttered area where the robot has little chance of running into dangerous obstacles. Think of it as a playpen for robots.

You can use household objects to close off a suitable area in one of your rooms, or use an existing walled area if it is completely enclosed by straight walls and is free from irregular furniture and other objects. For this chapter, I used the rear office wall and set up wooden partitions and filing cabinets to enclose a test area (see Figure 13-3).

I also tried to make sure the area was not symmetrical because I wanted to use the same map file and test area for Monte Carlo Localization later in this book. Once you have your test area ready, find a measuring tape and begin measuring the dimensions, preferably in centimeters. Record the measurements on a simple diagram. Nothing too fancy is required (see Figure 13-4).

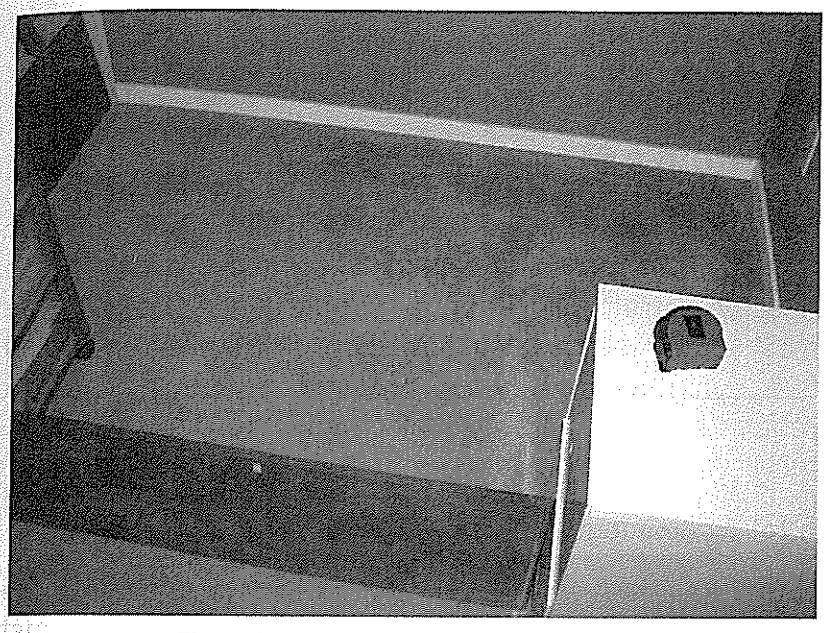
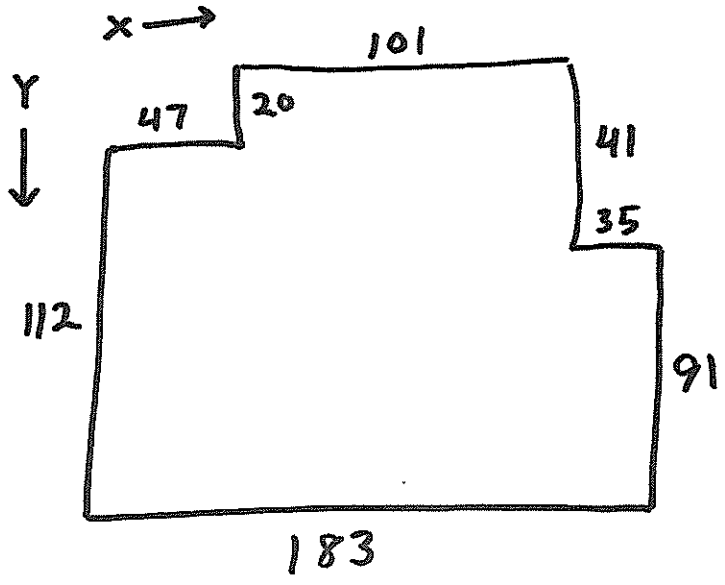


Figure 13-3: Creating an enclosed test area



Now that we have all the measurements, we can create our map.

Creating a Map

In this section we will create an SVG map using an editor. You could use any editor capable of using SVG files, such as Adobe Illustrator. However, for simplicity sake, we will use a dedicated web-based application hosted by Google called SVG-Edit. Because it is web-based, there is no installation required and it works on all platforms. You can find the editor at the following address.


Mac

http://code.google.com/p/svg-edit/

There are some excellent training videos available on the site, but they are largely overkill for our purposes. We just want to draw straight lines. To get started, go to the website and select the latest version (the screenshots in this chapter are from SVG-edit 2.6 running in a Firefox browser).

1. Click on the file menu, which looks something like a pencil drawing a flower (see Figure 13-5). Select Document Properties.
2. In the Document Properties window, enter a name for the map (see Figure 13-6). Enter the canvas dimensions to fit the size of your map area. The coordinate units in SVG are arbitrary, but we will consider them as centimeters. Uncheck the Snapping on/off option and click OK.
3. Select the line drawing tool and roughly draw one of the lines from your sketch. You do not need the exact line measurement when you sketch the line in because we will tweak the line dimensions. You can manually enter the proper coordinate values at the top [illegible] x1, y1, x2, y2 fields

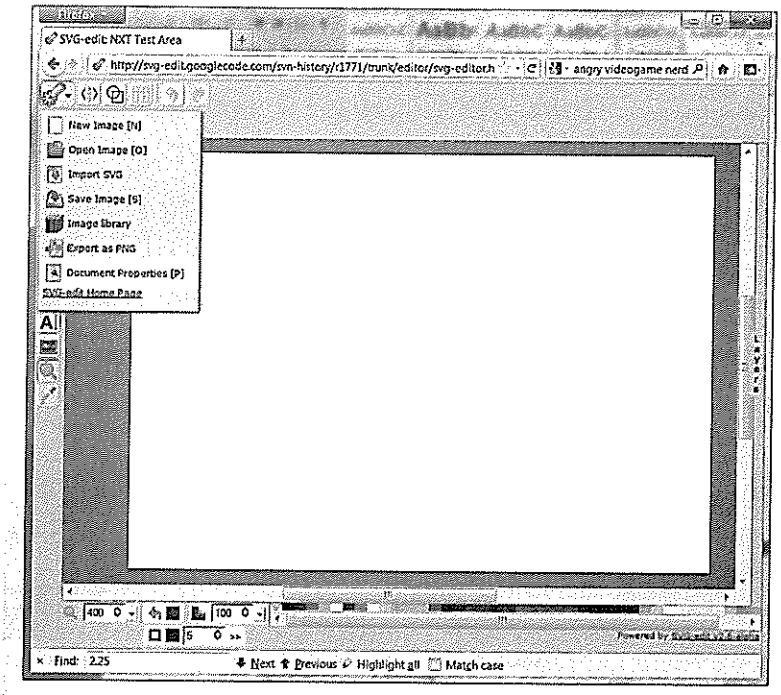
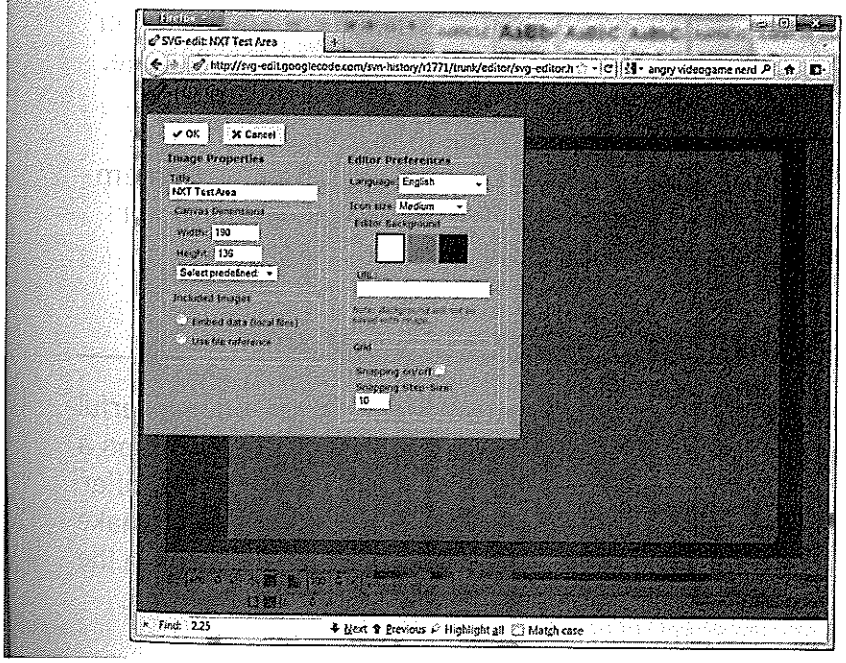


Figure 13-5: The file menu



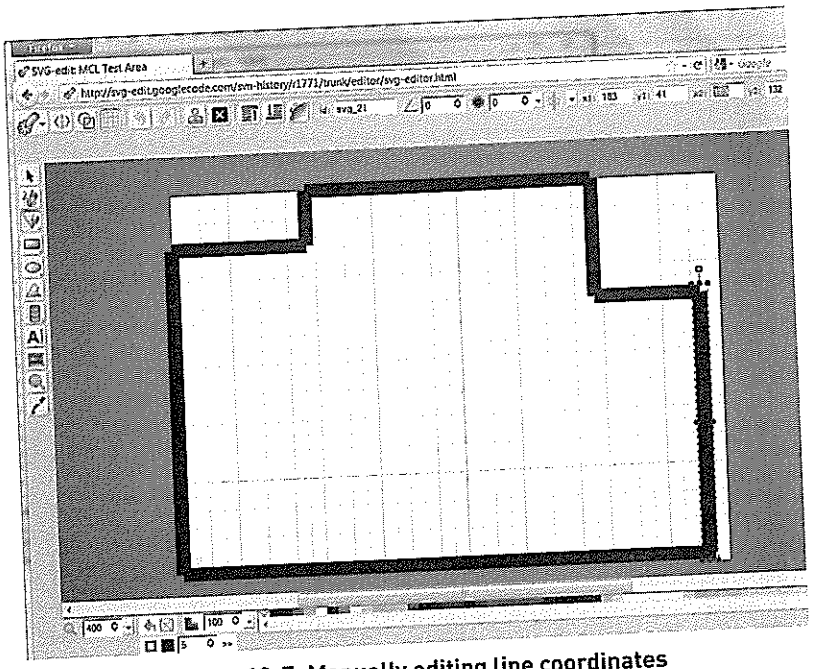


Figure 13-7: Manually editing line coordinates

4. Continue drawing all the lines of your map. When you are done, click the menu icon and select Save Image. In Firefox, a new window appears showing the map. You can either right click the map image and select Save Page As, or select File > Save Page As... Remember where you saved the file as we will copy it to our project folder later in this chapter.

Controlling a Robot using a Map

In this section we will create a very simple but powerful NXT application that will display real-time map data on a PC. You can make your robot travel, rotate and go to different coordinates by clicking on a point on your map to make the robot drive to that location. At all times you will see the present location of the robot and the path it has driven.

NXT Program

The NXT program is very simple. It creates the usual navigational classes, such as a pilot and a navigator. Once this is done, it waits for the PC to connect to it using the class NXTNavigationModel.

The NXTNavigationModel class allows you to send navigation commands to your robot, but also to synchronize data (such as pose) with the PC. The code below shows how to add a Pilot and Navigator to the NXTNavigationModel. Once these are added, the PC map application will visually update all the movements of your robot on the display.

```
import lejos.nxt.Motor;
import lejos.robotics.mapping.
NXTNavigationModel;
import lejos.robotics.navigation.*;

public class NXTSlave {
  public static void main(String[] args)
throws Exception {
    DifferentialPilot robot = new Differential
Pilot(5.6,16.4,Motor.B,Motor.C);
    robot.setAcceleration(500);
    Navigator navigator = new Navigator(robot);
    Pose start = new Pose(17, 97, 0);
    navigator.getPoseProvider().setPose(start);
    NXTNavigationModel model = new
NXTNavigationModel();
    model.addPilot(robot);
    model.addNavigator(navigator);

model.addPoseProvider(navigator.
getPoseProvider());
  }
}
```

This code uses a differential robot, such as the Carpet Rover (see Figure 3-3). Make sure to modify the code above with the proper tire size and track width for your robot. You will also need to decide the starting coordinates of your robot within your test area. You might want to place a coin at this location so you don't forget it for subsequent trials. Measure the coordinates of this location and then change the following line of code to include the x, y coordinates and heading.

```
Pose start = new Pose(17, 97, 0);
```

Upload and run the code. Place your robot in the test area at the starting pose you specified in the code. Now we need to run an application on the PC.

PC Program

The leJOS developers have included a comprehensive application for displaying map data and controlling navigators on a PC. The application, Map Command, is located in the bin directory of your leJOS installation. The file is nxjmapcommand.bat. Double click this file to run it and you will see a screen with a grid. Once you load in your map, you will see it displayed on screen. Use the zoom feature to adjust the view so your map takes up most of the screen (see Figure 13-8).

> **NOTE:** Map Command might not appear exactly as shown in the screenshots below due to version revisions.

Enter the name of your NXT brick in the connect field and press Connect. When the PC and NXT have established a Bluetooth connection you can begin making moves. Right click any part of the map area and select Go To. The robot will drive to that point. When it does, the GUI displays a trail where it has been (see Figure 13-9). You can also manually control the robot's travel distance and angle.

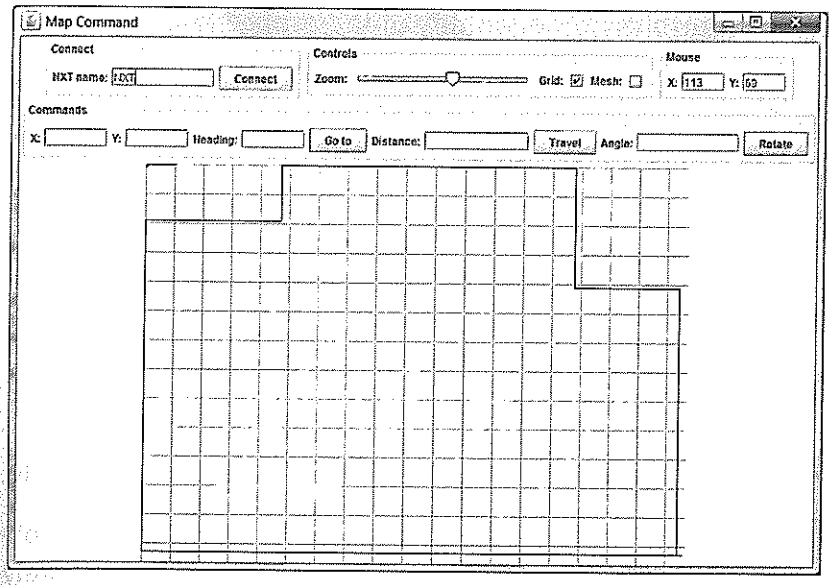


Figure 13-8: Adjusting the map zoom

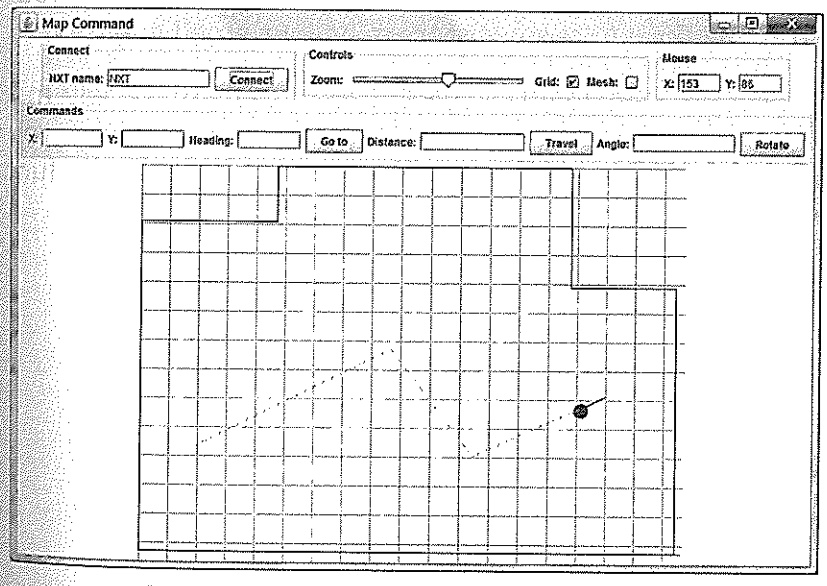


Figure 13-9: Traveling to coordinates on the map