Chapter 13

Knowing Where You Are

Solutions in this chapter:

- Choosing Internal or External Guidance
- Looking for Landmarks: Absolute Positioning
- Measuring Movement: Relative Positioning

Introduction

After our first few months of experimenting with robotics using the MINDSTORMS kit, we began to wonder if there was a simple way to make our robot know where it was and where it was going—in other words, we wanted to create some kind of navigation system able to establish its position and direction. We started reading books and searching the Internet, and discovered that this is still one of most demanding tasks in robotics and that there really isn't any single or simple solution.

In this chapter, we will introduce you to concepts of navigation, which can get very complex. We will start describing how positioning methods can be categorized into two general classes: *absolute* and *relative* positioning, the first based on external reference points, and the latter on internal measurements. Then we will provide some examples for both the categories, showing solutions and tricks that suit the possibilities of the MINDSTORMS system. In discussing absolute positioning, we will introduce you to navigation on pads equipped with grids or gradients, and to the use of laser beams to locate your robot in a room. As for relative positioning, we will explain how to equip your robot for the proper measurements, and will provide the math to convert those measurements into coordinates.

Choosing Internal or External Guidance

As we mentioned, there is no single method for determining the position and orientation of a robot, but you can combine several different techniques to get useful and reliable results. All these techniques can be classified into two general categories: *absolute* and *relative* positioning methods. This classification refers to whether the robot looks to the surrounding environment for tracking progress, or just to its own course of movement.

Absolute positioning refers to the robot using some external reference point to figure out its own position. These can be landmarks in the environment, either natural landmarks recognized through some kind of artificial vision, or more often, artificial landmarks easily identified by your robot (such as colored tape on the floor). Another common approach includes using radio (or light) beacons as landmarks, like the systems used by planes and ships to find the route under any weather condition. Absolute positioning requires a lot of effort: You need a prepared environment, or some special equipment, or both.

Relative positioning, on the other hand, doesn't require the robot to know anything about the environment. It deduces its position from its previous (known)

position and the movements it made since the last known position. This is usually achieved through the use of encoders that precisely monitor the turns of the wheels, but there are also inertial systems that measure changes in speed and direction. This method is also called *dead reckoning* (short for *deduced reckoning*).

Relative positioning is quite simple to implement, and applies to our LEGO robots, too. Unfortunately, it has an intrinsic, unavoidable problem that makes it impossible to use by itself: It accumulates errors. Even if you put all possible care into calibrating your system, there will always be some very small difference due to slippage, load, or tire deformation that will introduce errors into your measurements. These errors accumulate very quickly, thus relegating the utility of relative positioning to very short movements. Imagine you have to measure the length of a table using a very short ruler: You have to put it down several times, every time starting from the point where you ended the previous measurement. Every placement of the ruler introduces a small error, and the final result is usually very different from the real length of the table.

The solution employed by ships and planes, which use beacons like Loran or Global Positioning Systems (GPS) systems, and more recently by the automotive industry, is to combine methods from the two groups: to use dead reckoning to continuously monitor movements and, from time to time, some kind of absolute positioning to zero the accumulated error and restart computations from a known location. This is essentially what human beings do: When you walk down a street while talking to a friend, you don't look around continuously to find reference points and evaluate your position; instead, you walk a few steps looking at your friend, then back to the street for an instant to get your bearings and make sure you haven't veered off course, then you look back to your friend again.

You're even able to safely move a few steps in a room with your eyes shut, because you can deduce your position from your last known one. But if you walk for more than a few steps without seeing or touching any familiar object, you will soon lose your orientation.

In the rest of the chapter, we will explore some methods for implementing absolute and relative positioning in LEGO robots. It's up to you to decide whether or not to use any one of them or a combination in your applications. Either way, you will discover that this undertaking is quite a challenge!

Looking for Landmarks: Absolute Positioning

The most convenient way to place artificial landmarks is to put them flat on the floor, since they won't obstruct the mobility of your robot and it can read them with a light sensor without any strong interference from ambient light. You can stick some self adhesive tape directly on the floor of your room, or use a sheet of cardboard or other material over which you make your robot navigate.

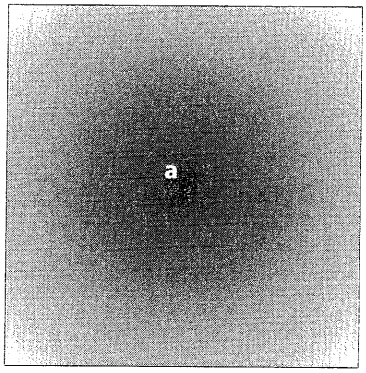
Line following, which we have talked a lot about, is probably the simplest example of navigation based on using an artificial landmark. In the case of line following, your robot knows nothing about where it is, because its knowledge is based solely on whether it is to the right or left of the line. But lines are indeed an effective system to steer a robot from one place to another. Feel free to experiment with line following; for example, create some interruptions in a straight line and see if you are able to program your robot to find the line again after the break. It isn't easy. When the line ends, a simple line follower would turn around and go back to the other side of the line. You have to make your software more sophisticated to detect the sudden change and, instead of applying a standard route correction, start a new searching algorithm that drives the robot toward a piece of line further on. Your robot will have to go forward for a specific distance (or time) corresponding to the approximate length of the break, then turn left and right a bit to find the line again and resume standard navigation.

When you're done and satisfied with the result, you can make the task even more challenging. Place a second line parallel to the first, with the same interruptions, and see if you can program the robot to turn 90 degrees, intercept the second line, and follow that one. If you succeed in the task, you're ready to navigate a grid of short segments, either following along the lines or crossing over them like a bar code.

You can improve your robot navigation capabilities, and reduce the complexity in the software, using more elaborate markers. As we explained in Chapter 4, the LEGO light sensor is not very good at distinguishing different colors, but is able to distinguish between differences in the intensity of the reflected light. You can play with black and gray tapes on a white pad, and use their color as a source of information for the robot. Remember that a reading at the border between black and white can return the same value of another on plain gray. Move and turn your robot a bit to decode the situation properly, or employ more than a single light sensor if you have them.

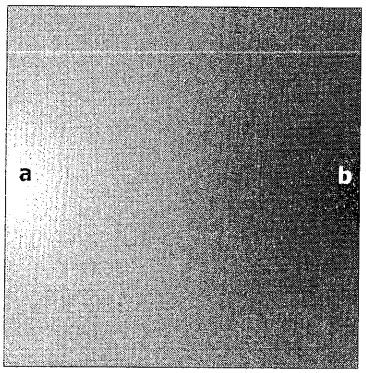
Instead of placing marks on the pad, you can also print on it with a special black and white gradient. For example, you can print a series of dots with an intensity proportional to their distance from a given point a. The closer to a, the darker the point; a is plain black (see Figure 13.1). On such a pad, your robot will be able to return to a from any point, by simply following the route until it reads the minimum intensity.

Figure 13.1 A Gradient Pad with a Single Attractor



The same approach can be used with two points a and b, one being white and the other black. Searching for the whitest route, the robot arrives at a, while following the darkest it goes to b (Figure 13.2). We first saw this trick applied during the 1999 Mindfest gathering at the Massachusetts Institute of Technology (MIT): two robots were playing soccer, searching for a special infrared (IR) emitting ball. When one got the ball, it used the pad to find the proper white or black goal. Months later, we successfully replicated this setup with Marco Berti during a demonstration at an exhibition in Italy.

Figure 13.2 A Gradient Pad with Two Attractors



DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS

Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS

Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS

Sync your system to PACER to automate legal marketing.