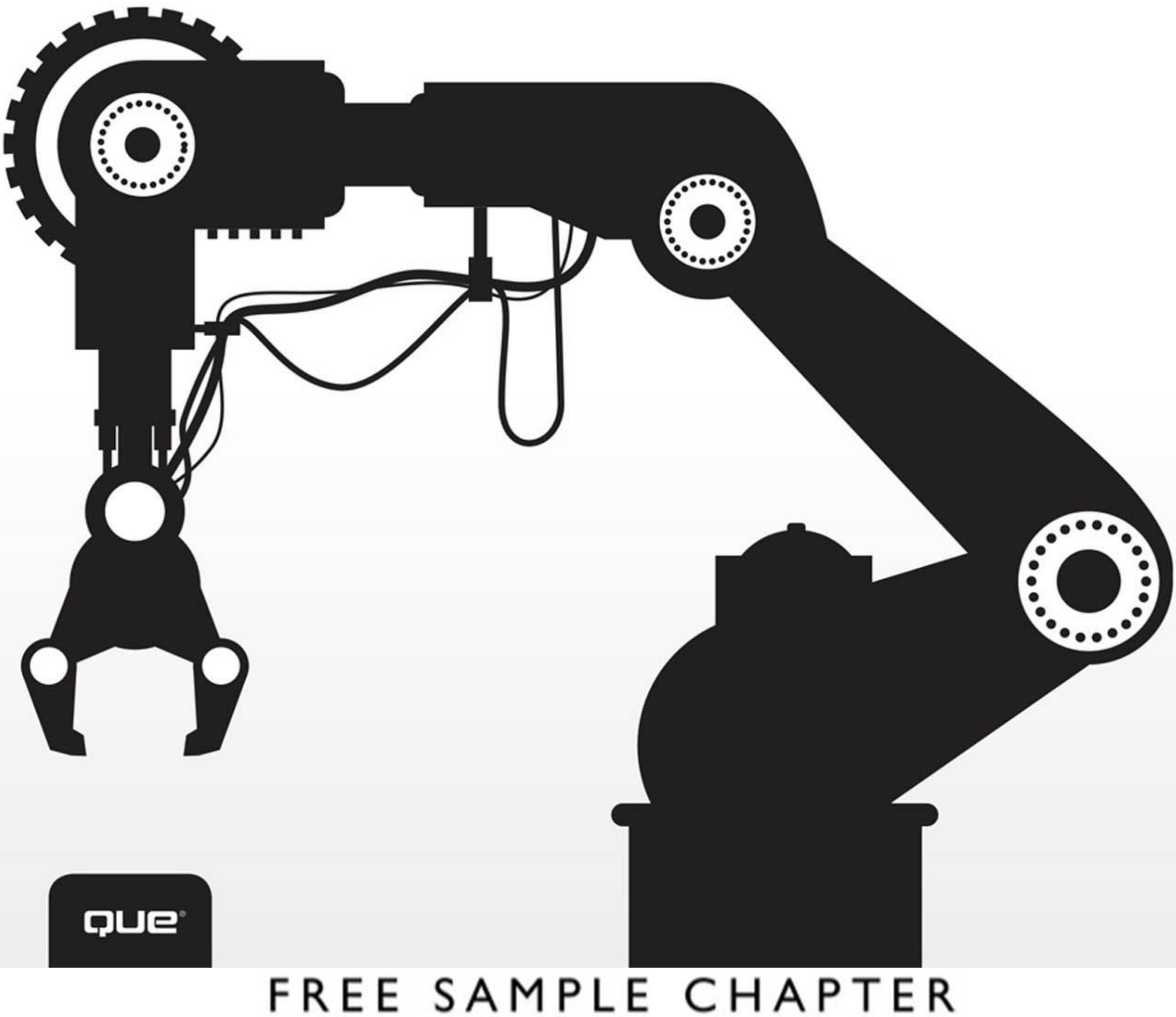
CAMERON HUGHES • TRACEY HUGHES

ROBOT PROGRAMMING

A GUIDE TO CONTROLLING AUTONOMOUS ROBOTS



QUE

FREE SAMPLE CHAPTER

SHARE WITH OTHERS

Robot Programming: A Guide to Controlling Autonomous Robots

Cameron Hughes
Tracey Hughes

800 East 96th Street
Indianapolis, Indiana 46240

ROBOT PROGRAMMING: A GUIDE TO CONTROLLING AUTONOMOUS ROBOTS

ISBN-13: 978-0-7897-5500-1
ISBN-10: 0-7897-5500-9

Library of Congress Control Number: 2015955656

First Printing: May 2016

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Que Publishing cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an "as is" basis. The authors and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book.

Special Sales

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the U.S., please contact intlcs@pearson.com.

Editor-in-Chief
Greg Wiegand

Executive Editor
Rick Kughen

Senior Acquisitions Editor
Laura Norman

Development Editor
William Abner

Technical Editor
John Baichtal

Managing Editor
Sandra Schroeder

Project Editor
Mandie Frank

Copy Editor
Geneil Breeze

Indexer
Ken Johnson

Proofreader
Gill Editorial Services

Editorial Assistant
Cindy Teeters

Cover Designer
Chuti Prasertsith

Compositor
Bronkella Publishing

CONTENTS AT A GLANCE

CONTENTS

ABOUT THE AUTHORS

Cameron Hughes is a computer and robot programmer. He holds a post as a Software Epistemologist at Ctest Laboratories where he is currently working on A.I.M. (Alternative Intelligence for Machines) and A.I.R. (Alternative Intelligence for Robots) technologies. Cameron is the lead AI Engineer for the Knowledge Group at Advanced Software Construction Inc., a builder of intelligent robot controllers and software-based knowledge components. He holds a staff appointment as a Programmer/Analyst at Youngstown State University.

Tracey Hughes is a senior software and graphics programmer at Ctest Laboratories and Advanced Software Construction Inc. where she develops user interfaces and information and epistemic visualization software systems. Her work includes methods of graphically showing what robots and computers are thinking. She is on the design and implementation teams for the East-Sidaz robots at Ctest as well.

Both Cameron and Tracey Hughes are members of the advisory board for the NREF (National Robotics Education Foundation) and members of the Oak Hill Collaborative Robotics Maker Space. They are project leaders of the technical team for the NEOACM CSI/CLUE Robotics Challenge and regularly organize and direct robot programming workshops for the Arduino, Mindstorms EV3, LEGO NXT, and RS Media robot platforms. Cameron and Tracey are two of the authors of *Build Your Own Teams of Robots with LEGO® Mindstorms® NXT and Bluetooth*, published by McGraw-Hill/TAB Electronics, January 2013. They have written many books and blogs on Software Development and Artificial Intelligence. They've also written books on multicore, multithreaded programming, Linux rapid application development, object-oriented programming, and parallel programming in C++.

Dedication

We dedicate this book to all those open source robot maker spaces that in spite of humble and meager resources continue to toil against the improbable and do amazing things with robots.

ACKNOWLEDGMENTS

We are greatly indebted to Valerie Cannon who played the role of "on location" robo-journalist and photographer for us at the 2015 DARPA Robotics Search and Rescue Challenge at the Fairplex in Pomona, California.

We would like to thank our two interviewees for our "Bron's Believe It or Not" interviews. We also thank Ken Burns from Tiny Circuits of Akron, Ohio, who provided us with a personal tour of his Arduino manufacturing space and endured our probing interview questions. Portions of the material on Arduino robotics hardware, especially the Phantom X Pincher Robot Arm, would not have been possible without the time and interview given to us from Kyle Granat at Trossen Robotics.

We are also indebted to the NEOACM CSI-Clue robotics challenge team who acted as a sounding board and early test bed for many of the robot example programs in this book. We are fortunate to be part of Ctest Laboratories, which provided us with unfettered access to their East Sidaz and Section 9 robots. The East Sidaz and Section 9 met every challenge we could throw at them. A special thanks to Pat Kerrigan, Cody Schultz, Ken McPherson, and all the folks at the Oak Hill Collaborative Robotics Maker Space who allowed us to subject them to some of our early robot designs. A special thanks to Howard Walker from Oak Hill Collaborative who introduced us to the Pixy camera. Thanks to Jennifer Estrada from Youngstown State University for her help with the Arduino-to-Bluetooth-to-Vernier magnetic field sensor connection and code. A special thanks goes to Bob Paddock for offering his insight and expertise on sensors and giving us a clear understanding of the Arduino microcontroller. A shout-out to Walter Pechenuk from IEEE Akron, Ohio, chapter for his subtle, cool, and calm interaction and responses as we went on endlessly about our approach to autonomous robotics. Further, this simply could not have been written without the inspiration, tolerance, and indirect contribution of many of our colleagues.

WE WANT TO HEAR FROM YOU!

As the reader of this book, *you* are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

We welcome your comments. You can email or write to let us know what you did or didn't like about this book—as well as what we can do to make our books better.

Please note that we cannot help you with technical problems related to the topic of this book.

When you write, please be sure to include this book's title and author as well as your name and email address. We will carefully review your comments and share them with the author and editors who worked on the book.

Email: feedback@quepublishing.com

Mail: Que Publishing
ATTN: Reader Feedback
800 East 96th Street
Indianapolis, IN 46240 USA

READER SERVICES

Register your copy of *Robot Programming* at quepublishing.com for convenient access to downloads, updates, and corrections as they become available. To start the registration process, go to quepublishing.com/register and log in or create an account*. Enter the product ISBN, 9780789755001, and click Submit. Once the process is complete, you will find any available bonus content under Registered Products.

*Be sure to check the box that you would like to hear from us in order to receive exclusive discounts on future editions of this product.

INTRODUCTION

ROBOT BOOT CAMP

caution

We who program robots have a special responsibility to make sure that the programming is safe for the public and safe for the robots. The safety of robot interaction with humans, animals, robots, or property is a primary consideration whenever a robot is being programmed. This is true for all kinds of robot programming and especially true for programming autonomous robots, which is the kind of robot programming that we explain in this book. The robot commands, instructions, programs, and software presented in this book are meant for exposition purposes only and as such are *not* suitable for safe public interaction with people, animals, robots, or property.

A serious treatment of robot safety is beyond the scope of this introductory book. Although the robot examples and applications presented in this book were tested to ensure correctness and appropriateness, we make no warranties that the commands, instructions, programs, and software are free of defects or error, are consistent with any particular standard of merchantability, or will meet your requirements for any particular application.

The robot code snippets, programs, and examples are meant for exposition purposes only and should not be relied on in any situation where their use could result in injury to a person, or loss of property, time, or ideas. The authors and publisher disclaim all liability for direct or consequential damages resulting from your use of the robots, commands, instructions, robot programs, and examples presented in this book or contained on the supporting website for this book.

Robot Programming Boot Camp

Welcome to *Robot Programming: A Guide to Controlling Autonomous Robots*. This robot programming "boot camp" ensures that you have all the information needed to get started. We have built and programmed many types of robots ranging from simple single-purpose robots to advanced multifunction autonomous robot teams and have found this short robot programming boot camp indispensable for those who are new to programming robots or who want to learn new techniques to program robots.

Ready, Set, Go! No Wires or Strings Attached

There are two basic categories for robot control and robot operation as shown in Figure I.1.

The telerobot group represents robot operations that are remotely controlled by a human operator using some kind of remote control device or puppet mode. Some remote controls require a tether (a wire of some sort) to be physically connected to the robot, and other types of remote control are wireless (for example, radio control or infrared).

The autonomous robot group represents the kind of robot that does not require a human operator. Instead, the robot accesses a set of instructions and carries them out autonomously without intervention or interruption from a remote control.

In this book, we focus on the autonomous group of robot operations and robot programming. Although we often discuss, explain, and contrast telerobots and autonomous robots, our primary focus is on introducing you to the basic concepts of programming a robot to operate and execute assigned tasks autonomously.

As you see in Chapter 9, "Robot SPACES," there are hybrids of the two types of robot control/operation with different mixes and matches for operation strategies. You are introduced to techniques that allow for mixing and matching different robot control strategies.

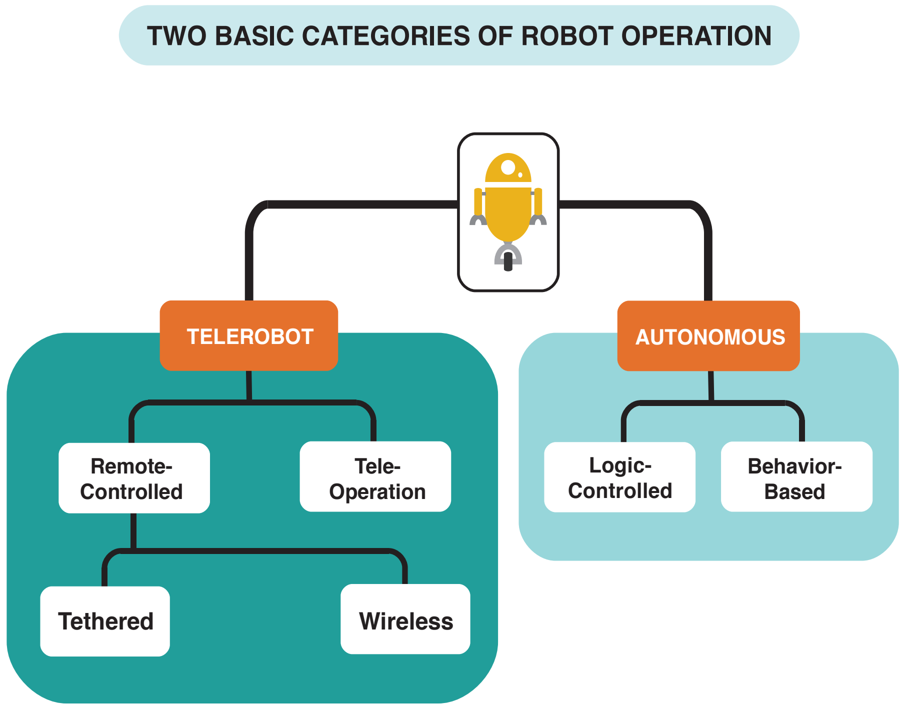
caution

Although *Robot Programming: A Guide to Controlling Autonomous Robots* does not assume that you have any previous experience programming robots, to get the most out the book it is assumed that you are familiar with basic programming techniques in standard programming languages such as Java or C++. While the book does present all the final robot programs in Java or C++, the basic robot instruction techniques and concepts are presented with diagrams or plain English first. The book also introduces you to approaches to program design, planning, and analysis such as RSVP (Robot Scenario Visual Planning) and REQUIRE (Robot Effectiveness Quotient Used in Real Environments) .

note

All robot instructions, commands, and programs in this book have been tested on ARM7, ARM9 microcontroller-based robots as well as on the widely available and popular LEGO NXT and EV3-based robots. All other robot-based software used in this book was tested and executed in Mac OSX and Linux environments.

Figure I.1
The two basic categories of robot operation



Boot Camp Fundamentals

Five basic questions must be answered prior to any attempt to program a robot:

1. What type of robot is being considered?
2. What is the robot going to do?
3. Where is the robot going to do it?
4. How is the robot going to do it?
5. How will the robot be programmed?

Many beginner and would-be robot programmers fail to answer these basic questions and end up with less than successful robot projects. Having the answers to these fundamental questions is the first step in the process of getting any kind of robot to execute the assigned task. In *Robot Programming: A Guide to Controlling Autonomous Robots* we demonstrate how these questions and their answers are used to organize a step-by-step approach to successfully instructing a robot to autonomously carry out a set of tasks.

Core Robot Programming Skills Introduced in This Book

In this book, we introduce you to the following basic techniques of the Robot Boot Camp shown in Table I.1.

Table I.1 The Boot Camp Notes

Techniques	Description
Robot motion planning & programming	Arm movement Gripper programming End-effector movement Robot navigation
Programming the robot to use different types of sensors	Infrared sensors Ultrasonic sensors Touch sensors Light sensors RFID sensors Camera sensors Temperature sensors Sound sensors Analysis sensors
Motor use	Motors used in robot navigations Motors used in robotic arms, grippers, and end-effectors Motors used in sensor positioning
Decision-making	Robot action selection Robot direction selection Robot path selection
Instruction translation	Translating English instructions and commands into a programming language or instructional format that a robot can process

These techniques are the core techniques necessary to get a robot to execute almost any assigned task. Make note of these five areas because they represent the second step in building a solid foundation for robot programming.

BURT—Basic Universal Robot Translator

In this book, we use two aids to present the robot programs and common robot programming issues in an easy-to-understand and quick reference format. The first aid, *BURT (Basic Universal Robot*

Translator), is used to present all the code snippets, commands, and robot programs in this book. BURT shows two versions of each code snippet, command, or robot program:

- Plain English version
- Robot language version

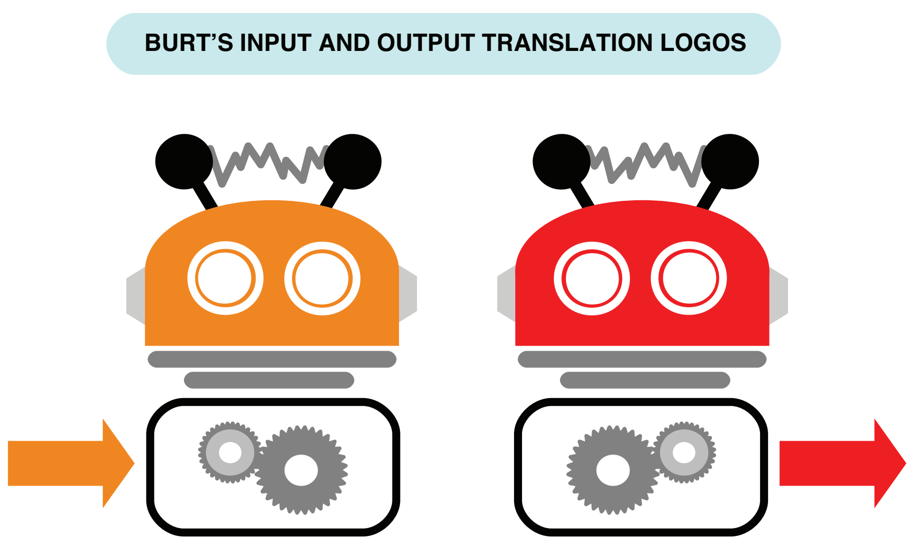
BURT is used to translate from a simple, easy-to-understand English version of a set of instructions to the robot language version of those instructions.

In some cases the English version is translated into diagrams that represent the robot instructions. In other cases, BURT translates the English into standard programming languages like Java or C++. BURT can also be used to translate English instructions into robot visual instruction environments like Labview or LEGO's G language for Mindstorms robots.

The BURT Translations are numbered and can be used for quick reference guides on programming techniques, robot instructions, or commands. BURT Translations have two components; an input and an output component. The input component will contain the pseudocode, or RSVPs. The output component will contain the program listing, whether it be a standard language or visual instruction. They will be accompanied with the BURT Translation Input or Output logo as shown in Figure I.2.

In addition to BURT Translations, this book contains BURT Gotchas, a.k.a. BURT's ***G***lossary of ***T***echnical ***C***oncepts and ***H***elpful ***A***cronyms. The world of robot programming is full of technical terms and acronyms that may be unfamiliar or tricky to recall. BURT Gotchas provide a convenient place to look up any acronym or some of the more technical terms used in this book. In some cases BURT Gotchas are listed at the end of the chapter in which they are first used, but a complete list of all of BURT Gotchas can be found in the book's glossary.

Figure I.2
BURT Translation Input and Output logos



BRON—Bluetooth Robot Oriented Network

The second aid is *BRON (Bluetooth Robot Oriented Network)*. We have put together a small team of robots that are connected and communicate through Bluetooth wireless protocols and the Internet. It is the responsibility of this team of robots to locate and retrieve useful tips, tricks, little-known facts, interviews, and news from the world of robot programming that the reader will find interesting and helpful. This material is presented in sections titled *BRON's Believe It or Not* and are identified by the logo shown in Figure I.3.

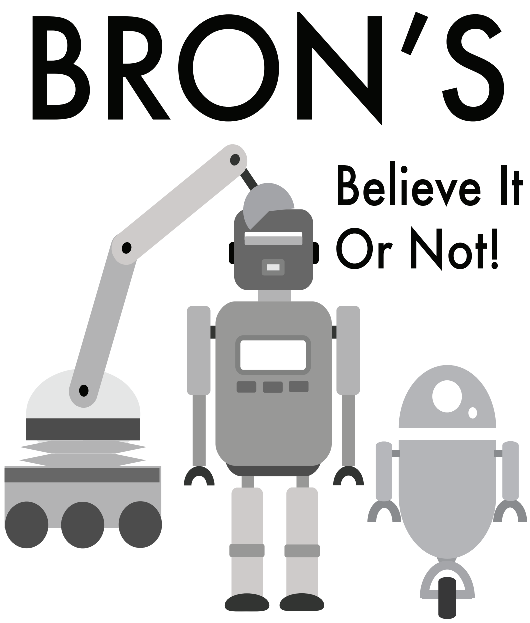


Figure I.3
BRON's Believe It or Not logo

These sections contain supplementary material that the reader can skip, but often offer additional insight on some idea that has been presented in the chapter. In some instances, a BRON's Believe It or Not contains news that is hot off the presses and relevant to some aspect of robot programming. In other instances, a BRON section contains excerpts from interviews of individuals making important contributions to the world of robotics or robot programming. In all cases, BRON's Believe It or Not sections are designed to give you a deeper understanding and appreciation for the world of robotics and robot programming.

Assumptions About the Reader's Robot(s)

Robot Programming: A Guide to Controlling Autonomous Robots can be read and much can be learned without any access to robots at all. Most chapters explain the concepts in plain English and are reinforced with diagrams. However, to get the maximum benefit from reading this book, it is assumed you will try out and test the commands, instructions, or programs on robots that you have access to.

We used and tested the instructions and programs in this book on several different types of robots, and the ideas presented in this book broadly apply to many classes of robots. If you have access to a robot with *at least one* capability from each column shown in Table I.2, you will be able to adapt any program in this book to your robot.

note

We do show you how to program a robot to use other sensors beyond those listed in Table I.2. But the main ideas in the book can be tried and tested with only those listed in Table I.2.

Table I.2 The Boot Camp's Matrix of Robot Capabilities

Movement Capability	Sensing	Actuating	Control
Wheels	Infrared	Gripper	ARM7 Microcontroller
Bipedal	Ultrasonic	Robot arm	ARM9 Microcontroller
Quadruped	Camera	Pusher	LEGO Mindstorms EV3 Microcontroller
Hexaped (etc.)	Heat		LEGO Mindstorms NXT Microcontroller
Aerial	Light		Arduino
	Color		ARM Cortex/Edison Processor
	Touch		

How Midamba Learned to Program a Robot

In this book, we tell a short story of how free-spirited, fun-loving Midamba found himself in a precarious predicament. As luck would have it, his only chance out of the predicament required that he learn how to program robots. Although Midamba had some basic experience programming a computer, he had very little knowledge of robots and no experience programming them. So throughout the book, we use Midamba's predicament and his robot programming triumph as an example. We walk you through the same basic lessons that Midamba learned and the steps he had to take to successfully program his first robot.

This page intentionally left blank

3

RSVP: ROBOT SCENARIO VISUAL PLANNING

Robot Sensitivity Training Lesson #3: *Don't instruct the robot to perform a task you can't picture it performing.*

As described in Chapter 2, "Robot Vocabularies," the *robot vocabulary* is the language you use to assign a robot tasks for a specific situation or scenario. And once a vocabulary has been established, figuring out the instructions for the robot to execute using that vocabulary is the next step.

Making a picture or a "visual representation" of the scenario and instructions you want the robot to perform can be great way to ensure your robot performs the tasks properly. A picture of the instructions the robot will perform allows you to think through the steps before translating them to the code. Visuals can help you understand the process, and studying that visual can improve its development by seeing what has to be done and elucidating that which may otherwise pose a problem. We call this the RSVP (Robot Scenario Visual Planning). The RSVP is a visual that helps develop the plan of instructions for what the robot will do. The RSVP is composed of three types of visuals:

- A floorplan of the physical environment of the scenario
- A statechart of the robot and object's states
- Flowcharts of the instructions for the tasks

These visuals ensure that you have a "clear picture" of what has to be done to program a robot to do great feats that can save the world or light the candles on a cake. RSVP can be used in any combination. Flowcharts may be more useful than statecharts for some. For others, statecharts are best. All we suggest is that a floorplan or layout is needed whether statecharts or flowcharts are utilized.

The saying "a picture is worth a thousand words" means that a single image can convey the meaning of a complex idea as well as a large amount of descriptive text. We grew up with this notion while in grade school especially when trying to solve word problems; "draw a picture" of the main ideas of the word problem and magically it becomes clear how to solve it. That notion still works. In this case, drawing a picture of the environment, a statechart, and flowcharts will be worth not only a thousand words but a thousand commands. Developing an RSVP allows you to plan your robot navigation through your scenario and work out the steps of the instructions for the tasks in the various situations. This avoids the trials and errors of directly writing code.

Mapping the Scenario

The first part of the RSVP is a map of the scenario. A map is a symbolic representation of the environment where the tasks and situations will take place. The environment for the scenario is the world in which the robots operate. Figure 3.1 shows the classic Test Pad for NXT Mindstorms robot.

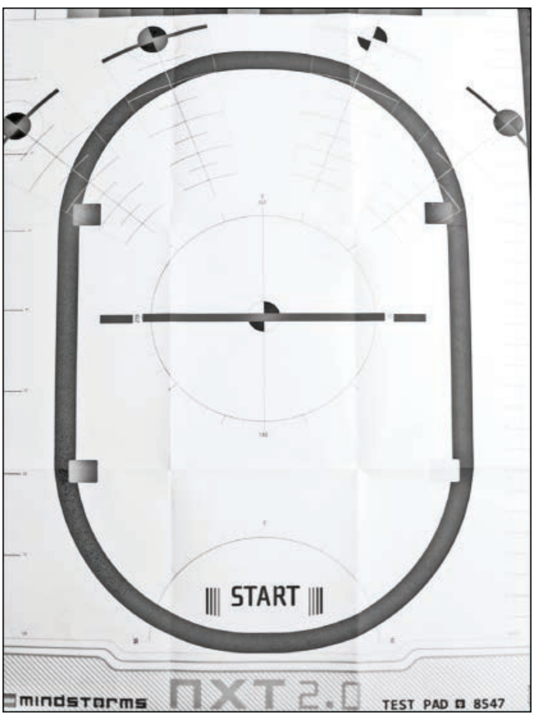


Figure 3.1
A robot world for NXT Mindstorms Test Pad

A Test Pad like the one shown in Figure 3.1 is part of the Mindstorms robot kits. This Test Pad is approximately 24 inches wide, 30 inches long, and has a rectangular shape. There are 16 colors on the Test Pad and 38 unique numbers with some duplicates. There is a series of straight lines and arcs on the pad. Yellow, blue, red, and green squares are on the Test Pad along with other colored shapes in various areas on the pad. It is the robot's world or environment used for the initial testing of NXT Mindstorms robots' color sensors, motors, and so on.

Like the Test Pad, a floorplan shows the locations of objects that are to be recognized like colored squares, objects the robot will interact with, or obstacles to be avoided. If objects are too high or too far away, sensors may not be able to determine their location. Determining the path the robot must navigate to reach those locations can also be planned by using this map.

The dimension of the space and of the robot (the robot footprint) may affect the capability of the robot to navigate the space and perform its tasks. For example, for our BR-1 robot, what is the location of the cake relative to the location of the robot? Is there a path? Are there obstacles? Can the robot move around the space? This is what the map helps determine.

tip

Next to the actual robot, the robot's environment is the most important consideration.

Creating a Floorplan

The map can be a simple 2D layout or floorplan of the environment using geometric shapes, icons, or colors to represent objects or robots. For a simple map of this kind, depicting an accurate scale is not that important, but objects and spaces should have some type of relative scale.

Use straight lines to delineate the area. Decide the measurement system. Be sure the measurement system is consistent with the API functions. Use arrows and the measurements to mark the dimensions of the area, objects, and robot footprint. It's best to use a vector graphics editor to create the map. For our maps we use Libre Office Draw. Figure 3.2 shows a simple layout of a floorplan of the robot environment for BR-1.

In Figure 3.2, the objects of interest are designated: locations of the robot, the table, and the cake on the table. The floorplan marks the dimensions of the area and the footprint of the robot. The lower-left corner is marked (0,0) and the upper-right corner is marked (300,400). This shows the dimensions of the area in cm. It also marks distances between objects and BR-1. Although this floorplan is not to scale, lengths and widths have a relative relationship. BR-1's footprint length is 50 cm and width is 30 cm.

BR-1 is to light the candles on the cake. The cake is located at the center of an area that is 400 cm × 300 cm. The cake has a diameter of 30 cm on a table that is 100 cm × 100 cm. That means the robot arm of BR-1 should have a reach of at least 53 cm from the edge of the table to reach the candle at the farthest point in the X dimension.

The maximum extension of the robot arm to the tip of the end-effector is 80 cm, and the length of the lighter adds an additional 10 cm. The task also depends on some additional considerations:

- The height of the candle
- The height of the cake
- The length of BR-1 from the arm point to the top of the candle wick
- The location of the robot

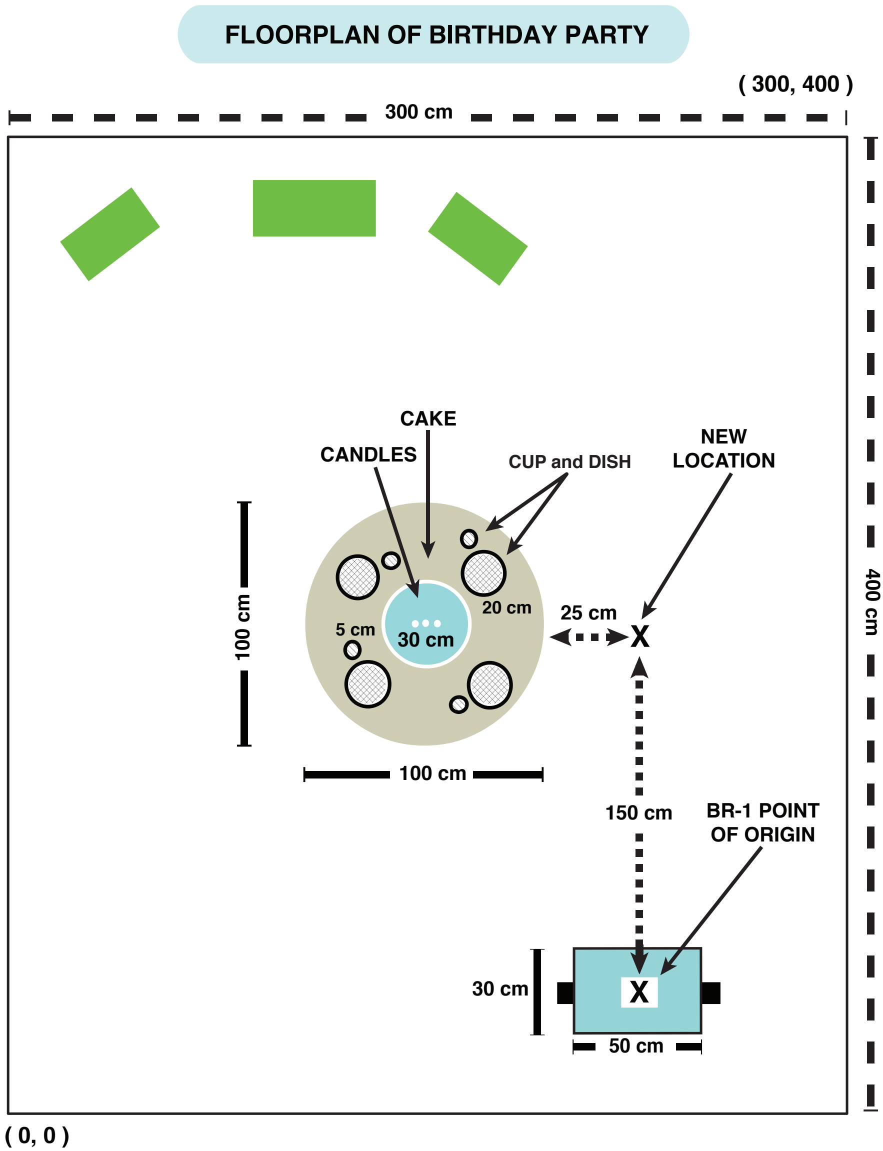


Figure 3.2
A layout of the floorplan for the BR-1 robot environment

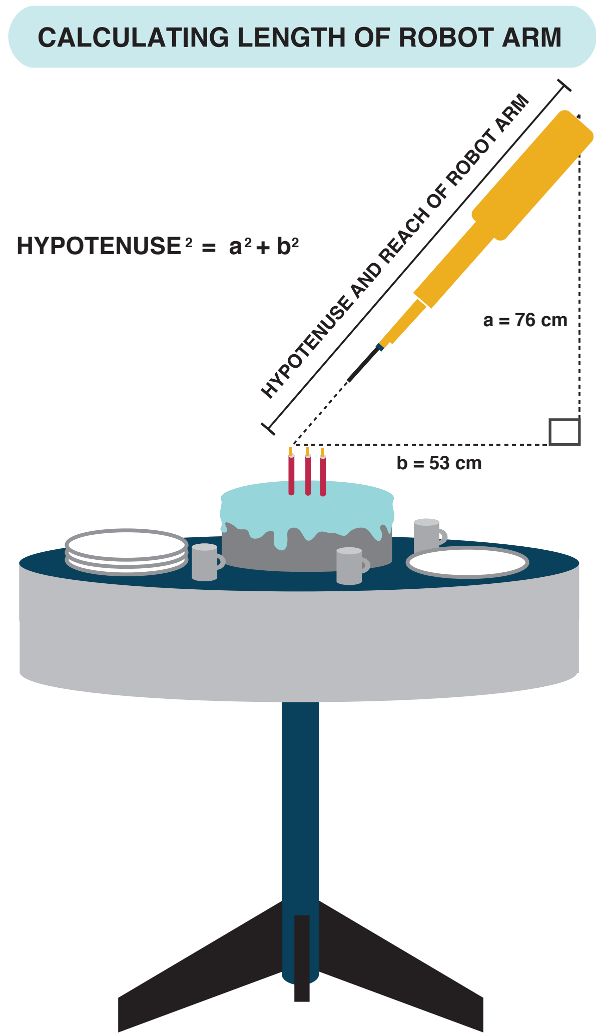
Figure 3.3 shows how to calculate the required reach to light the candle. In this case, it is the hypotenuse of a right triangle. Leg “a” of the triangle is the height of the robot from the top of the wick to the robot arm joint which is 76 cm, and leg “b” is the radius of the table plus the 3 cm to the location of the farthest candle on the cake, which is 53 cm.

So the required reach of the robot arm, end-effector, and lighter is around 93 cm. But the robot’s reach is only 90 cm. So BR-1 will have to lean a little toward the cake or get a lighter that is 3 cm longer to light the wick.

note

Determining the positions and required extension of a robot arm is far more complicated than this simple example and is discussed in Chapter 9, "Robot SPACES." But what is important in the example is how the layout/floorplan helps elucidate some important issues so that you can plan your robot's tasks.

Figure 3.3
Calculating the length of the robot arm as the hypotenuse of a right triangle



The Robot's World

For the robot to be automated it requires details about its environment. Consider this: If you are traveling to a new city you know nothing about, how well will you be able to do the things you want to do? You do not know where anything is. You need a map or someone to show you around and tell you "here is a restaurant" and "here is a museum." A robot that is *fully automated* must have sufficient information about the environment. The more information the robot has, the more likely the robot can accomplish its goal.

> **note**
> The robot's world is the environment where the robot performs it tasks. It's the only world the robot is aware of. Nothing outside that environment matters, and the robot is not aware of it.

All environments are not alike. We know environments are dynamic. The robot's environments can be partially or fully accessible to a robot. A fully accessible environment means all objects and aspects of the environment are within the reach of the robot's sensors. No object is too high, low, or far away from the robot to detect or interact with. The robot has all the necessary sensors to receive input from the environment. If there is a sound, the robot can detect it with its sound sensor. If a light is on, the robot can detect it with its light sensor.

A partially accessible environment means there are aspects of the environment the robot cannot detect or there are objects the robot cannot detect or interact with because it lacks the end-effector to pick it up or the location sensor to detect it. An object that is 180 cm from the ground is out of the reach of the robot with a 80 cm arm extension and a height of 50 cm. What if BR-1 is to light the candles once the singing begins and it does not have a sound sensor? Sound is part of the environment; therefore, it will not be able to perform the task. So when creating the floorplan for a partially accessible environment, consider the "robot's perspective." For example, for objects that are not accessible by the robot, use some visual indicator to distinguish those for the objects the robot can access. Use color or even draw a broken line around it.

Deterministic and Nondeterministic Environments

What about control? Does the robot control every aspect of its environment? Is the robot the only force that controls or manipulates the objects in its environment? This is the difference between a *deterministic* and *nondeterministic* environment.

With a deterministic environment, the next state is completely determined by the current state and the actions performed by the robot(s). This means if the BR-1 robot lights the candles, they will stay lit until BR-1 blows them out. If BR-1 removes the dishes from the table, they will stay in the location they're placed.

With a nondeterministic environment, like the one for the birthday party scenario, BR-1 does not blow out the candles. (It would be pretty mean if it did.) Dishes can be moved around by the attendees of the party, not just BR-1. What if there are no obstacles between BR-1 and its destination and then a partygoer places an obstacle there? How can BR-1 perform its tasks in a dynamic nondeterministic environment?

Each environment type has its own set of challenges. With a dynamic nondeterministic environment, the robot is required to consider the previous state and the current state before a task is attempted and then make a decision whether the task can be performed.

Table 3.1 lists some of the types of environments with a brief description.

Table 3.1 Some Types of Environments with a Brief Description

Environment Type	Description
Fully accessible	All aspects of the environment are accessible through the robot's sensors, actuators, and end-effectors.
Partially accessible	Some objects are not accessible or cannot be sensed by the robot.
Deterministic	The next state of the environment is completely determined by the current state and actions performed by the robot.
Nondeterministic	The next state of the environment is not completely under the control of the robot; the object may be influenced by outside factors or external agents.

RSVP READ SET

Many aspects of the environment are not part of the layout or floorplan but should be recorded somehow to be referenced when developing the instructions for the tasks. For example, the color, weight, height, and even surface type of the objects are all detectable characteristics that are identified by sensors or affect motors and end-effectors as well as the environment type, identified outside forces, and their impact on objects.

Some of these characteristics can be represented in the floorplan. But a READ set can contain all the characteristics. Each type of environment should have its own READ set.

For example, color is a detectable characteristic identified by a color or light sensor. The object's weight determines whether the robot can lift, hold, or carry the object to another location based on the torque of the servos. The shape, height, and even the surface determine whether the object can be manipulated by the end-effector.

> **note**
>
> A READ (Robot Environmental Attribute Description) set is a construct that contains a list of objects that the robot will encounter, control, and interact with within the robot's environment. It also contains characteristics and attributes of the objects detectable by the robot's sensors or that affect how the robot will interact with that object.

Any characteristic of the environment is part of the READ set, such as dimensions, lighting, and terrain. These characteristics can affect how well sensors and motors work. The lighting of the environment, whether sunlight, ambient room light, or candle light, affects the color and light sensor differently. A robot traveling across a wooden floor is different from the robot traveling across gravel, dirt, or carpet. Surfaces affect wheel rotation and distance calculations.

Table 3.2 is the READ set for the Mindstorms NXT Test Pad.

Table 3.2 READ Set for the Mindstorms NXT Test Pad

Object: Physical Work Space	
Attribute	**Value**
Environment type	Deterministic, fully accessible
Width	24 inches
Length	30 inches
Height	0
Shape	Rectangular
Surface	Paper (smooth)
Object: Color (Light)	
Attribute	**Value**
Num of colors	16
Light intensities	16
Colors	Red, green, blue, yellow, orange, white, black, gray, green, light blue, silver, etc.
Object: Symbols	
Attribute	**Value**
Symbol	Integers
Integer values	0–30, 90, 120, 180, 270, 360, 40, 60, 70
Geometric	Lines, arcs, squares

The READ set for the Test Pad describes the workspace including its type (fully accessible and deterministic), all the colors, and symbols. It describes what will be encountered by a robot when performing a search, such as identifying the blue square. The sets list the attributes and values of the physical workspace, colors, and symbols on the Test Pad.

For a dynamic environment such as our birthday party scenario, the READ set can contain information pertaining to the outside forces that might interact with the objects. For example, there are initial locations for the dishes and cups on the table, but the partygoers may move their dishes and cups to a new location on the table. The new locations should be represented in the READ set along with the time or the condition this occurred. Once the party is over and BR-1 is to remove those dishes and cup, each location should be updated. Table 3.3 is the READ set for the birthday party for the BR-1.

Table 3.3 READ Set for the Birthday Party Scenario

Attribute	Value	Force	Time/Condition	New Value
Object: Physical Work Space				
Environment type	Nondeterministic partially			
Width	300 cm			
Length	400 cm			
Height	0			
Shape	Rectangular			
Surface	Paper (smooth)			
Lighting	Artificial			
Object: Cake				
Attribute	Value	Force	Time/Condition	New Value
Height	14 cm			
Diameter	30 cm			
Location	150, 200	External	N/A	
Placement	Table	External	N/A	
Related objects	Candles			
Object: Candles				
Attribute	Value	Force	Time/Condition	New Value
Height	4 cm			
Number of candles	3			
Locations	1 153, 200 2 150, 200 3 147, 200	External	N/A	
Condition 1	Unlit	BR-1	Singing starts	Lit
Condition 2	Lit	External	Singing ends	Unlit
Object: Dishes				
Attribute	Value	Force	Time/Condition	New Value
Diameter	20 cm			
Height	1 cm			
Number of dishes	4			
Locations	1 110, 215 2 110, 180 3 170, 215 4 170, 180	External	After party ends	All at 110, 215 (stacked) Height 2 cm

Object: Cups				
Attribute	Value	Force	Time/Condition	New Value
Diameter	5 cm			
Height	10 cm			
Number of dishes	4			
Locations	1 119, 218 2 105, 189 3 165, 224 4 163, 185	External	After party ends	All at 119, 218 (stacked) Height 14 cm

This READ set has three additional columns:

- Force
- Time/Condition
- New Value

Force is the source of the iteration with the object; this force is anything working in the environment that is not the robot. The *Time/Condition* denotes when or under what condition the force interacts with the object. The *New Value* is self-explanatory.

Pseudocode and Flowcharting RSVP

Flowcharting is an RSVP used to work out the flow of control of an object to the whole system. It is a linear sequence of lines of instructions that can include any kind of looping, selection, or decision-making. A flowchart explains the process by using special box symbols that represent a certain type of work. Text displayed within the boxes describes a task, process, or instruction.

Flowcharts are a type of statechart (discussed later in this chapter) since they also contain states that are converted to actions and activities. Things like decisions and repetitions are easily represented, and what happens as the result of a branch can be simply depicted. Some suggest flowcharting before writing pseudocode. Pseudocode has the advantage of being easily converted to a programming language or utilized for documenting a program. It can also be easily changed. A flowchart requires a bit more work to change when using flowcharting software.

Table 3.4 list advantages and disadvantages of pseudocode and flowcharting. Both are great tools for working out the steps. It is a matter of personal taste which you will use at a particular time in a project.

Table 3.4 Advantages and Disadvantages of Pseudocode and Flowcharting

RSVP Type	Advantages	Disadvantages
Pseudocode: A method of describing computer instructions using a combination of natural language or programming language.	Easily created and modified in any word processor. Implementation is useful in any design. Written and understood easily. Easily converted to a programming language.	Is not visual. No standardized style or format. More difficult to follow the logic.
Flowcharting: Flow from the top to the bottom of a page. Each command is placed in a box of the appropriate shape, and arrows are used to direct program flow.	Is visual, easier to communicate to others. Problems can be analyzed more effectively.	Can become complex and clumsy for complicated logic. Alterations may require redrawing completely.

The four common symbols used in flowcharting are

- **Start and stop:** The start symbol represents the beginning of the flowchart with the label "start" appearing inside the symbol. The stop symbol represents the end of the flowchart with the label "stop" appearing inside the symbol. These are the only symbols with keyword labels.
- **Input and output:** The input and output symbol contains data that is used for input (e.g., provided by the user) and data that is the result of processing (output).
- **Decisions:** The decision symbol contains a question or a decision that has to be made.
- **Process:** The process symbol contains brief descriptions (a few words) of a rule or some action taking place.

Figure 3.4 shows the common symbols of flowcharting.

Each symbol has an inbound or outbound arrow leading to or from another symbol. The start symbol has only one outbound arrow, and the stop symbol has only one inbound arrow. The "start" symbol represents the beginning of the flowchart with the label "start" appearing inside the symbol.

The "stop" symbol represents the end of the flowchart with the label "stop" appearing inside the symbol. These are the only symbols with keyword labels. The decision symbol will contain a question or a decision that has to be made. The process symbol will contain brief descriptions (a few words) of a rule or some action taking place. The decision symbol has one inbound arrow and two outbound arrows. Each arrow represents a decision path through the process starting from that symbol:

- TRUE/YES
- FALSE/NO

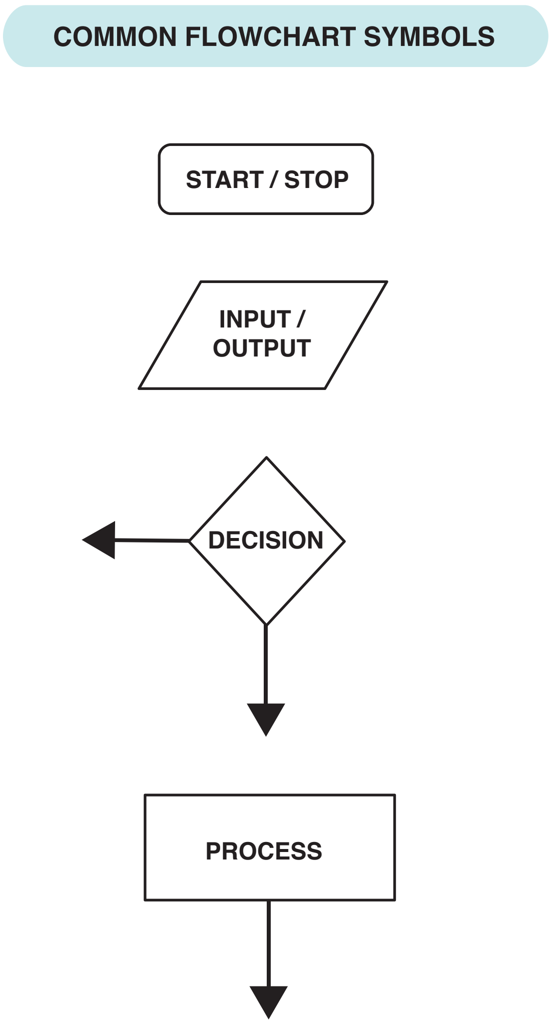
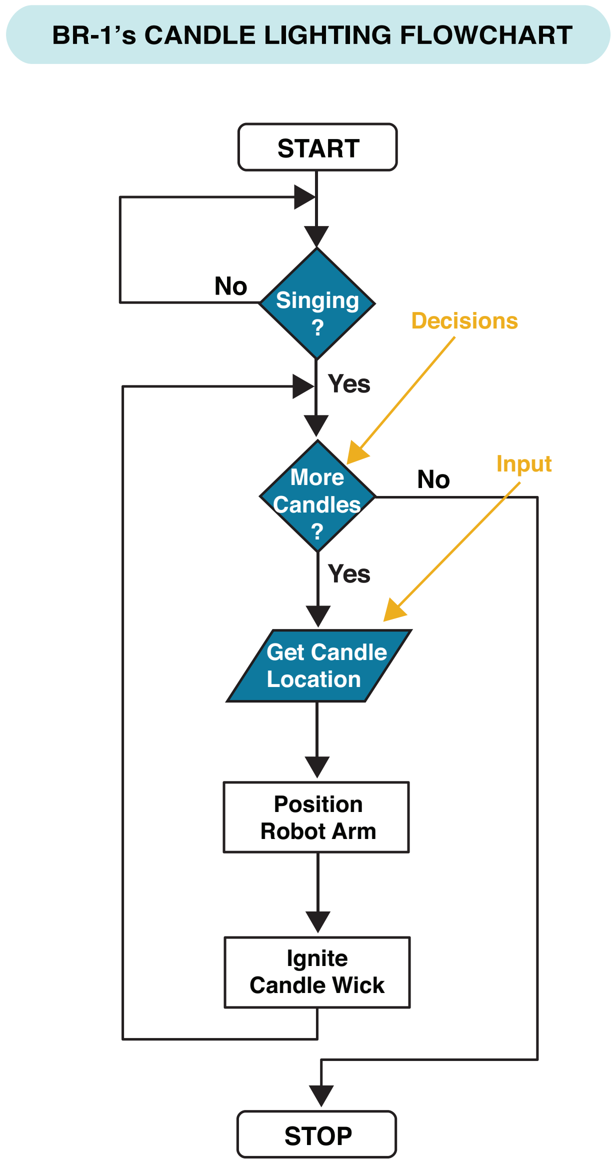


Figure 3.4
The common symbols of flowcharting

The process, input, and output symbols have one inbound and one outbound arrow. The symbols contain text that describes the rule or action, input or output. Figure 3.5 shows the "Lighting candles" flowchart.

Notice at the beginning of the flowchart, below the start symbol, BR-1 is to wait until the singing begins. A decision is made on whether the singing has started. There are two paths: If the singing has not started, there is a FALSE/NO answer to the question and BR-1 continues to wait. If the singing has started, there is a TRUE/YES answer and BR-1 enters a loop or decision.

Figure 3.5
The lighting candles flowchart



If there are candles to light, that is the decision. If yes, it gets the position of the next candle, positions the robot arm to the appropriate position to ignite the wick, and then ignites the wick. An input symbol is used to receive the position of the next candle to light. The BR-1 is to light all the candles and stops once complete.

Flow of Control and Control Structures

The task a robot executes can be a series of steps performed one after another, a sequential flow of control. The term *flow of control* details the direction the process takes, which way program control "flows." Flow of control determines how a computer responds when given certain conditions and parameters. An example of sequential flow of control is in Figure 3.6. Another robot in our birthday scenario is BR-3. Its task is to open the door for the guests. Figure 3.6 shows the sequential flow of control for this task.

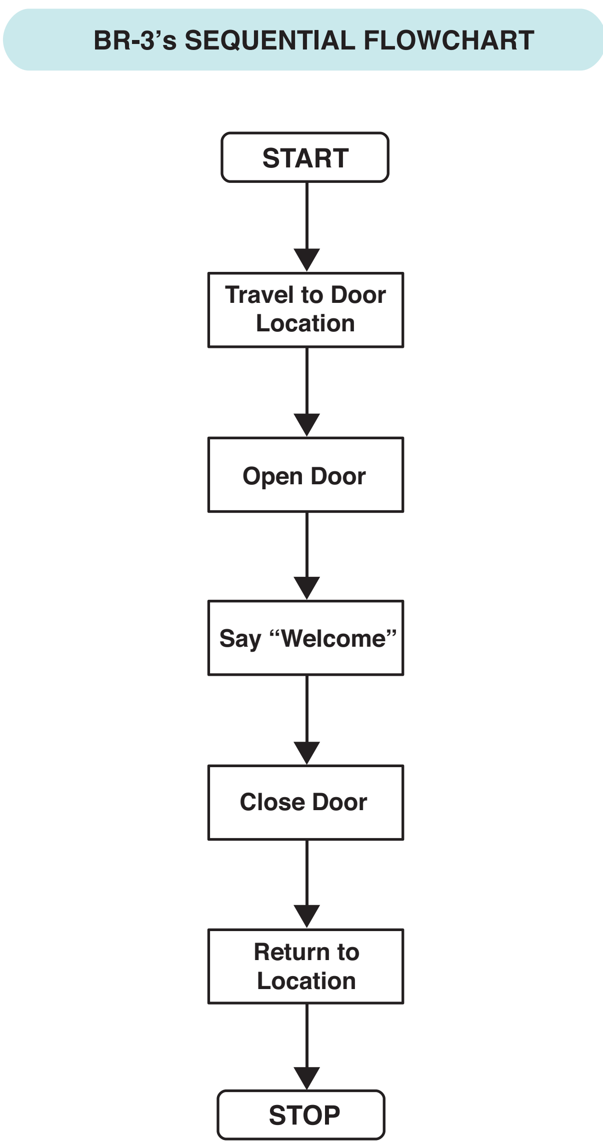


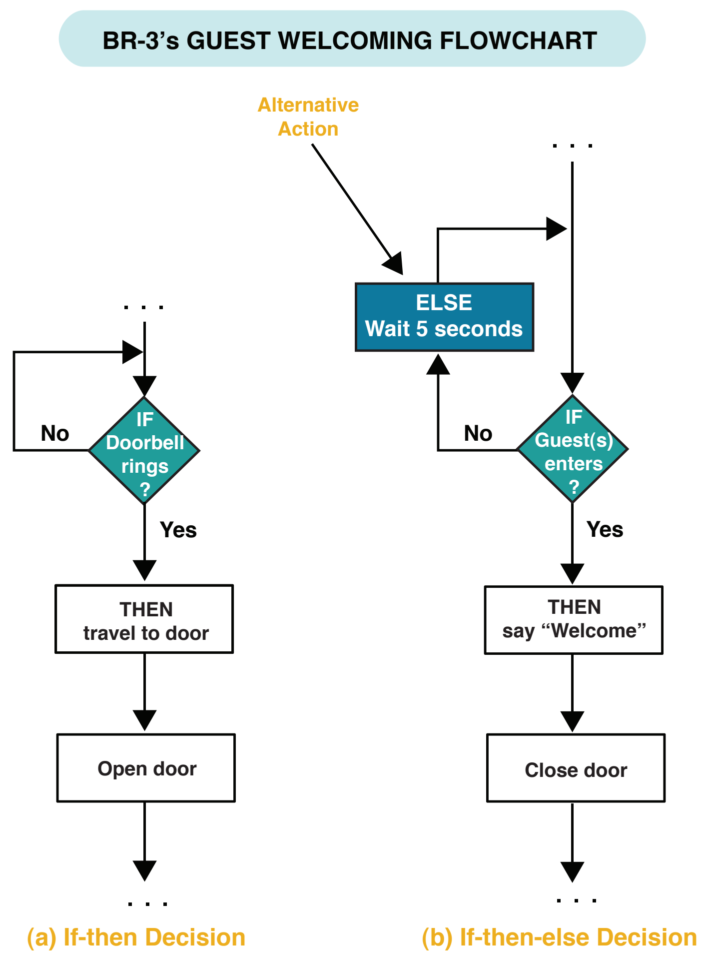
Figure 3.6
The flowchart for BR-3

The robot goes to the door, opens it, says, "Welcome," and then closes the door and returns to its original location. This would look like a rather inconsiderate host. Did the doorbell ring, signaling BR-3 that guests were at the door? If someone was at the door, after saying "Welcome," did BR-3 allow the guest to enter before closing the door? BR-3 should be able to act in a predictable way at the birthday party. That means making decisions based on events and doing things in repetition.

A decision symbol is used to construct branching for alternative flow controls. Decision symbols can be used to express decision, repetition, and case statements. A simple decision is structured as an if-then or if-then-else statement.

A simple if-then decision for BR-3 is shown in Figure 3.7 (a). "If Doorbell rings, then travel to door and open it." Now BR-3 will wait until the guest(s) enters before it says "Welcome." Notice the alternative action to be taken if the guest(s) has not entered. BR-3 will wait 5 seconds and then check if the guest(s) has entered yet. If Yes then BR-3 says "Welcome" and closes the door. This is shown in Figure 3.7 (b) if-then-else; the alternative action is to wait.

Figure 3.7
The flowchart for if-then and if-then-else decisions



In Figure 3.7, the question (or condition test) to be answered is whether the doorbell has rung. What if there is more than one question/condition test that has to be met before BR-3 is to open the door? With about BR-1, what if there were multiple conditions that had to be met before lighting the candles:

- "If there is a singing AND the lighter is lit then light the candles."
- In this case, both conditions have to be met. This is called a *Nested* decision or condition.

What if there is a question or condition in which there are many different possible answers and each answer or condition has a different action to take? For example, what if as our BR-1 or BR-3 travels across the room it encounters an object and has to maneuver around the object to reach its destination. It could check the range of the object in its path to determine the action to take to avoid it. If the object is within a certain range, BR-1 and BR-3 turn to the left either 90 degrees or 45 degrees, travel a path around the object, and then continue on their original path to their destinations as shown in Figure 3.8.

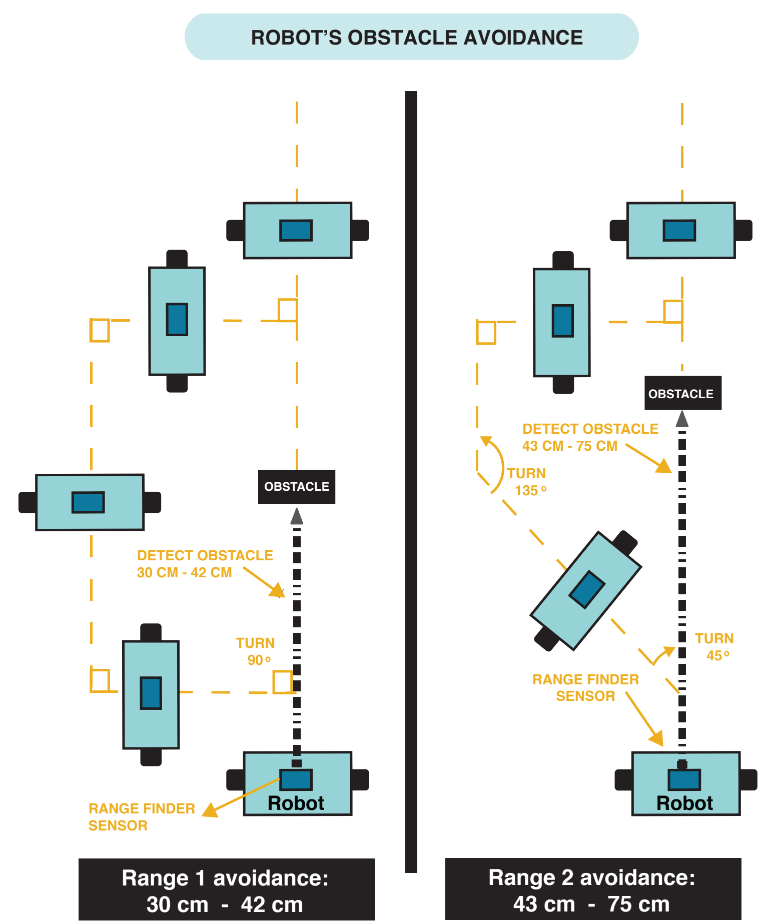
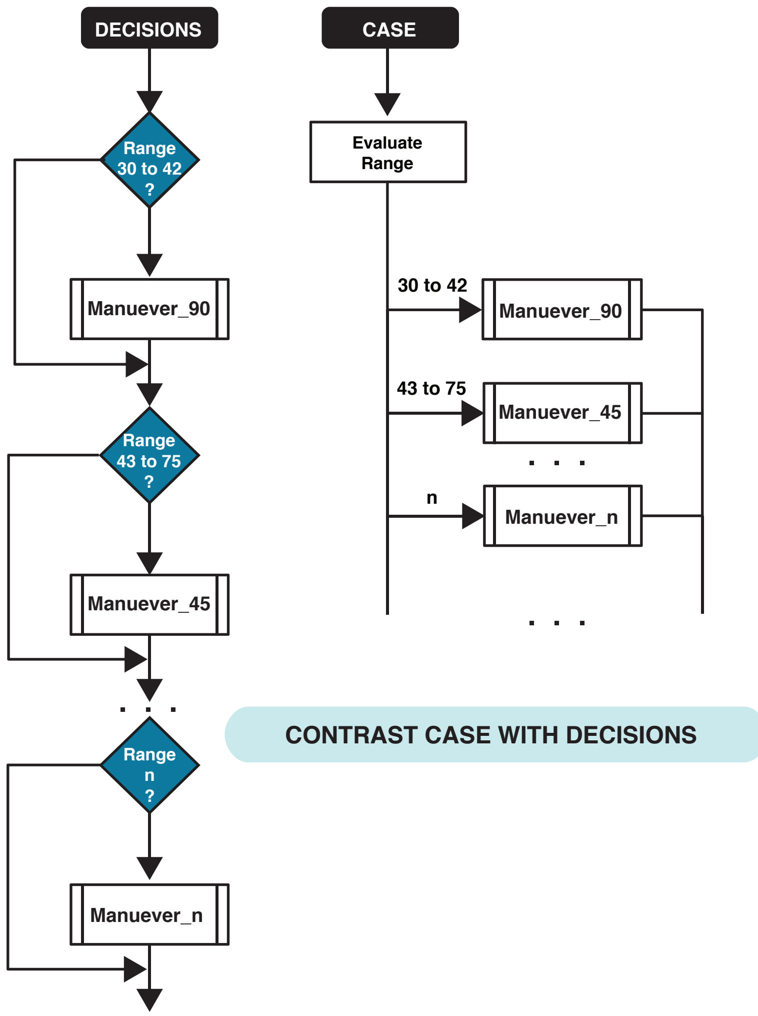


Figure 3.8
Robots obstacle avoidance

Using the flowchart, this can be expressed as a series of decisions or a case statement. A case is a type of decision where there are several possible answers to a question. With the series of decisions, the same question is asked three times, each with a different answer and action. With a case statement, the question is expressed only one time. Figure 3.9 contrasts the series of decisions in the case statement, which is simpler to read and understand what is going on.

Figure 3.9
Contrast case statement from a series of decisions



Repetition or looping is shown in Figure 3.10. In a loop, a simple decision is coupled with an action that is performed before or after the condition test. Depending on the result, the action is performed again. In Figure 3.10 (a), the action will be performed at least once. If the condition is not met (singing has not started—maybe everyone is having too much fun), the robot must continue to wait. This is an example of a do-until loop, "do" this action "until" this condition is true. A while loop performs the condition test first and if met, then the action is performed. This is depicted in Figure 3.10 (b), while singing has not started, wait. BR-1 will loop and wait until singing starts, as in the do-until loop. The difference is a wait is performed after the condition is met. Another type is the for loop, shown in Figure 3.10 (c), where the condition test controls the specific number of times the loop is executed.

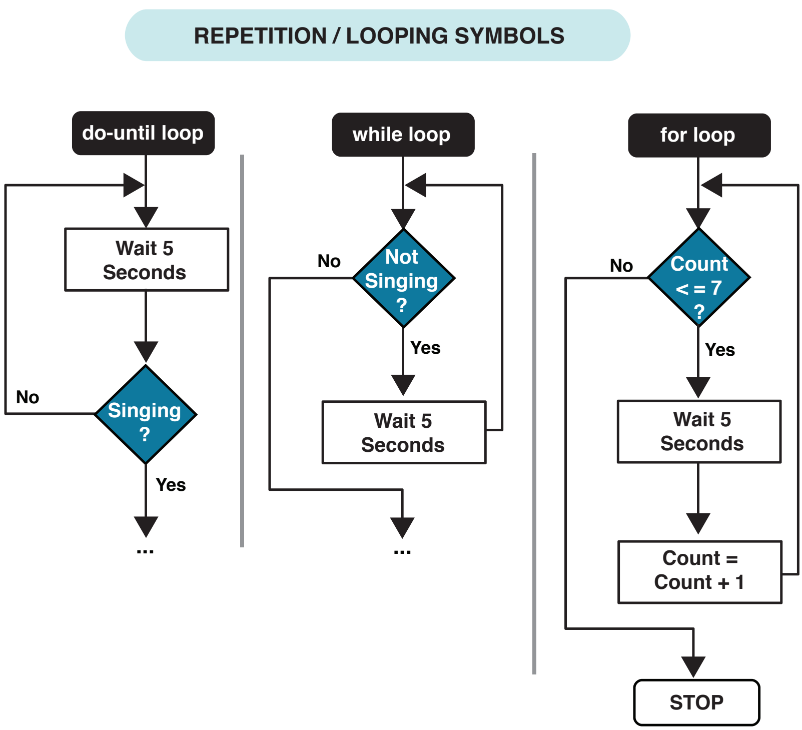


Figure 3.10
Repetition flowcharts for (a) do-until, (b) while, and (c) for loops

Subroutines

When thinking about what role your robot is to play in a scenario or situation, the role is broken down into a series of actions. BR-1's role is to be a host at a birthday party. This role is broken down into four states:

- Idle
- Traveling
- Lighting candles
- Waiting
- Removing dishes

This can be broken down into a series of actions or tasks:

1. Wait until singing begins.
   - Travel to birthday cake table.
   - Light the candles on the cake.
   - Travel to the original location.
2. Wait until party is over.
   - Remove dishes from cake table.
   - Travel back to original location.

These are short descriptions of tasks. Each task can be further broken down into a series of steps or subroutines. "Lighting candles" is a composite state that is broken down into other substates:

- Locating wick
- Igniting wick

Actually, "Remove dishes from cake table" and "Travel back to original location" should also be broken down into subroutines. Removing dishes from the cake table requires the positioning of the robot arm to remove each plate and cup subroutines, and traveling requires the rotating of motors subroutines.

Figure 3.11 shows the flowcharting for LightingCandles and its subroutines LocatingWick and IgnitingWick.

A subroutine symbol is the same as a process symbol, but it contains the name of the subroutine with a vertical line on each side of the name of the subroutine. The name of a subroutine can be a phrase that describes the purpose of the subroutine.

Flowcharts are then developed for those subroutines. What's great about using subroutines is the details don't have to be figured out immediately. Figuring out how the robot will perform a task can be put off for a while. The highest level processes can be worked out and then later actions/tasks can be broken down.

A subroutine can be identified and generalized from similar steps used at different place, in the robot's process. Instead of repeating a series of steps or developing different subroutines, the process can be generalized and placed in one subroutine that is called when needed. For example, the traveling procedure started out as a series of steps for BR-1 to travel to the cake table (TableTravel) and then a series of steps to travel back to its original location (OriginTravel). These are the same tasks with different starting and ending locations. Instead of subroutines that use the starting and ending locations, a Travel subroutine requires both the current and final locations of the robot to be used.

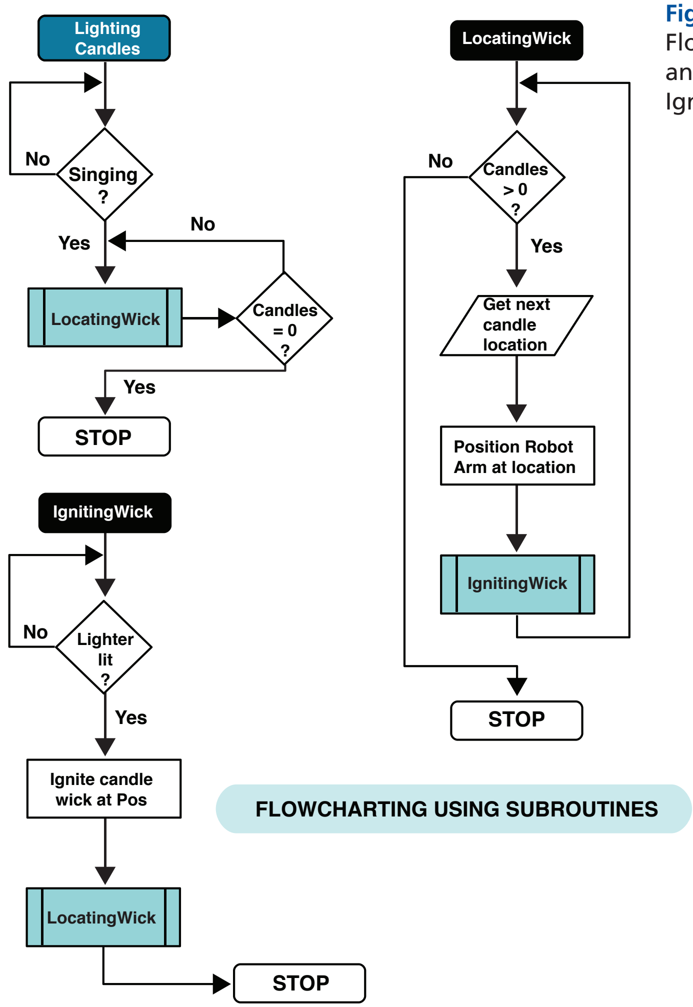


Figure 3.11
Flowcharting for LightingCandles and its subroutines LocatingWick and IgnitingWick

Statecharts for Robots and Objects

A statechart is one way to visualize a state machine.

For example, a "change of state" can be as simple as a change of location. When the robot travels from its initial location to the location next to the table, this is a change of the robot's state. Another example is that the birthday candles change from an unlit state to a lit state. The state machine captures the events, transformations, and responses. A statechart is a diagram of these activities. The statechart is used to capture the possible

note
A state machine models the behavior of a single robot or object in an environment. The states are the transformations the robot or object goes through when something happens.

situations for that object in that scenario. As you recall from Chapter 2, a situation is a snapshot of an event in the scenario. Possible situations for the BR-1 are

- **Situation 1:** BR-1 waiting for signal to move to new location
- **Situation 2:** BR-1 traveling to cake table
- **Situation 3:** BR-1 next to cake on a table with candles that have not yet been lit
- **Situation 4:** BR-1 positioning the lighter over the candles, and so on

All these situations represent changes in the state of the robot. Changes in the state of the robot or object take place when something happens, an event. That event can be a signal, the result of an operation, or just the passing of time. When an event happens, some activity happens, depending on the current state of the object. The current state determines what is possible.

The event works as a trigger or stimulus causing a condition in which a change of state can occur. This change from one state to another is called a transition. The object is transitioning from stateA, the source state, to stateB, the target state. Figure 3.12 shows a simple state machine for BR-1.

Figure 3.12
State machine for BR-1

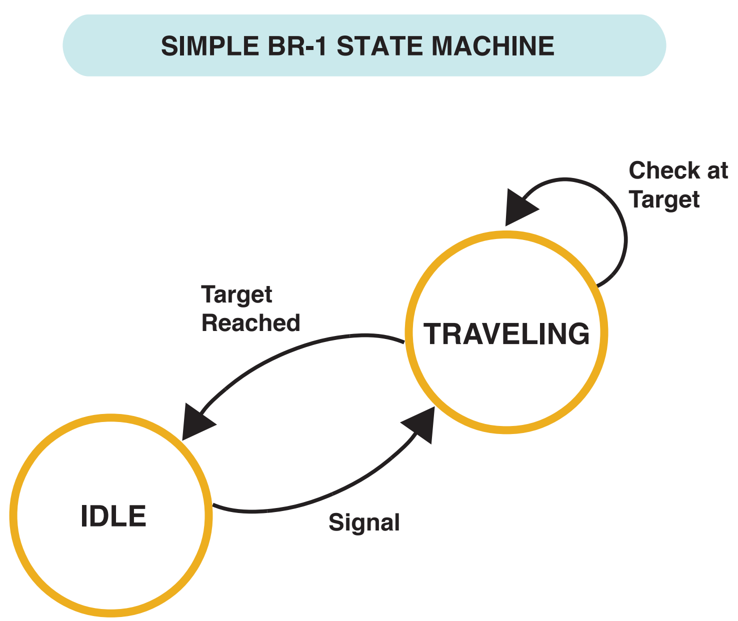


Figure 3.12 shows two states for BR-1: Idle or Traveling. When BR-1 is in an Idle state, it is waiting for an event to occur. This event is a signal that contains the new location for the robot. Once the robot receives this signal, it transitions from Idle to Traveling state. BR-1 continues to travel until it reaches its target location. Once reached, the robot transitions from the Traveling state back to an Idle state. Signals, actions, and activities may be performed or controlled by the object or by outside forces. For example, the new location will not be generated by BR-1 but by another agent. BR-1 does have the capability to check its location while traveling.

Developing a Statechart

As discussed earlier, a state is condition or situation of an object that represents a transformation during the life of the object. A state machine shows states and transitions between states. There are many ways to represent a state machine. In this book, we represent a state machine as a UML (Unified Modeling Language) statechart. Statecharts have additional notations for events, actions, conditions, parts of transitions, and types and parts of states.

> **note**
> In a statechart, the nodes are states and the arcs are transitions. The states are represented as circles or as rounded-corner rectangles in which the name of the state is displayed. The transitions are arcs that connect the source and the target state with an arrow pointing to the target state.

There are three types of states:

- **Initial:** The default starting point for the state machine. It is a solid black dot with a transition to the first state of the machine.
- **Final:** The ending state, meaning the object has reached the end of its lifetime. It is represented as a solid dot inside a circle.
- **Composite state and substate:** A state contained inside another state. That state is called a superstate or composite state.

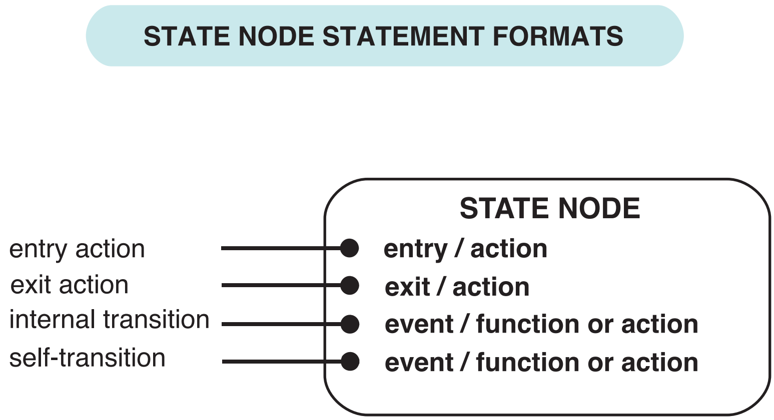
States have different parts. Table 3.5 lists the parts of states with a brief description. A state node displaying its name can also display the parts listed in this table. These parts can be used to represent processing that occurs when the object transitions to the new state. There may be actions to take as soon as the object enters and leaves the state. There may be actions that have to be taken while the object is in a particular state. All this can be noted in the statechart.

Table 3.5 Parts of a State

Part	Description
Name	The unique name of the state distinguishes it from other states.
Entry/exit actions	Actions executed when entering the state (entry actions) or executed when exiting the state (exit actions).
Composite/substates	A nested state; the substates are the states that are activated inside the composite state.
Internal transitions	Transitions that occur within the state are handled without causing a change in the state but do not cause the entry and exit actions to execute.
Self-transitions	Transitions that occur within the state that are handled without causing a change in the state. They cause the exit and entry actions to execute when exiting and then reentering the state.

Figure 3.13 shows a state node and format for actions, activities, and internal transition statements.

Figure 3.13
A state node and the format of statements



The entry and exit action statements have this format:

- Entry/action or activity
- Exit/action or activity

This is an example of an entry and exit action statement for a state called Validating:

- entry action: entry / validate(data)
- exit action: exit / send(data)

Upon entering the Validating state, the validate(data) function is called. Upon exiting this state, the exit action send(data) is called.

Internal transitions occur inside the state. They are events that take place after entry actions and before exit actions if there are any. Self-transitions are different from internal transitions. With a self-transition, the entry and exit actions are performed. The state is left; the exit action is performed.

Then the same state is reentered and the entry action is performed. The action of the self-transition is performed after the exit action and before the entry action. Self-transitions are represented as a directed line that loops and points back to the same state.

An internal or self-transition statement has this format:

- Name/action or function

For example:

- do / createChart(data)

“do” is the label for the activity, the function “createChart(data)” is executed.

There are several parts of a transition, the relationship between two states. We know that triggers cause transitions to occur, and actions can be coupled with triggers. A met condition can also cause a transition. Table 3.6 lists the parts of a transition.

Table 3.6 Parts of a Transition

Part	Description
Source state	The original state of the object; when a transition occurs the object leaves the source state.
Target state	The state the object enters after the transition.
Event trigger	The event that causes the transition to occur. A transition may be triggerless, which means the transition occurs as soon as the object completes all activities in the source state.
Guard condition	A boolean expression associated with an event trigger, which when evaluated to TRUE, causes the transition to take place.
Action	An action executed by the object that takes place during a transition; may be associated with an event trigger or guard condition.

An event trigger has a similar format as a state action statement:

- Name/action or function
- name [Guard] / action or function

For example, for the internal transition statement, a guarded condition can be added:

- do [Validated] / createChart(data)

Figure 3.14 is the statechart for BR-1.

There are four states: Idle, Traveling, Lighting candles, Waiting and Removing dishes. When transitioning from Idle to Traveling, BR-1 gets the new location and knows its mission:

- do [GetPosition] / setMission()

There are two transitions from the Traveling state:

- Traveling to Lighting candles
- Traveling to Removing dishes

> **note**
> A guard condition is a boolean value or expression that evaluates to True or False. It is enclosed in square brackets. The guard condition has to be met for the function to execute. It can be used in a state or transition statement.

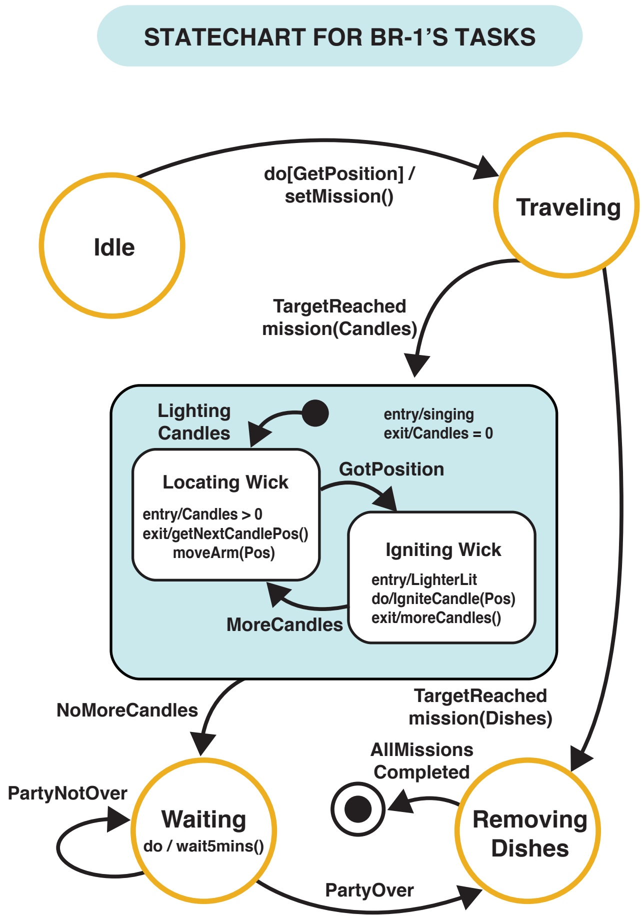
> **note**
> Validated is a boolean value. It is a condition that has to be met for createChart() to execute.

Traveling transitions to LightingCandles when its target is reached and its Mission is *candles*. Traveling transitions to Removing dishes when its target is reached and its Mission is *dishes*. To transition from LightingCandles, "candles" mission must be complete. To transition from RemovingDishes to the final state, all missions must be completed.

LightingCandles is a composite state that contains two substates: LocatingWick and IgnitingWick. Upon entering the Lighting candles state, the boolean value Singing is evaluated. If there is singing, then the candles are to be lit. First the wick has to be located, then the arm is moved to that location, and finally the wick can be lit. In the LocatingWick state, the entry action evaluates the expression:

- Candles > 0

Figure 3.14
Statechart for BR-1



If True, the state exits when the position of the first or next candle is retrieved and then the robot arm is moved to the position (Pos).

The position of the wick is retrieved, so BR-1 transitions to "IgnitingWick." Upon entry, the lighter is checked to see if it is lit. If lit the candle wick is lit, (an internal state). To exit this state, Candles > 0, then "LocatingWick" state is reentered. If Candles = 0, then BR-1 transitions to "Waiting" state. BR-1 waits until the party is over. Then BR-1 can remove all the dishes. In the "Waiting" state, there is a self-transition "PartyNotOver." Remember, with a self-transition, the exit and entry actions are performed as the state is exited and reentered. In this case, there are no exit actions, but there is an entry action "wait 5 minutes". The guard condition is checked, "PartyNotOver." If the party is not over, then the state is reentered and the entry action is executed; BR-1 waits for 5 minutes. Once the party is over, then BR-1 transitions to "RemovingDishes." This is the last state. If boolean value

AllMissionsCompleted is True, BR-1 transitions to the final state. But some objects may not have a final state and work continuously. Statecharts are good for working out the changing aspect of an object during the lifetime of the object. It shows the flow of control from one state of the object to another state.

What's Ahead?

In Chapter 4, "Checking the Actual Capabilities of Your Robot," we discuss what your robot is capable of doing. This means checking the capabilities and limits of the microcontroller, sensors, motors, and end-effectors of the robots.

INDEX

A

B

C

D

E

F

G

H

I

J-L

M

N

O

P

Q-R

S

T

U

V

W-X-Y-Z