Includes link to **ONLINE ROBOT SIMULATOR**

- Teaches concepts of behavior-based programming
- Includes history and theory of behavior-based robots
- Provides example code for robot behaviors and behavior arbitration
- Provides insight into building real-world robotic applications

Robot Programming

A Practical Guide to Behavior-Based Robotics

TAB ROBOTICS

JOSEPH L. JONES

Robotic Simulator by Daniel Roth

Robot Programming

A Practical Guide to Behavior-Based Robotics

Joseph L. Jones

Robotic Simulator by Daniel Roth

McGraw-Hill

New York Chicago San Francisco Lisbon
London Madrid Mexico City Milan
New Delhi San Juan Seoul
Singapore Sydney Toronto

The McGraw-Hill Companies

Cataloging-in-Publication Data is on file with the Library of Congress

1 2 3 4 5 6 7 8 9 0 DOC/DOC 0 9 8 7 6 5 4 3

ISBN 0-07-142778-3

The sponsoring editor for this book was Judy Bass and the production supervisor was Pamela A. Pelton. It was set in Melior by Patricia Wallenburg.

Printed and bound by RR Donnelly.

This book is printed on recycled, acid-free paper containing a minimum of 50 percent recycled, de-inked fiber.

To Sue, Kate, and Emily

Preface

I got my first taste of robot programming in the early 1980s when I joined the research staff at MIT's Artificial Intelligence Laboratory. My group was trying to solve a classic challenge in robotics called pick-and-place—make a robot pick up an object at one spot and put it down somewhere else. Given an object and a destination all the robot has to do is to figure out the actual arm and gripper motions needed to move the object to the goal—the sort of thing any two-year-old can do. Four of us worked on the problem for about five years.[1]

Other group members worked on the parts of the program that would generate the large-scale motions of the robot arm, motions to move the arm from one region of the workspace to another. My job was to write the software that would enable our robot arm (see **Figure P.1**) to work out how to move the last few inches toward an object and grasp the object. The solution to the overall problem has many constraints: the robot has to grasp the object at a viable spot; the robot must avoid bumping into anything as it moves about; the robot must avoid violating what are called kinematic constraints.[2]

[1]See *Handey: a Robot Task Planner* by Tomás Lozano-Pérez, Joseph L. Jones, Patrick A. O'Donnell, and Emmanuel Mazer, MIT Press, 1992

[2]A kinematic constraint is a limitation on how a robot can move. If the robot can say, position joint A only between the angles 0 and 120 degrees, the con-

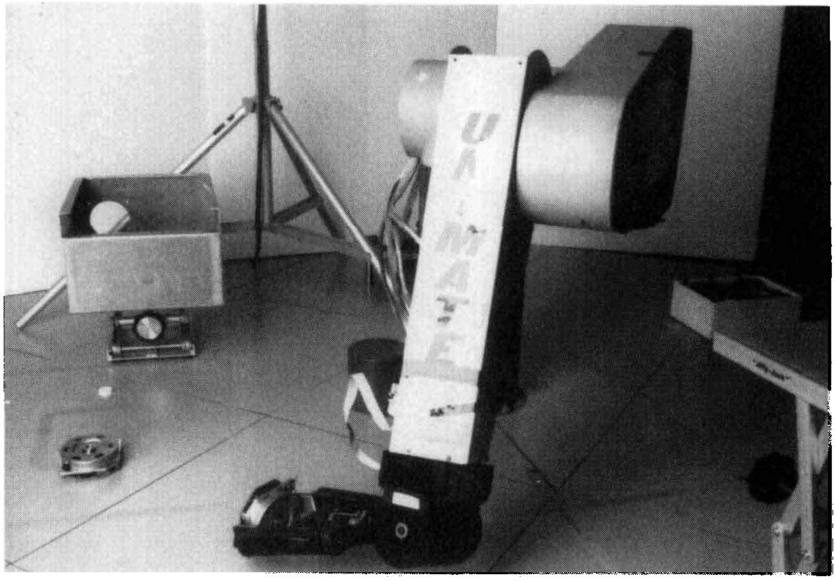

Figure P.1

This Puma model 560 with custom-built gripper was one of the manipulator robots used in the Handey project. In the foreground the robot picks up a motor that will be added to an assembly (contained in the white box) at the back right. The Handey program generates all the joint motion commands needed to move the motor from the point where it is picked up, avoiding all the obstacles, and insert it into the assembly. *(Photograph courtesy of Prof. Tomás Lozano-Pérez of the MIT Artificial Intelligence Laboratory.)*

Yet another part of our work was to write code that would figure out how to reposition the object in the robot's gripper if the initial grasp conflicted with obstacles or the robot's kinematic limits at the putdown point. The tricky bit was that our software was supposed to be completely general—the code had to work for any robot, in any environment, transporting any part.

In order to accomplish all these things we had to first build a world model. A world model tells the robot the geometric shape of every object in the robot's workspace and where every object is located in relation to the robot. And in the same meticulous way that we modeled the environment, we also had to model the robot and to program the equations that described the robot's

trolling program must refrain from instructing the robot to move joint A to 135 degrees.

kinematics—how the robot's joints relate to each other and in which ways and how far each joint is able to move.

Our task was excruciating. Any small error in the world model could cause the robot to collide with an object when the robot tried to execute the motions it had planned. Any little mistake in the equations that describe the robot meant the robot might fail to reach the designated pickup-object or whack something along the way. If one of us accidentally bumped some object in the robot's workspace, thus creating a mismatch between the real world and the robot's world model, the robot would most likely strike that object. And, when at last the robot came up with a successful plan for moving an object from one place to another, the robot's motions invariably looked awkward and unnatural.

We hoped that our work would have the practical result of making manufacturing by robots faster and more flexible. Our software should enable assembly line designers to describe in generic terms what they wanted the robot to do rather than telling the robot in precise detail how to do it. Using the most sophisticated computers and the best thinking available at the time we succeeded in solving an interesting academic problem.

However, our efforts did little to change the way robots manufacture products. It still makes economic sense for assembly line workers to follow the painstaking process of teaching their robots each and every tiny motion needed to perform an assembly. Robots that plan such motions by themselves can't compete—they don't seem to add enough value to earn their keep.

At the same time that I worked on the pick-and-place problem another team at the AI Lab tackled a different robotic challenge—getting autonomous mobile robots to negotiate a real world environment. The Mobile Robot[3] group focused on a different class of robots in a different environment and took an approach fundamentally different from the one my group followed.

Insects fascinated the Mobile Robot folks. They noted that these creatures are a marvel in a minuscule package. In a complex and

[3]The Mobile Robot group was established and led by Prof. Rodney Brooks.

dangerous world, insects manage to find food, shelter, and mates. Insects escape from predators. Insects navigate their world and don't get lost. And sometimes insects even seem to cooperate in building large structures and in performing other impressive feats. Yet insects have the tiniest of brains. For many insects, sight is accomplished using primitive vision systems-systems that boast fewer pixels than a cheap video camera. What were dumb bugs doing that put our best robots to shame?

The (partial) answer that the Mobile Robot group and others developed is behavior-based robotics. Behavior-based robotics is having an impact not just in academia but in the larger world as well. Sojourner, the robot that successfully explored a little part of Mars in 1997, used behavior-based programming to achieve its otherworldly feat. But a little floor cleaning robot called Roomba® provides us with a more, shall we say, down-to-earth example of behavior-based robotics. See **Figure P.2**.

Many extol Roomba®[4] Robotic FloorVac as the world's first practical consumer robot. Indeed Roomba® has established a growing

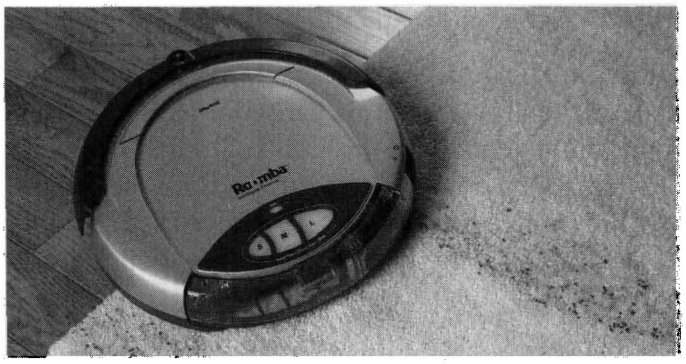

Figure P.2

Roomba® is a widely available floor cleaning robot manufactured by iRobot, Corporation of Burlington, MA. Roomba uses a behavior-based programming scheme. *(Photo courtesy of iRobot Corporation, Burlington, MA)*

[4]Many people at iRobot Corporation of Burlington, Massachusetts (www.irobot .com) made crucial contributions to the development of the Roomba® Robotic FloorVac development. The original team included Paul Sandin, Phil Mass, Eliot Mack, Chris Casey, Winston Tao, Jeff Ostezewski, Sara Farragher, and Joe Jones.

presence in a habitat previously far more forbidding to robots than even the dusty plains of Mars—the display shelves at mass-market retailers. Like Sojourner, Roomba® abides by the principles of behavior-based robotics. Those principles endow both robots with significant capabilities: the ability to make do with a small, low-end processor, to respond quickly to sensory inputs; to perform robustly; and to degrade gracefully in the presence of inaccurate data and partial sensory failure.

Because of its modest computational requirements and ease of implementation, behavior-based robotics is very well suited to the needs and abilities of hobbyists, students, and robot enthusiasts alike. By learning the principles of behavior-based robotics you will be able to create robots that are affordable, responsive, robust, fun, and maybe even useful.

Joe Jones

Acknowledgments

Many people helped to bring this book into being. Judy Bass of McGraw-Hill provided the initial impetus by proposing that I write a new book on robotics. The material of Chapter 8 is derived from a project that Ben Wirz and I worked on for several years. Others made important contributions by reviewing the manuscript and offering crucial suggestions. They include: Adam Craft, Matt Cross, Branden Gunn, Danniel Ozick, Paul Sandin, Steve Shamlian, Jennifer Smith, Sue Stewart, Chuck Rosenberg, Clara Vu, Greg White, Bill Wong, and Holly Yanco.

Gratitude is also due to the growing numbers of robotics enthusiasts around the world—hobbyists, students, educators, and researchers alike. These are the individuals who make robots (and books about robots) possible; they comprise the fount from which future progress in robotics will flow.

Introduction

There are many ways to program a mobile robot. The least complex robots have programs written in solder[1] where sensors are connected more or less directly to motors. At the other extreme are robots programmed in high-level languages—languages like Lisp and Java that are often used to support artificial intelligence. But regardless of how the programming is accomplished, all robot programs exhibit some sort of structure or architecture.

Programs composed by beginning roboticists often have a structure that might be described as *ad hoc* or perhaps organic—the programs just grow. Without any overarching principles or methodology, the programmer thinks of a feature and writes the code that implements that feature. A second feature springs from the fertile mind of the programmer and is combined with the first, a third feature comes to be, and so on. As development proceeds, the organic analogy becomes more and more apropos. A bit of code that implements one aspect of the program intertwines with code that implements another, magic numbers take root, and special cases flourish and spread.

[1]For an approach to robotics rather different from the microprocessor-based systems in this book you may want to investigate the innovative work of Mark Tilden. Start at the Web site: *http://www.nis.lanl.gov/projects/robot/*.

The organic approach *does* work. With enough time, patience, and code space, the programmer can coerce an organically structured robot program into producing desired results. But without the guidance of robot-specific principles, such programs become more and more convoluted. It always seems to take longer to implement the next feature than it took to implement the last; programmers often refrain from uprooting obsolete features because bugs tend to crop up when "unused," but highly cross-coupled sections of code are deleted. Even when programmers follow modern tenets of modularity and other good programming practices, robot programs can become cumbersome and fragile.

Robot/Computer Differences

Programming trouble develops because beginning roboticists often miss the importance of a basic fact: robots and computers are different. The goals of a robot program and a computer program are different; the constraints imposed on computers and robots are different—and these differences are crucial. A roboticist must understand and respect the distinctions if he or she is to develop effective robot programs.

Serial versus Parallel

The intuition that programmers develop learning to program computers often does not carry over to programming robots. One aspect of programmer intuition that sometimes fails concerns the issue of serial versus parallel execution. Serial execution is satisfactory for most computer programs, but robot programs demand a parallel approach.

A typical computer program is designed to compute an answer and return a result. At their core, even highly interactive and intensely graphical computer programs like video games follow this basic approach. Computation proceeds in a sequence of steps where, typically, the output of one step becomes the input of the next step. The total time required to reach an answer is the sum of the times taken for each step. Therefore, a faster computer is always a better computer—the user of a faster computer gets the answer more quickly than the user of a slower computer.

Alternately, a faster computer can give a more precise answer or can give more answers in the same time taken by a slow computer. For a video game, computer speed translates directly into better resolution and more realistic simulations.

But computing an answer is *not* the purpose of an autonomous mobile robot. Rather, a robot seeks to achieve a goal or maintain a state while avoiding hazards and traps—very much as a living organism does. The robot must attend *simultaneously* to all of its concerns; e.g., don't collide with anything, don't fall down the steps, don't run out of power far from the charger. Catastrophe might result if, for example, the robot were to neglect watching out for the edge of the stairs while concentrating exclusively on avoiding a collision with the baseboard. A robot crash can be a rather more serious matter than a computer crash.

Plans versus Opportunities

A typical computer program executes a plan—one thing happens after another until a result is reached. But an autonomous robot needs to be opportunistic—sometimes the robot's goal is already achieved; all the robot has to do is notice. For example, suppose we want our robot to find its way to a charger when the batteries are low. A plan-based program deciding to recharge the batteries might first have the robot search for a central beacon in the room, go to the beacon, orient itself in a particular direction (toward the charger), then proceed until the robot encounters the local beacon that indicates the charger.

This sounds like a reasonable plan—unless the robot happens to be positioned only a foot from the charger when the robot decides that the batteries need recharging. In that case, the plan-following robot first moves *away* from the charger to find the central beacon, only to turn back immediately toward the charger. A more sensible approach would have the robot notice that it was next to the charger to begin with, and then act accordingly, rather than follow a plan blindly, ignoring real-world opportunities. Constant input from sensors and a programming paradigm able to make use of the rush of data are needed to enable a robot to take advantage of its opportunities.

Graceful Degradation

Given accurate data and a logically correct program, a computer will return a correct result. Given inaccurate data, the computer's output is unreliable. This observation gives rise to the well-known GIGO adage, "Garbage in, garbage out." The computer depends on a human operator to input accurate data and has little recourse if the data are wrong. Thus incorrect computer responses are ascribed to "human error."

An autonomous robot collects its own data through its sensors. And sensors, as we shall see, often mislead. The less you pay for a sensor, the less inclined it is toward veracity. But even very expensive sensors provide unreliable data in common situations. Because of the unreliability of its inputs, a robot program must be engineered in such a way that the robot works as well as possible under the circumstances. That is, robot performance should degrade gracefully in the presence of inaccurate or missing data. A robot program must not collapse in a heap (as a computer program might) at the first sign of erroneous input.

Behavior-Based Advantages

Behavior-based approaches to robot programming excel at parallel execution, opportunistic goal realization, and graceful degradation. Programming a robot according to behavior-based principles makes the program inherently parallel, enabling the robot to attend simultaneously to all the hazards it may face as well as the serendipitous opportunities it may encounter. Further, behavior-based robots can easily accommodate methods that allow performance to degrade gracefully in the presence of sensor error or failure.

What's All the Fuss?

Experienced programmers may be wondering: Is there anything new here? Isn't an autonomous mobile robot really just an example of an embedded system plus a real-time operating system? Might "behavior-based programming" be nothing more than a fancy name applied to commonplace practices?

Mobile robots do indeed qualify as embedded systems. And whether the robot's software runs under a commercially supplied real-time kernel or is implemented in raw code, every robot must have some semblance of a real-time operating system. Further, the good practices a roboticist follows in constructing a behavior-based program are not inconsistent with the good practices an embedded system programmer might follow writing code that controls, say, a DVD player or a cell phone.

However, by themselves, common practices for writing embedded system code are not sufficient for constructing effective robot programs. Autonomous robots, as will become increasingly clear, regularly confront challenges rarely faced by other embedded systems. Dealing with these challenges calls for the additional set of organizing principles that behavior-based programming supplies.

Focus

The behavior-based robot programming paradigm is an eminently practical one and likewise, throughout this text, my treatment of behavior-based robotics will focus more on practical issues than on academic rigor.[2] My intention is to provide you with important fundamentals that will help you program robots effectively. I hope that you will take away from your study of behavior-based robotics an appreciation of the power and scope of this robot programming method and that you will feel confident implementing a behavior-based approach in your future robotic projects.

In this text, I will explain robot programming theory, offer examples of successful robot programs, and relate insights that I have found useful. But ultimately you cannot learn robot programming from a book. To learn robot programming, you must program robots. Unfortunately, because of cost and other complications, not everyone has immediate access to a programmable

[2]For an excellent example of a more rigorous approach see *Behavior-Based Robotics* by Ronald Arkin, MIT Press, 1998. See also course notes prepared by long-time behavior-based robotics researchers Ian Horswill at http://www.cs.northwestern.edu/academics/courses/special_topics/395-robotics/.

mobile robot. To address this difficulty, my colleague, Daniel Roth, has developed a robotic simulator, BSim; we have integrated BSim with the text. To access BSim please visit the Web site: www.behaviorbasedprogramming.com.

We recommend that you make a practice of running BSim frequently as you work toward understanding the concepts presented here. As a pedagogical tool, robot simulators can be quite helpful. BSim lets you isolate aspects of robot behavior and slow down processes so that you can easily observe what is going on. However, as a predictor of how a physical robot will interact with physical objects, simulators are notoriously unreliable. *Do not assume that what works in simulation will work in the real world.* To fully appreciate how a robot behaves in the physical world there is no substitute for programming an actual, physical robot.[3–6]

Prerequisites

What do you need to know before you begin? Because robots have such a broad appeal and attract a diverse set of enthusiasts, it is hard to choose a firm set of prerequisite knowledge. Each reader will bring a different set of interests, experience, and willingness to acquire missing understanding. However, in general,

[3]For a compilation of inexpensive robots see *Personal Robotics: Real Robots to Construct, Program, and Explore the World* by Richard Raucci; AK Peters, Ltd., 1999. More recent offerings may be found by searching the Web for "Robot Kits." Also, try http://www.robotstore.com.

[4]The possibilities for experimenting with behavior-based robotics have expanded in recent years. LEGO offers a product called Mindstorms (http://www.legomindstorms.com) powered by the RCX, a user-programmable controller. The RCX, though it possesses only a limited number of inputs and outputs, facilitates the construction of many interesting robots. Enthusiasts have greatly extended the abilities of the RCX by creating new software systems (some of them free) compatible with the controller. One such system is called LeJOS. This system allows you to program the RCX in Java. LeJOS even provides libraries that promote a behavior-based approach! Look for information on LeJOS at: http://lejos.sourceforge.net/.

[5]A respected organization that promotes learning about robots and behavior-based programming is the KISS Institute for Practical Robotics; see http://www.kipr.org/.

[6]For the authoritative text on building robots using the LEGO construction system, see *Robotic Explorations,* Fred Martin, Prentice-Hall, Inc., 2001.

a reader will have an easier time if he or she has a working knowledge of algebra and trigonometry and has some experience with vectors. You will miss little if you have yet to master linear algebra and calculus. It will be helpful to have basic familiarity with computer programming before reading this book, but no great depth of experience is assumed.

Organization

In Chapter 1, we use BSim to observe the behavior of a working system, a simulated robot. The simulated robot exhibits interesting and complex interactions with its environment. To understand these interactions we take a step back and ask: what exactly is a robot and what are its essential components?

In Chapter 2, we review the feedback control system, a traditional method of connecting sensing to actuation. Returning to BSim, we observe the functioning of elementary control systems. But from the very beginning we will learn the painful lessons of how good control systems can go bad.

In Chapter 3, we begin to build primitive behaviors, learning about triggers and about ballistic versus servo behaviors.

Chapter 4 deals with arbiters, the software construct that all behavior-based systems must have to manage conflicts between behaviors.

Putting together all that we have learned, in Chapter 5, we can write complete programs. We commence filling a bag of useful robot programming tricks.

With the basic science now mastered, we proceed to the art of robot programming in Chapter 6. Here we address how a problem statement can be converted into a robot program. Principles and heuristics to guide this perilous but necessary step are suggested.

In Chapter 7, we take a software-centered look at some common sensors, how they function, and the ways that sensor output can be misleading.