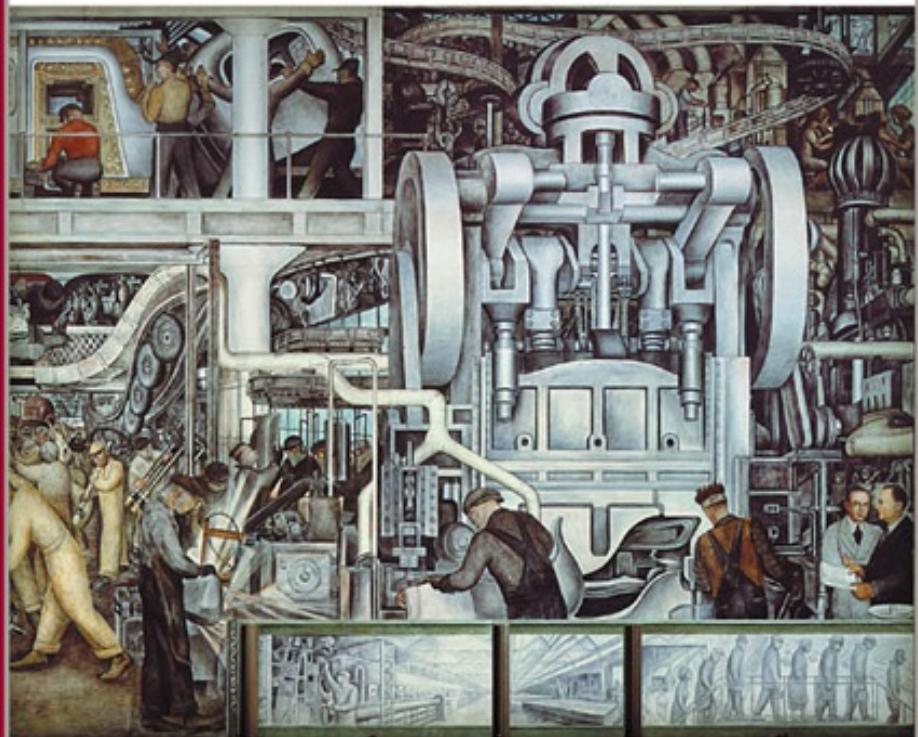


INTRODUCTION TO
AI ROBOTICS

SECOND EDITION



ROBIN R. MURPHY

Intelligent Robotics and Autonomous Agents

Edited by Ronald C. Arkin

A complete list of the books in the Intelligent Robotics and Autonomous Agents series appears at the back of this book.

Introduction to AI Robotics

Second Edition

Robin R. Murphy

The MIT Press
Cambridge, Massachusetts
London, England

© 2019 Massachusetts Institute of Technology

All rights reserved. No part of this book may be reproduced in any form by any electronic or mechanical means (including photocopying, recording, or information storage and retrieval) without permission in writing from the publisher.

This book was typeset by the author using $\text{\LaTeX} 2\epsilon$.

Printed and bound in the United States of America.

Library of Congress Cataloging-in-Publication Data

Names: Murphy, Robin, 1957- author.

Title: Introduction to AI robotics / Robin R. Murphy.

Description: Second edition. | Cambridge, MA: The MIT Press, 2019. | Includes bibliographical references and index.

Identifiers: LCCN 2017059414 | ISBN 9780262038485 (hardcover: alk. paper)

Subjects: LCSH: Robotics. | Artificial intelligence.

Classification: LCC TJ211.M865 2018 | DDC 629.8/9563—dc23 LC record available at <https://lccn.loc.gov/2017059414>

d_r0

To Kevin

...and Carlyle Ramsey, Monroe Swilley, Chris Trowell

Brief Contents

I Framework for Thinking About AI and Robotics

- 1 What Are Intelligent Robots?*
- 2 A Brief History of AI Robotics*
- 3 Automation and Autonomy*
- 4 Software Organization of Autonomy*
- 5 Telesystems*

II Reactive Functionality

- 6 Behaviors*
- 7 Perception and Behaviors*
- 8 Behavioral Coordination*
- 9 Locomotion*
- 10 Sensors and Sensing*
- 11 Range Sensing*

III Deliberative Functionality

- 12 Deliberation*
- 13 Navigation*
- 14 Metric Path Planning and Motion Planning*
- 15 Localization, Mapping, and Exploration*
- 16 Learning*

IV Interactive Functionality

17 MultiRobot Systems (MRS)

18 Human-Robot Interaction

V Design and the Ethics of Building Intelligent Robots

19 Designing and Evaluating Autonomous Systems

20 Ethics

Contents

I Framework for Thinking About AI and Robotics

1 What Are Intelligent Robots?

- 1.1 Overview
- 1.2 Definition: What Is an Intelligent Robot?
- 1.3 What Are the Components of a Robot?
- 1.4 Three Modalities: What Are the Kinds of Robots?
- 1.5 Motivation: Why Robots?
- 1.6 Seven Areas of AI: Why Intelligence?
- 1.7 Summary
- 1.8 Exercises
- 1.9 End Notes

2 A Brief History of AI Robotics

- 2.1 Overview
- 2.2 Robots as Tools, Agents, or Joint Cognitive Systems
- 2.3 World War II and the Nuclear Industry
- 2.4 Industrial Manipulators
- 2.5 Mobile Robots
- 2.6 Drones
- 2.7 The Move to Joint Cognitive Systems
- 2.8 Summary
- 2.9 Exercises
- 2.10 End Notes

3 Automation and Autonomy

- 3.1 Overview
- 3.2 The Four Sliders of Autonomous Capabilities
 - 3.2.1 Plans: Generation versus Execution
 - 3.2.2 Actions: Deterministic versus Non-deterministic
 - 3.2.3 Models: Open- versus Closed-World
 - 3.2.4 Knowledge Representation: Symbols versus Signals

- 3.3 Bounded Rationality
- 3.4 Impact of Automation and Autonomy
- 3.5 Impact on Programming Style
- 3.6 Impact on Hardware Design
- 3.7 Impact on Types of Functional Failures
 - 3.7.1 Functional Failures
 - 3.7.2 Impact on Types of Human Error
- 3.8 Trade-Spaces in Adding Autonomous Capabilities
- 3.9 Summary
- 3.10 Exercises
- 3.11 End Notes

4 *Software Organization of Autonomy*

- 4.1 Overview
- 4.2 The Three Types of Software Architectures
 - 4.2.1 Types of Architectures
 - 4.2.2 Architectures Reinforce Good Software Engineering Principles
- 4.3 Canonical AI Robotics Operational Architecture
 - 4.3.1 Attributes for Describing Layers
 - 4.3.2 The Reactive Layer
 - 4.3.3 The Deliberative Layer
 - 4.3.4 The Interactive Layer
 - 4.3.5 Canonical Operational Architecture Diagram
- 4.4 Other Operational Architectures
 - 4.4.1 Levels of Automation
 - 4.4.2 Autonomous Control Levels (ACL)
 - 4.4.3 Levels of Initiative
- 4.5 Five Subsystems in Systems Architectures
- 4.6 Three Systems Architecture Paradigms
 - 4.6.1 Trait 1: Interaction Between Primitives
 - 4.6.2 Trait 2: Sensing Route
 - 4.6.3 Hierarchical Systems Architecture Paradigm
 - 4.6.4 Reactive Systems Paradigm
 - 4.6.5 Hybrid Deliberative/Reactive Systems Paradigm
- 4.7 Execution Approval and Task Execution
- 4.8 Summary
- 4.9 Exercises
- 4.10 End Notes

5 *Telesystems*

- 5.1 Overview
- 5.2 Taskable Agency versus Remote Presence

- 5.3 The Seven Components of a Telesystem
- 5.4 Human Supervisory Control
 - 5.4.1 Types of Supervisory Control
 - 5.4.2 Human Supervisory Control for Telesystems
 - 5.4.3 Manual Control
 - 5.4.4 Traded Control
 - 5.4.5 Shared Control
 - 5.4.6 Guarded Motion
- 5.5 Human Factors
 - 5.5.1 Cognitive Fatigue
 - 5.5.2 Latency
 - 5.5.3 Human: Robot Ratio
 - 5.5.4 Human Out-of-the-Loop Control Problem
- 5.6 Guidelines for Determining if a Telesystem Is Suitable for an Application
 - 5.6.1 Examples of Telesystems
- 5.7 Summary
- 5.8 Exercises
- 5.9 End Notes

II Reactive Functionality

6 Behaviors

- 6.1 Overview
- 6.2 Motivation for Exploring Animal Behaviors
- 6.3 Agency and Marr's Computational Theory
- 6.4 Example of Computational Theory: Rana Computatrix
- 6.5 Animal Behaviors
 - 6.5.1 Reflexive Behaviors
- 6.6 Schema Theory
 - 6.6.1 Schemas as Objects
 - 6.6.2 Behaviors and Schema Theory
 - 6.6.3 S-R: Schema Notation
- 6.7 Summary
- 6.8 Exercises
- 6.9 End Notes

7 Perception and Behaviors

- 7.1 Overview
- 7.2 Action-Perception Cycle
- 7.3 Gibson: Ecological Approach
 - 7.3.1 Optic Flow
 - 7.3.2 Nonvisual Affordances

- 7.4 Two Perceptual Systems
- 7.5 Innate Releasing Mechanisms
 - 7.5.1 Definition of Innate Releasing Mechanisms
 - 7.5.2 Concurrent Behaviors
- 7.6 Two Functions of Perception
- 7.7 Example: Cockroach Hiding
 - 7.7.1 Decomposition
 - 7.7.2 Identifying Releasers
 - 7.7.3 Implicit versus Explicit Sequencing
 - 7.7.4 Perception
 - 7.7.5 Architectural Considerations
- 7.8 Summary
- 7.9 Exercises
- 7.10 End Notes

8 *Behavioral Coordination*

- 8.1 Overview
- 8.2 Coordination Function
- 8.3 Cooperating Methods: Potential Fields
 - 8.3.1 Visualizing Potential Fields
 - 8.3.2 Magnitude Profiles
 - 8.3.3 Potential Fields and Perception
 - 8.3.4 Programming a Single Potential Field
 - 8.3.5 Combination of Fields and Behaviors
 - 8.3.6 Example Using One Behavior per Sensor
 - 8.3.7 Advantages and Disadvantages
- 8.4 Competing Methods: Subsumption
 - 8.4.1 Example
- 8.5 Sequences: Finite State Automata
 - 8.5.1 A Follow the Road FSA
 - 8.5.2 A Pick Up the Trash FSA
- 8.6 Sequences: Scripts
- 8.7 AI and Behavior Coordination
- 8.8 Summary
- 8.9 Exercises
- 8.10 End Notes

9 *Locomotion*

- 9.1 Overview
- 9.2 Mechanical Locomotion
 - 9.2.1 Holonomic versus Nonholonomic
 - 9.2.2 Steering

- 9.3 Biomimetic Locomotion
- 9.4 Legged Locomotion
 - 9.4.1 Number of Leg Events
 - 9.4.2 Balance
 - 9.4.3 Gaits
 - 9.4.4 Legs with Joints
- 9.5 Action Selection
- 9.6 Summary
- 9.7 Exercises
- 9.8 End Notes

10 *Sensors and Sensing*

- 10.1 Overview
- 10.2 Sensor and Sensing Model
 - 10.2.1 Sensors: Active or Passive
 - 10.2.2 Sensors: Types of Output and Usage
- 10.3 Odometry, Inertial Navigation System (INS) and Global Positioning System (GPS)
- 10.4 Proximity Sensors
- 10.5 Computer Vision
 - 10.5.1 Computer Vision Definition
 - 10.5.2 Grayscale and Color Representation
 - 10.5.3 Region Segmentation
 - 10.5.4 Color Histogramming
- 10.6 Choosing Sensors and Sensing
 - 10.6.1 Logical Sensors
 - 10.6.2 Behavioral Sensor Fusion
 - 10.6.3 Designing a Sensor Suite
- 10.7 Summary
- 10.8 Exercises
- 10.9 End Notes

11 *Range Sensing*

- 11.1 Overview
- 11.2 Stereo
- 11.3 Depth from X
- 11.4 Sonar or Ultrasonics
 - 11.4.1 Light Stripers
 - 11.4.2 Lidar
 - 11.4.3 RGB-D Cameras
 - 11.4.4 Point Clouds
- 11.5 Case Study: Hors d’Oeuvres, Anyone?
- 11.6 Summary

- 11.7 Exercises
- 11.8 End Notes

III Deliberative Functionality

12 *Deliberation*

- 12.1 Overview
- 12.2 Strips
 - 12.2.1 More Realistic Strips Example
 - 12.2.2 Strips Summary
 - 12.2.3 Revisiting the Closed-World Assumption and the Frame Problem
- 12.3 Symbol Grounding Problem
- 12.4 Global World Models
 - 12.4.1 Local Perceptual Spaces
 - 12.4.2 Multi-level or Hierarchical World Models
 - 12.4.3 Virtual Sensors
 - 12.4.4 Global World Model and Deliberation
- 12.5 Nested Hierarchical Controller
- 12.6 RAPS and 3T
- 12.7 Fault Detection Identification and Recovery
- 12.8 Programming Considerations
- 12.9 Summary
- 12.10 Exercises
- 12.11 End Notes

13 *Navigation*

- 13.1 Overview
- 13.2 The Four Questions of Navigation
- 13.3 Spatial Memory
- 13.4 Types of Path Planning
- 13.5 Landmarks and Gateways
- 13.6 Relational Methods
 - 13.6.1 Distinctive Places
 - 13.6.2 Advantages and Disadvantages
- 13.7 Associative Methods
- 13.8 Case Study of Topological Navigation with a Hybrid Architecture
 - 13.8.1 Topological Path Planning
 - 13.8.2 Navigation Scripts
 - 13.8.3 Lessons Learned
- 13.9 Discussion of Opportunities for AI
- 13.10 Summary
- 13.11 Exercises

13.12 End Notes

14 Metric Path Planning and Motion Planning

14.1 Overview

14.2 Four Situations Where Topological Navigation Is Not Sufficient

14.3 Configuration Space

14.3.1 Meadow Maps

14.3.2 Generalized Voronoi Graphs

14.3.3 Regular Grids

14.3.4 Quadtrees

14.4 Metric Path Planning

14.4.1 A* and Graph-Based Planners

14.4.2 Wavefront-Based Planners

14.5 Executing a Planned Path

14.5.1 Subgoal Obsession

14.5.2 Replanning

14.6 Motion Planning

14.7 Criteria for Evaluating Path and Motion Planners

14.8 Summary

14.9 Exercises

14.10 End Notes

15 Localization, Mapping, and Exploration

15.1 Overview

15.2 Localization

15.3 Feature-Based Localization

15.4 Iconic Localization

15.5 Static versus Dynamic Environments

15.6 Simultaneous Localization and Mapping

15.7 Terrain Identification and Mapping

15.7.1 Digital Terrain Elevation Maps

15.7.2 Terrain Identification

15.7.3 Stereophotogrammetry

15.8 Scale and Traversability

15.8.1 Scale

15.8.2 Traversability Attributes

15.9 Exploration

15.9.1 Reactive Exploration

15.9.2 Frontier-Based Exploration

15.9.3 Generalized Voronoi Graph Methods

15.10 Localization, Mapping, Exploration, and AI

15.11 Summary

- 15.12 Exercises
- 15.13 End Notes

16 Learning

- 16.1 Overview
- 16.2 Learning
- 16.3 Types of Learning by Example
- 16.4 Common Supervised Learning Algorithms
 - 16.4.1 Induction
 - 16.4.2 Support Vector Machines
 - 16.4.3 Decision Trees
- 16.5 Common Unsupervised Learning Algorithms
 - 16.5.1 Clustering
 - 16.5.2 Artificial Neural Networks
- 16.6 Reinforcement Learning
 - 16.6.1 Utility Functions
 - 16.6.2 Q-learning
 - 16.6.3 Q-learning Example
 - 16.6.4 Q-learning Discussion
- 16.7 Evolutionary Robotics and Genetic Algorithms
- 16.8 Learning and Architecture
- 16.9 Gaps and Opportunities
- 16.10 Summary
- 16.11 Exercises
- 16.12 End Notes

IV Interactive Functionality

17 MultiRobot Systems (MRS)

- 17.1 Overview
- 17.2 Four Opportunities and Seven Challenges
 - 17.2.1 Four Advantages of MRS
 - 17.2.2 Seven Challenges in MRS
- 17.3 Multirobot Systems and AI
- 17.4 Designing MRS for Tasks
 - 17.4.1 Time Expectations for a Task
 - 17.4.2 Subject of Action
 - 17.4.3 Movement
 - 17.4.4 Dependency
- 17.5 Coordination Dimension of MRS Design
- 17.6 Systems Dimensions in Design
 - 17.6.1 Communication

- 17.6.2 MRS Composition
- 17.6.3 Team Size
- 17.7 Five Most Common Occurrences of MRS
- 17.8 Operational Architectures for MRS
- 17.9 Task Allocation
- 17.10 Summary
- 17.11 Exercises
- 17.12 End Notes

18 *Human-Robot Interaction*

- 18.1 Overview
- 18.2 Taxonomy of Interaction
- 18.3 Contributions from HCI, Psychology, Communications
 - 18.3.1 Human-Computer Interaction
 - 18.3.2 Psychology
 - 18.3.3 Communications
- 18.4 User Interfaces
 - 18.4.1 Eight Golden Rules for User Interface Design
 - 18.4.2 Situation Awareness
 - 18.4.3 Multiple Users
- 18.5 Modeling Domains, Users, and Interactions
 - 18.5.1 Motivating Example of Users and Interactions
 - 18.5.2 Cognitive Task Analysis
 - 18.5.3 Cognitive Work Analysis
- 18.6 Natural Language and Naturalistic User Interfaces
 - 18.6.1 Natural Language Understanding
 - 18.6.2 Semantics and Communication
 - 18.6.3 Models of the Inner State of the Agent
 - 18.6.4 Multi-modal Communication
- 18.7 Human-Robot Ratio
- 18.8 Trust
- 18.9 Testing and Metrics
 - 18.9.1 Data Collection Methods
 - 18.9.2 Metrics
- 18.10 Human-Robot Interaction and the Seven Areas of Artificial Intelligence
- 18.11 Summary
- 18.12 Exercises
- 18.13 End Notes

V *Design and the Ethics of Building Intelligent Robots*

19 *Designing and Evaluating Autonomous Systems*

- 19.1 Overview
- 19.2 Designing a Specific Autonomous Capability
 - 19.2.1 Design Philosophy
 - 19.2.2 Five Questions for Designing an Autonomous Robot
- 19.3 Case Study: Unmanned Ground Robotics Competition
- 19.4 Taxonomies and Metrics versus System Design
- 19.5 Holistic Evaluation of an Intelligent Robot
 - 19.5.1 Failure Taxonomy
 - 19.5.2 Four Types of Experiments
 - 19.5.3 Data to Collect
- 19.6 Case Study: Concept Experimentation
- 19.7 Summary
- 19.8 Exercises

20 *Ethics*

- 20.1 Overview
- 20.2 Types of Ethics
- 20.3 Categorizations of Ethical Agents
 - 20.3.1 Moor's Four Categories
 - 20.3.2 Categories of Morality
- 20.4 Programming Ethics
 - 20.4.1 Approaches from Philosophy
 - 20.4.2 Approaches from Robotics
- 20.5 Asimov's Three Laws of Robotics
 - 20.5.1 Problems with the Three Laws
 - 20.5.2 The Three Laws of Responsible Robotics
- 20.6 Artificial Intelligence and Implementing Ethics
- 20.7 Summary
- 20.8 Exercises
- 20.9 End Notes

Bibliography

Index

List of Figures

Figure 1.1 Two views of robots: a) the humanoid robot from the 1926 movie Metropolis (image courtesy Fr. Doug Quinn and the Metropolis Home Page), and b) a High Mobility Multipurpose Wheeled Vehicle (HMMWV), a military vehicle capable of driving on roads and open terrains

(photograph courtesy of the National Institute for Standards and Technology).

Figure 1.2 Two examples of UGVs being used in a hazardous materials incident exercise: a) a man-packable QinetiQ Dragonrunner with the operator control unit and antennas mounted to the backpack and b) a man-portable iRobot Packbot 510.

Figure 1.3 Two examples of rotor-craft UAVs in a hazardous materials incident exercise: a) a Leptron Avenger and b) a AirRobot 100B quadrotor.

Figure 1.4 Two types of unmanned marine vehicles used at the 2011 Tohoku tsunami response: a box-like SeaBotix ROV on the bottom and a torpedo shaped YSI Oceanmapper AUV on the top.

Figure 2.1 Timeline of major milestones in intelligent robotics.

Figure 2.2 A Model 8 Telemanipulator or “waldo.” The upper portion of the device is placed in the ceiling, and the portion on the right extends into the hot cell. (Photograph courtesy Central Research Laboratories.)

Figure 2.3 An RT3300 industrial manipulator. (Photograph courtesy of Seiko Instruments.)

Figure 2.4 A MOVEMASTER® robot: a.) the robot arm and b.) the associated joints.

Figure 2.5 Motivation for intelligent planetary rovers: a.) Astronaut John Young awkwardly collecting lunar samples on Apollo 16, and b.) Astronaut Jim Irwin stopping the lunar rover as it slides down a hill during the Apollo 15 mission. (Photographs courtesy of the National Aeronautics and Space Administration.)

Figure 2.6 Shakey, the first AI robot. (Photograph courtesy of SRI.)

Figure 2.7 Sojourner Mars rover. (Photograph courtesy of the National Aeronautics and Space Administration.)

Figure 2.8 The iRobot Roomba® vacuum cleaner that uses animal-like intelligence to adapt to rooms.

Figure 2.9 Teledyne Ryan Firebee UAV (target variant, IDF designation Shadmit) at Muzeyon Heyl ha-Avir, Hatzerim airbase, Israel. 2006. (Photograph courtesy of WikiCommons under the Creative Commons Attribution-Share Alike 3.0 Unported license.)

Figure 3.1 Visualizing how the four aspects of automation and autonomy combine to create the DNA for a particular intelligent system.

Figure 4.1 Central nervous system represented as an operational architecture.

Figure 4.2 Organization of the primitives in the Reactive Layer.

Figure 4.3 Primitives in the Deliberative Layer.

Figure 4.4 Sublayers in the Deliberative Layer with the four key deliberative functions.

Figure 4.5 Canonical AI robotics operational architecture.

Figure 4.6 The five common subsystems in intelligent robotics, adapted from Technology Development for Army Unmanned Ground Vehicles.

Figure 4.7 The three styles of interaction: a) SENSE, PLAN, ACT, b) SENSE-ACT, and c) PLAN, SENSE-ACT.

Figure 4.8 Example of the three sensing routes for a driverless car: local, global, and hybrid.

Figure 4.9 Example of a Hierarchical Systems Architecture for a driverless car.

Figure 4.10 Example of a Reactive Systems Architecture for a driverless car.

Figure 4.11 Example of a Hierarchical Systems Architecture for a driverless car.

Figure 4.12 SENSE-PLAN-APPROVE-ACT cycle.

Figure 5.1 Organization of a telesystem. (Photographs courtesy of Oak Ridge National Laboratory.)

Figure 5.2 Manipulator interface used in nuclear hot cell work. (Photograph courtesy of Central Research Laboratories.)

Figure 5.3 The types of supervisory control represented as quadrants.

Figure 5.4 Human supervisory control matrix expanded for telesystems.

Figure 5.5 Manual control in teleoperation, showing that the remote may have some onboard control.

Figure 5.6 Local display from an iRobot Packbot showing perception.

Figure 5.7 Traded control between a teleoperator and a telefactor.

Figure 5.8 Shared control between a teleoperator and a telefactor.

Figure 5.9 DarkStar unmanned aerial vehicle. (Photograph courtesy of DefenseLink Office of the Assistant Secretary of Defense-Public Affairs.)

Figure 5.10 A teleoperated Bobcat and Talon robots working together to enable the Bobcat to clear debris at Fukushima. (Photograph courtesy of QinetiQ.)

Figure 6.1 Marr's Computational Theory.

Figure 6.2 Robots built at the Bio-Bot Laboratory at Case Western Reserve University imitate cockroaches at Level 3: a.) Robot I, an earlier version, and b.) Robot III. (Photographs courtesy of Roger Quinn.)

Figure 6.3 Genghis, a legged robot built by Colin Angle, IS Robotics, which imitates an insect at Levels 1 and 2. (Photograph courtesy of the National Aeronautics and Space Administration.)

Figure 6.4 Schema theory of a frog snapping at a fly.

Figure 6.5 Graphical illustration of the feed behavior emphasizing the computational theory Level 3 specificity needed for implementing in a computer.

Figure 6.6 Schema theory representation of a toad snapping at a fly when presented with two equidistant flies.

Figure 6.7 Unified model language representation of behavior, motor, and perceptual schemas.

Figure 6.8 Feed behavior decomposed into perceptual and motor schemas.

Figure 6.9 A graphical representation of a behavior.

Figure 7.1 Action-Perception Cycle.

Figure 7.2 A collection of artificial bait, possibly the first example of humans exploiting affordances. Notice that the lures exaggerate one or more attributes of what a fish might eat.

Figure 7.3 The GRUFF system: a.) input, and b.) different types of chairs recognized by GRUFF. (Figures courtesy of Louise Stark.)

Figure 7.4 Innate Releasing Mechanism as a process for activating behaviors.

Figure 7.5 The hiding behavior expressed as actograms and internal state for the set of component behaviors.

Figure 8.1 A graphical illustration of the coordination function.

Figure 8.2 A taxonomy of algorithms that can serve as the coordination function.

Figure 8.3 Example of an obstacle exerting a repulsive potential field over the radius of one meter.

Figure 8.4 Five primitive potential fields: a.) uniform, b.) perpendicular, c.) attraction, d.) repulsion, and e.) tangential.

Figure 8.5 Plots of magnitude profiles for a field of radius 5 units: a.) constant magnitude, b.) linear drop off with a slope of -1, and c.) exponential drop off.

Figure 8.6 A bird's-eye view of a world with a goal and obstacle and the two active behaviors for the robot who will inhabit this world.

Figure 8.7 Potential fields from the world in figure 8.6: a.) repulsive from the obstacle, b.) attractive from the goal, and c.) combined.

Figure 8.8 Path taken by the robot.

Figure 8.9 Khepera miniature robot: a.) a Khepera on a floppy disk, b.) IR sensors (small black squares) along the "waistline," and c.) orientation of the IR sensors.

Figure 8.10 Khepera in a box canyon a.) range readings and b.) vectors from each instance of RUNAWAY (gray) and the summed output vector.

Figure 8.11 Problem with potential fields: a.) an example, and b.) the use of NaTs to eliminate the problem.

Figure 8.12 "Veteran" robots of the MIT AI Laboratory using the subsumption architecture. (Photograph courtesy of Rod Brooks.)

Figure 8.13 Level 0 in the subsumption architecture.

Figure 8.14 Polar plot of eight sonar range readings: a.) "robocentric" view of range readings along acoustic axes, and b.) unrolled into a plot.

Figure 8.15 Level 0 recast as primitive behaviors.

Figure 8.16 Level 1: wander.

Figure 8.17 Level 1 recast as primitive behaviors.

Figure 8.18 Level 2: follow corridors.

Figure 8.19 A FSA representation of the coordination and control of behaviors in the UGV competition: a.) a diagram and b.) the table.

Figure 8.20 An alternative FSA representation of the coordination and control of behaviors in the UGV competition: a.) a diagram and b.) the table.

Figure 8.21 Georgia Tech's award-winning Pick Up the Trash robots with the trash gripper (photograph courtesy of Tucker Balch and AAAI).

Figure 8.22 A FSA for the Pick Up the Trash task: a.) state diagram, and b.) table showing state transitions.

Figure 8.23 Comparison of script structures to behaviors.

Figure 9.1 Clementine, a Denning DRV-1 robot.

Figure 9.2 The Uranus robot with omnidirectional wheels (courtesy of Gwpcmu - Own work, CC BY 3.0, <https://commons.wikimedia.org/w/index.php?curid=11440618>).

Figure 9.3 Example of skid-steered robots: a.) an Inuktun microVGTV with variable geometry and b.) an Endeavor Packbot 510 with flippers.

Figure 9.4 Examples of biomimetic robots: a.) a long, thin active scope camera robot being inserted into rubble and b.) a snake robot. (Snake robot photograph courtesy of Howie Choset.)

Figure 9.5 RHex robot (Courtesy of Gadlopes at English Wikipedia (Transferred from en.wikipedia to Commons.) Public domain, via Wikimedia Commons).

Figure 9.6 Energy required for different modes of locomotion, adapted from.

Figure 9.7 Dante (photograph courtesy of NASA Ames Research Center).

Figure 9.8 A leg as an inverted pendulum illustrating the zero moment point (ZMP).

Figure 9.9 The three basic gaits: trot, pace, and bound.

Figure 10.1 A model of reactive sensing (a world model would replace the schemas in deliberative sensing).

Figure 10.2 The ring of IR sensors on a Khepera robot. Each black square mounted on three posts is a emitter and receiver.

Figure 10.3 RGB color cube.

Figure 10.4 Images showing visual erosion of an orange landmark sticking up from a small robot (not visible): a.) Original image and RGB segmentation and b.) original image and degradation in RGB segmentation as robot moves farther from the object.

Figure 10.5 HSV space representation.

Figure 10.6 Spherical coordinate transform.

Figure 10.7 A comparison of color spaces: a.) a landmark in the original image, b.) RGB segmentation, c.) HSI segmentation, and d.) SCT segmentation.

Figure 10.8 Segmentation of a red Coca-Cola can: a.) original image and b.) resulting red regions. Note that some pixels not associated with the can showed a reddish color.

Figure 10.9 An urban search and rescue scene: a.) a Denning mobile robot searching, b.) image from the camera, and c.) segmentation for orange.

Figure 10.10 a.) A histogram for b.) the image of the children's toy, Barney®.

Figure 10.11 A Denning mobile robot using a color histogram to play tag with a poster of Sylvester and Tweety.

Figure 10.12 Sequence showing a Denning mobile robot with redundant cameras, responding to sensing failures introduced by Dave Hershberger.

Figure 10.13 Example of redundant top and bottom sonar rings.

Figure 10.14 Three types of behavioral sensor fusion: a.) sensor fission, b.) action-oriented sensor fusion, and c.) sensor fashion.

Figure 11.1 Ways of extracting depth from a pair of cameras: a.) vergence of the cameras to determine the depth of a point, and b.) a set of rectified stereo images.

Figure 11.2 A stereo camera pair mounted on a pan/tilt head.

Figure 11.3 Uranus undergoing calibration. (Photograph courtesy of Hans Moravec.)

Figure 11.6 A set of stereo images from a Triclops stereo camera and resulting depth map. (Images courtesy of Chandra Kambhamettu.)

Figure 11.4 Robots and stereo: a.) The Stanford Cart developed in the late 1970s (Photograph courtesy of Hans Moravec.), and b.) The Marsokhod rover developed in the late 1990s jointly by scientists from McDonnell Douglas, Russia, and NASA Ames Research Center. (Image courtesy of the National Aeronautics and Space Administration.)

Figure 11.5 A simplified flow of operations in extracting range from a stereo pair.

Figure 11.7 Polly a visually guided reactive robot using a black and white camera. (Photograph courtesy of Ian Horswill and AAAI.)

Figure 11.8 Polaroid ultrasonic transducer. The membrane is the disk.

Figure 11.9 The regions of space observed by an ultrasonic sensor.

Figure 11.10 Three problems with sonar range readings: a.) foreshortening, b.) specular reflection, and c.) cross-talk.

Figure 11.11 Maps produced by a mobile robot using sonars in: a.) a lab and b.) a hallway. (The black line is the path of the robot.)

Figure 11.12 Situations and resulting images of a.) flat surface, b.) an obstacle, and c.) a negative obstacle.

Figure 11.13 Lidar images from an Odetics LADAR camera: a.) range and b.) intensity. (Images courtesy of University of South Florida and Oak Ridge National Laboratory.)

Figure 11.14 SICK laser, covering a 180° area.

Figure 11.15 Three views of a person's hand taken with a Kinect in different lighting. Above is indoor lighting, below left is partial shade outdoors, and below right is full sun. The green boxes show where an extraction algorithm hypothesizes the presence of a hand. (Images courtesy of Carlos Soto.)

Figure 11.16 The sequence of processing activities associated with point clouds.

Figure 11.17 USF robots in the "Hors d'Oeuvres, Anyone?" event. a.) Family portrait, where Borg Shark is on the left, Puffer Fish on the right, with b.) the thermal sensor located as the Borg Shark's "third eye," c.) the SICK laser located behind Borg Shark's teeth (head piece is removed for better view), and d.) a profile of Puffer Fish's skirt showing spatial relationship to sonar.

Figure 11.18 State diagram for the Borg Shark, annotated to show sensors being used.

Figure 12.1 An example for Strips of two rooms joined by an open door.

Figure 12.2 Erratic, a Pioneer robot. (Photograph courtesy of Kurt Konolige.)

Figure 12.3 Simplified view of Saphira architecture.

Figure 12.4 Nested Hierarchical Controller.

Figure 12.5 Examination of planning components in the NHC architecture.

Figure 12.6 Each of the 3T layers running on a separate computer, controlling the EVAHR robot. (Photo courtesy of NASA Johnson Space Center.)

Figure 12.7 3T architecture.

Figure 13.1 Two Tour Guide Robots: a.) Rhino in the Deutsches Museum in Bonn (in the center), and b.) a close up of the more emotive Minerva. (Photographs courtesy of Sebastian Thrun and Wolfram Burgard.)

Figure 13.2 Types of path planning algorithms.

Figure 13.3 Artificial landmarks used in the 1992 AAAI Mobile Robot Competition. (Photograph courtesy of AAAI.)

Figure 13.4 Representation of a floor plan as a relational graph.

Figure 13.5 Propagation of error in a relational graph.

Figure 13.6 Multi-level spatial hierarchy, after Byun and Kuipers.

Figure 13.7 A robot reacting to a corner and moving to a distinctive place within the neighborhood.

Figure 13.8 Behaviors serving as local control strategies and releasers serving as means of signaling the entrance to a neighborhood.

Figure 13.9 A student testing an unidentified entry in the competition arena at the 1994 AAAI Mobile Robot Competition “Office Delivery” event. (Photograph courtesy of AAAI.)

Figure 13.10 a.) Metric map of an office layout, b.) graphical topological representation, and c.) refined graph for traveling from R3 to R7.

Figure 13.11 Scenario for moving from R7 to R2. Shaded gateways are extraneous and discarded by the planner.

Figure 13.12 Scenario for moving from R1 to R0 with blocked path.

Figure 14.1 Taxonomy of path planning algorithms.

Figure 14.2 Reduction of a 6DOF world space to a 2DOF configuration space.

Figure 14.3 An a priori map, with object boundaries “grown” to the width of the robot (shown in gray). The robot is not shown on the map.

Figure 14.4 Meadow maps: a.) partitioning of free space into convex polygons and b.) generating a graph using the midpoints.

Figure 14.5 String tightening as a relaxation of an initial path.

Figure 14.6 Graph from a generalized Voronoi graph (GVG).

Figure 14.7 Regular grid.

Figure 14.8 Quadtree Cspace representation.

Figure 14.9 Graph for an *A* search algorithm example.

Figure 14.10 An *A** example.

Figure 14.11 a.) What *A** “sees” as it considers a path A-?-E, and b.) the original graph. c.) What *A**

“sees” from considering a path A-D-?-E, and d.) the associated graph.

Figure 14.12 A wave propagating through a regular grid. Elements holding the current front are shown in gray: older elements are shown in dark gray.

Figure 14.13 Layout showing unmodeled obstacle. a.) Gray line shows expected path, long dashed line shows the actual path with Trulla, and short dashed line shows purely reactive path. b.) *Clementine* opportunistically turning.

Figure 14.14 Opportunity to improve path. The gray line is the actual path, while the dashed line represents a more desirable path.

Figure 14.15 A random tree generated by the RRT-Connect algorithm and a partial path through the tree. Courtesy of Dmitry Trifonov.

Figure 14.16 GVG exercise.

Figure 14.17 A* exercise.

Figure 15.1 Three types of localization.

Figure 15.2 Markov global localization example. a.) Robot senses unknown location, b.) resulting in four equally probable poses, c.) moves forward and updates sensing, d.) resulting in two highly probable poses and two less likely poses, e.) moves forward and updates sensing again, f.) resulting in a set of three possible poses, one of which is clearly more probable.

Figure 15.3 a.) An ATRV robot crossing a field with tall grass and b.) a comparison of the unfiltered and filtered laser range scans.

Figure 15.4 An orthomosaic of a mudslide derived from approximately 20 images taken by a small unmanned aerial vehicle.

Figure 15.5 a.) a digital surface map and b.) a point cloud of a mudslide derived from approximately 100 images taken by a small unmanned aerial vehicle.

Figure 15.6 Three regimes of gross scale of a region’s environment relative to that of a physical agent.

Figure 15.7 An example of the tortuosity of a region for a UAV (from Agarwal and Murphy).

Figure 15.8 Example of a robot exploring an area using frontier-based method. (Figure courtesy of the Navy Center for Applied Research in Artificial Intelligence, Naval Research Laboratory.)

Figure 15.9 Example of a robot exploring an indoor area using a GVG method. a.) The robot starts in the lower right corner and b.) explores the center. The black lines indicate the ideal GVG and the white

lines indicate the portion of the GVG that the robot has traveled. (Figures courtesy of Howie Choset.)

Figure 15.10 Example of a robot exploring an indoor area using a GVG method, continued. a.) reaches a dead end and backtracks, and b.) completely covers the area, though it has not traveled all edges. (Figures courtesy of Howie Choset.)

Figure 16.1 An example of induction.

Figure 16.2 Two examples of overfitting. Both are consistent with the training data but unlikely to correctly classify new data.

Figure 16.3 An example of grouping of outputs by a support vector machine.

Figure 16.4 An example of a.) a training set and the resulting b.) decision tree.

Figure 16.5 An example of a feedforward artificial neural network.

Figure 16.6 State transitions for removing a cap from a bottle with Rewards Matrix and Q Matrix.

Figure 16.7 Genetic algorithm example: a.) The rover and its genome, b.) the members of the first generation, c.) associated belief mass, d.) resultant belief function, and e.) mutations.

Figure 16.8 Mitchell's learning architecture adapted for AI robotics.

Figure 16.9 ANN for classifying logos by color.

Figure 16.10 Rewards and U matrix.

Figure 17.1 Relationship between multirobot systems research and other topics in artificial intelligence and robotics.

Figure 17.2 Two views of a marsupial robot team at University of South Florida. Silver Bullet is the "mother" carrying a tracked chemical inspection robot Bujold, the "daughter." Bujold exits from the rear of the jeep. (Photographs by Tom Wagner.)

Figure 17.3 Operational architecture showing relationship between interactive and deliberative and reactive functionality.

Figure 17.4 The Nerd Herd. (Photograph courtesy of USC Interaction Laboratory.)

Figure 18.1 The Mine Crawler robot and its user interface showing the robot's-eye view of the environment with time, date, and distance information overlaid.

Figure 18.2 How cognitive task analysis and cognitive work analysis relate to each other for human-robot interaction.

Figure 18.3 Example of conceptual dependency theory decomposing a directive into acts.

Figure 18.4 The proxemic regions around a person; distances may vary slightly with culture and gender.

Figure 18.5 Form used for recording field observations about human-robot interaction.

Figure 18.6 An example of a HRI study participant being fitted with biometric sensors.

Figure 19.1 Omnibot, a mobile robot built from a Fisher Price Power Wheels battery-powered toy jeep by students and Omnitech Robotics, Inc.

Figure 19.2 The course for the 1994 Ground Robotics Competition.

Figure 19.3 The behavioral layout of the CSM entry in the 1994 Unmanned Ground Vehicle Competition.

Figure 19.4 A taxonomy of robot failure types.

Figure 19.5 Nuclear forensic localization concept experimentation: a.) the Prop 133 site at Disaster City® serving as the simulated hospital; the room containing the radioactive source is highlighted, b.) the radioactive source in a bright blue container on floor, c.) an iRobot Packbot 510, and d.) robot path relative to the source.

List of Tables

Table 4.1 Three robot primitives defined in terms of inputs and outputs.

Table 4.2 Representative levels of automation hierarchy, adapted from Endsley and Kaber.

Table 4.3 The three robot primitives and their inputs and outputs.

Table 8.1 Relationship of finite state automata to behaviors.

Table 9.1 Combinations of leg events for two legs.

Table 11.1 Sensors for Borg Shark.

Table 11.2 Sensors for Puffer Fish.

Table 11.3 Behaviors for Borg Shark.

Table 11.4 Behaviors for Puffer Fish.

Table 18.1 Examples of human-robot interaction for combinations of cooperation and engagement.

Table 19.1 Example behavior table.

Table 19.2 New behavioral table for the CSM entry.

Table 19.3 Four types of experiments, modified from Murphy.

Preface

This book is intended to serve as a textbook for advanced juniors and seniors, first year graduate students in computer science and engineering, and practitioners of unmanned systems. The reader is not expected to have taken a course in artificial intelligence (AI) as the book attempts to introduce key AI concepts from all subdisciplines of AI throughout the text. Indeed, a review of the first edition of this book for AI Magazine suggested that this book could double as an introduction to artificial intelligence. For those readers with a background in AI, the book includes pointers to additional readings and advanced exercises.

The second edition of *Introduction to AI Robotics* attempts to cover all the topics needed to design and evaluate an artificially intelligent robot for applications involving sensing, acting, planning, and learning. Like the 2001 first edition, the second edition is a broad survey attempting to anchor the reader in the key areas of AI and explain how they contribute to autonomous capabilities, provide the terminology needed to search for what has and is being done in a particular area, and enable the reader to assess whether an algorithm or system will work for a particular application. The second edition is a major expansion and reorganization of the first edition reflecting the tremendous progress made in the past 15 years. The first edition focused on explicitly introducing all the pieces of the “500-piece puzzle” that was AI robotics at the time. The second edition focuses on introducing the pieces of the larger “1000-piece puzzle” that AI robotics has become and on designing and evaluating a set of autonomous capabilities for systems that will interact with other robots, software agents, and humans. The book is not about how to code specific algorithms but rather about what algorithms to code and why. Informally, I think of the second edition as a combination of enabling a reader to “walk the walk and talk the talk” and defending oneself from the hype around AI by learning “how to kick the tires” of a flawed car that an unscrupulous salesman may be trying foist on an unsuspecting buyer.

The book is divided into five parts following a design narrative. Each part consists of multiple chapters, and each chapter generally follows a similar layout of overview, details, discussion of how that capability involves each of the core areas of AI, summary, exercises, and end notes. Case studies are incorporated wherever possible. The end notes are references to essential books for a roboticist’s bookshelf or pointers to related interesting concepts. The most popular end notes, perhaps the most popular portion of the first edition, have been about robot trivia and how concepts in the chapter show up (or are greatly abused) in science fiction. The second edition maintains this tradition, but the robotics-through-science-fiction connection is now a separate book and blog.

Part I attempts to provide a framework for thinking about artificial intelligence for robotics and a skeleton for the muscles and nerves of intelligent functions. Chapter 1 defines intelligent robots and

introduces why artificial intelligence is needed. Chapter 2 takes the reader through a historical journey of the process of organizing intelligence, leading to a discussion in chapter 3 of automation and autonomy and how automation and autonomy represent fundamentally different paradigms in designing intelligence. Artificial intelligence researchers concentrate on autonomy and have converged on a canonical hybrid deliberative/reactive operational architecture with *reactive*, *deliberative*, and *interactive* functionality and a systems architecture with *Planning*, *Cartographer*, *Navigation*, *Motor Schema*, and *Perception* subsystems, described in chapter 4. Part I concludes with a review of teleoperation and telesystems in chapter 5 as teleoperation is treated, incorrectly, as a degenerative case of autonomy. Understanding why teleoperation autonomy isn't unintelligent or a failure of autonomy is an important step in understanding AI robotics and the AI robotics design paradigm.

Part II expounds on the reactive functionality of sensing and acting in AI robotics as every intelligent robot using AI principles is likely to have a reactive component. Chapter 6 introduces principles of animal intelligence that motivate "lower" or "less intelligent" forms of autonomy. Chapter 7 dives further into biology, concentrating on how perception feeds intelligence, and introduces schema theory, the object-oriented-like structure used by biologists, psychologists, and roboticists. An intelligent agent will have multiple behaviors running concurrently thus understanding how to coordinate them is essential; that is the subject of chapter 8. Reactive behaviors drive basic locomotion (chapter 9) and requires sensing. General sensors and sensing techniques are presented in chapter 10 and range sensing is given special attention as a separate chapter (chapter 11). Note that manipulation is rarely treated as a reactive function and thus is covered as part of motion planning in chapter 14.

Part III introduces the deliberative functions most often associated with intelligence and the capability for autonomous initiative. Chapter 12 details the four functions of deliberation: to generate and monitor plans or solve problems, and to select and allocate resources. Most intelligent robots move in the environment, and thus an overview of what "navigation" means and entails is presented in chapter 13. Chapter 14 concentrates on metric path planning and motion planning, while chapter 15 delves into simultaneous localization and mapping (SLAM) as well as exploration. Learning can be either reactive or deliberative, but, as it is often associated with "higher" forms of intelligence, chapter 16 includes this part.

Reactive and deliberative functionality are sufficient to program the internals of an intelligent robot, but robots may work with other robots and will always engage a human. Part IV concentrates on the increasing interest in interaction. Chapter 17 surveys the myriad ways in which robots can form multirobot systems, while chapter 18 explores human-robot interaction. Human-robot interaction is an emerging field that includes user interfaces, human factors, robust systems, and trust.

Part V ends the book with a meta view of how to design and evaluate autonomous systems and the ethics of doing so. While each chapter has tried to identify the open challenges and gaps in classes of algorithms, chapter 19 revisits design and evaluation principles. The very public discussion of ethics and the killer robot and robot uprising themes in fiction motivate, in part, chapter 20. Ethics in robotics is not a purely debatable topic as designers have obligations to uphold professional ethics and must consider what is the ethical agency of their particular designs.

Since *Introduction to AI Robotics* is an introductory survey, it is impossible to cover all the fine work that has been done in the field. The guiding principle has been to include only material that clearly illuminates a specific topic. References to other approaches and systems are usually included as

an advanced reading question at the end of the chapter or as an end note. The *Springer Handbook of Robotics* provides a thorough survey of the field and should be an instructor's companion.

Acknowledgments

It would be impossible to thank all of the people involved in making this book possible, but I would like to try to list the ones who made the most obvious contributions. I'd like to thank my parents (I think this is the equivalent of scoring a goal and saying "Hi Mom!" on national TV) and my family (Kevin, Kate, and Allan). I had the honor of being in the first AI robotics course taught by my PhD advisor Ron Arkin at Georgia Tech (where I was also his first PhD student and the first student from the College of Computing to receive a PhD in robotics). Any errors in this book are strictly mine. David Kortenkamp suggested that I write the first book after using my course notes for a class he taught, which served as a very real catalyst. My colleagues on the Defense Science Board strongly encouraged me to write the second edition.

My students at Texas A&M University, both at College Station and at Corpus Christi, have been patient and insightful in their readings of various drafts of the book. In particular, I would like to thank Scott King, Jan Dufek, Matt Hegarty, Jesus Orozco, Xuesu Xiao, and Tim Woodward for their detailed suggestions, as well as Siddharth Agarwal, John DiLeo, Brittany Duncan, Zachary Henkel, Joshua Peschel, Cassandra Oduola, Traci Sarmiento, Carlos Soto, Vasant Srinivasan, and Grant Wilde. Certainly the students at both the Colorado School of Mines (CSM), where I first developed my robotics courses, and at the University of South Florida (USF) still merit special thanks for being guinea pigs for the first edition. I would like to specifically thank Leslie Baski, John Blitch, Glenn Blauvelt, Ann Brigante, Greg Chavez, Aaron Gage, Dale Hawkins, Floyd Henning, Jim Hoffman, Dave Hershberger, Kevin Gifford, Matt Long, Charlie Ozinga, Tonya Reed Frazier, Michael Rosenblatt, Jake Sprouse, Brent Taylor, and Paul Wiebe from my CSM days and Jenn Casper, Aaron Gage, Jeff Hyams, Liam Irish, Mark Micire, Brian Minten, and Mark Powell from USF. Special thanks go to the numerous reviewers of the first edition, especially Karen Sutherland and Ken Hughes. Karen Sutherland and her robotics class at the University of Wisconsin-LaCrosse (Kristoff Hans Ausderau, Teddy Bauer, Scott David Becker, Corrie L. Brague, Shane Brownell, Edwin J. Colby III, Mark Erickson, Chris Falch, Jim Fick, Jennifer Fleischman, Scott Galbari, Mike Halda, Brian Kehoe, Jay D. Paska, Stephen Pauls, Scott Sandau, Amy Stanislowski, Jaromy Ward, Steve Westcott, Peter White, Louis Woyak, and Julie A. Zander) painstakingly reviewed an early draft of the book and made extensive suggestions and added review questions. Ken Hughes also deserves special thanks; he also provided a chapter by chapter critique as well as witty emails. Ken always comes to my rescue.

Likewise, the book would not be possible without my ongoing involvement in robotics research; my efforts have been supported by the National Science Foundation (NSF), Defense Advanced Research Projects Agency (DARPA), and the Office of Naval Research (ONR). Most of the case studies came

from work or through equipment sponsored by NSF. Howard Moraff, Rita Rodriguez, and Harry Hedges were always very encouraging, beyond the call of duty of even the most dedicated NSF program director. Michael Mason also provided encouragement, in many forms, to hang in there and focus on education.

My editors, Marie Lufkin Lee on the second edition and Bob Prior on the first edition, and the others at the MIT Press (Amy Hendrickson, Christine Savage, Katherine Innis, Judy Feldmann, Margie Hardwick, and Maureen Cuper) also have my deepest appreciation for providing unfailingly good-humored guidance, technical assistance, and general savvy. Katherine and Judy were very patient on the first edition. Mike Hamilton at the Association for the Advancement of Artificial Intelligence (AAAI) was very helpful in making available the various “action shots” used throughout the book. Chris Manning provided the $\text{\LaTeX} 2\epsilon$ style files, and Liam Irish and Ken Hughes contributed helpful scripts.

Besides the usual suspects, there are some very special people who indirectly helped me. Without the encouragement of three liberal arts professors, Carlyle Ramsey, Monroe Swilley, and Chris Trowell, at South Georgia College in my small home town of Douglas, Georgia, I probably wouldn’t have seriously considered graduate school in engineering and computer science. They taught me that learning isn’t a place like a big university but rather a personal discipline. The efforts of my husband, Kevin Murphy, were, as always, essential. He worked hard to make sure I could spend the time on this book without missing time with the kids or going crazy. He also did a serious amount of editing, typing, scanning, and proofreading on both editions. I miss him dearly. I dedicate the book to these four men who have influenced my professional career as much as any academic mentor.

I

Framework for Thinking About AI and Robotics

Contents:

- [Chapter 1: What Are Intelligent Robots?](#)
- [Chapter 2: A Brief History of AI Robotics](#)
- [Chapter 3: Automation and Autonomy](#)
- [Chapter 4: Software Organization of Autonomy](#)
- [Chapter 5: Telesystems](#)

1

What Are Intelligent Robots?

Chapter Objectives:

- Define *intelligent robot*.
- List the three modalities of autonomous (unmanned) systems.
- List the five components common to all intelligent robots.
- Give four motivations for intelligent robots.
- List the seven areas of artificial intelligence and describe how they contribute to an intelligent robot.

1.1 Overview

ARTIFICIAL INTELLIGENCE

Artificial intelligence, or AI for short, can be thought of as *the science of making machines act intelligently*. AI has no commonly accepted, precise definition. One of the first textbooks on AI defined it as “the study of ideas that enable computers to be intelligent,”²¹⁹ which seemed to beg the question. A later textbook was more specific, “AI is the attempt to get the computer to do things that, for the moment, people are better at.”²¹⁶ This definition is interesting because it implies that once a task is performed successfully by a computer, then the technique that made it possible is no longer AI, but something mundane. That definition is fairly important to a person investigating AI methods for robots because it explains why certain topics suddenly seem to disappear from the AI literature: they were perceived as being resolved! Perhaps the most amusing of all AI definitions was the slogan for the now defunct computer company, Thinking Machines, Inc., “... making machines that will be proud of us.”

The concept of AI is controversial and has sparked ongoing philosophical debates on whether a machine can ever be intelligent. As the physicist Roger Penrose notes in his book, *The Emperor’s New Mind*: “Nevertheless, it would be fair to say that, although many clever things have indeed been done, the simulation of anything that could pass for genuine intelligence is yet a long way off.”²¹⁷ Engineers often dismiss AI as wild speculation. As a result of such vehement criticisms, many AI researchers often label their work as “intelligent systems” or “knowledge-based systems” in an attempt to avoid the controversy surrounding the term “AI.”

Fortunately, a single, precise definition of AI is not necessary to study AI robotics. AI robotics is *the application of AI techniques to robots*. More specifically, AI robotics is the consideration of topic areas traditionally covered by AI for application to robotics: learning, planning, reasoning, problem solving, knowledge representation, and computer vision.

This chapter poses six common questions about AI robotics. The most important question is *what is an intelligent robot?* The chapter answers this by first answering *what is a robot?* and then using the AI perspective to distinguish an intelligent robot from an unintelligent robot. Regardless of whether a robot is intelligent or not, it has some physical manifestation, which leads to two other questions: *What are the components of a robot?* and *What are the kinds of robots?* Complementary to the question: What is an intelligent robot? is the question of motivation: *What are intelligent robots used for?* The chapter attempts to justify *Why is intelligence needed?* by introducing the seven key areas in artificial intelligence and describing how those areas can contribute to making a robot more useful for applications.

1.2 Definition: What Is an Intelligent Robot?

In popular culture, the term robot generally connotes the anthropomorphic (human-like) appearance of a mechanical device; consider robot “arms” for welding. The tendency to think about robots as having a human-like appearance may stem from the origins of the term “robot.” The word “robot” came into the popular consciousness on January 25, 1921, in Prague, with the first performance of Karel Čapek’s play, R.U.R. (Rossum’s Universal Robots).⁶¹ In R.U.R., an unseen inventor, Rossum, has created a race of workers that were made from a vat of biological parts, smart enough to replace a human in any job (therefore “universal”). Čapek called the workers robots, a term derived from the Czech word “roboří,” which is loosely translated as menial laborer. The term robot workers implied that the artificial creatures were strictly meant to be servants to free “real” people from any type of labor but were too lowly to merit respect. This attitude towards robots has disastrous consequences, and the moral of the rather socialist story is that work defines a person.

The shift from robots as human-like servants constructed from biological parts to human-like servants made up of mechanical parts was probably due to science fiction. Three classic films, *Metropolis* (1926), *The Day the Earth Stood Still* (1951), and *Forbidden Planet* (1956), cemented the assumption that robots were mechanical in origin, and ignored the biological origins in Čapek’s play. Meanwhile, computers were becoming commonplace in industry and accounting, gaining a reputation of being literal minded. Industrial automation confirmed this reputation as robot arms were installed that would go through the motions of assembling parts, even if there were no parts. Eventually, the term robot took on nuances of factory automation, mindless and good only for well-defined repetitious work.

The notion of anthropomorphic, mechanical, and literal-minded robots complemented the viewpoint elaborated in many of the short stories in Isaac Asimov’s perennial favorite collection, *I, Robot*.¹⁴ Many (but not all) of these stories involve either a “robopsychologist,” Dr. Susan Calvin, or two erstwhile trouble shooters, Powell and Donovan, diagnosing robots who behaved logically but did the wrong thing. The stories often revolve around Asimov’s Three Laws of Robotics, which cleverly appear reasonable but create the opportunity for unexpected behavior because of hidden inconsistencies in the

laws. Asimov's Three Laws of Robotics are often cited as a formal, comprehensive model for actual safe, robot operation when, as detailed in¹⁴⁵ they are anything but good laws (see chapter 20).

BIOMIMETIC

The shift from human-like mechanical creatures to whatever physical form gets the job done is based on good engineering design principles. While robots are mechanical, they do not have to be anthropomorphic or even animal-like in appearance. Consider robot vacuum cleaners; they look like vacuum cleaners, not janitors or Rosie the robotic maid on the Jetsons. And the HelpMate Robotics, Inc., robot which delivers hospital meals to patients to permit nurses to spend more time with patients, looks like a cart, not a nurse. Even robots that are based on animals, termed *biomimetic* robots, may duplicate biological principles but not resemble the animal. For example, RHex, the hexapod robot, has curved springs for legs and the legs rotate, yet the unusual legs duplicate the springiness of a cockroach's legs, just as curved blades used by amputee runners, such as Oskar Pistorius, duplicate the springiness of human knees.



a.



b.

Figure 1.1 Two views of robots: a) the humanoid robot from the 1926 movie Metropolis (image courtesy Fr. Doug Quinn and the Metropolis Home Page), and b) a High Mobility Multipurpose Wheeled Vehicle (HMMWV), a military vehicle capable of driving on roads and open terrains (photograph courtesy of the National Institute for Standards and Technology).

The fact that robots interact with the physical world, regardless of shape, contributes to a workable definition of an intelligent robot. In artificial intelligence, an agent is an entity, a “something,” that can sense its surroundings and take actions that change the environment. Note that an agent is *situated* in an external world, that the agent is interesting because it effects change on its surroundings rather than solely observes and modifies only itself internally. The working definition of a robot as being able to effect change in its environment is at odds with the popular press, which uses the term “robot” synonymously with remote capabilities. For example, a sensor swallowed by a person, that passively collects measurements as it passes through the body, may be called a robot even though there is no action.

PHYSICALLY SITUATED

A robot is a special type of agent; it is *physically situated* in the “real world” while a software agent is situated in a virtual world defined by the World Wide Web, simulation, or confines of a software system. Not all robots are physically situated agents; factory robots, such as welding robots, blindly execute preprogrammed motions and thus do not meet the criterion of sensing their surroundings.

INTELLIGENT AGENT

But being a physically situated agent is not sufficient as a definition of an intelligent robot. A thermometer can be considered a physically situated agent; after all, a thermometer senses temperature and acts through a circuit to modify the environment. Fortunately, artificial intelligence makes a further distinction, specifying an *intelligent agent*, versus just an agent, *as a system that perceives its environment and takes actions which maximize its chances of success*.¹⁸¹

The phrase “maximize its chances of success” is the key in attributing intelligence to a device or program. A thermometer does not actively maximize its chances of success: it either works or does not, therefore it is not considered intelligent. In biology, success is usually defined in terms of surviving and reproducing. Cockroaches seem able to survive and reproduce with little of what is considered intelligence. But beavers go further in maximizing chances of success for survival of the individual and the species: they build dams to capture food and provide protection. Beavers show remarkable perceptual and optimization capabilities in selecting the location of a dam and the choice of trees to cut down and easily transport to the site.

INTELLIGENT ROBOT

Therefore, the definition that will be used in this book is: *An intelligent robot is a physically situated intelligent agent*.

1.3 What Are the Components of a Robot?

Robots are physically situated agents that consist of five major components: *effectors, perception, control, communications, and power*. Effectors are the appendages of the robot that move or move it, such as legs, arms, necks, and wrists. Effectors enable a robot to act on the environment. The major effectors in a ground robot are the wheels, tracks, or legs that give it navigational mobility. A new spate of work in robotics is centered on robot manipulators, essentially robot hands and arms. Perception is the set of sensors and sensing that provide a robot with the equivalent of eyes, ears, nose, smell, and touch. As described in chapter 9, perception generally requires both a sensor, the device that collects a signal, and algorithms to interpret the signal. Perception is how a robot senses the environment. Control is analogous to the central nervous system, where a computer processor(s) provides inner and outer loop control of the robot. The control component contains the computations that allow an intelligent robot to maximize its chances of success. Communication is how a robot interacts with other agents, if only the robot operator. Animals use bird and whale songs, displays of color and posture to communicate, while humans use natural language, gestures, proxemics, and other mechanisms to communicate with each other. Power enables the other functions; it duplicates the role of food and the digestive system in animals.

1.4 Three Modalities: What Are the Kinds of Robots?

UNMANNED VEHICLES

Intelligent robots can be used on the ground, in the air, or in water. If they function outdoors, they are commonly called *unmanned vehicles* or unmanned systems rather than robots. Despite there being only three modalities—ground, air, and water—there are at least seven abbreviations for robot platforms according to modality, size, and user community.

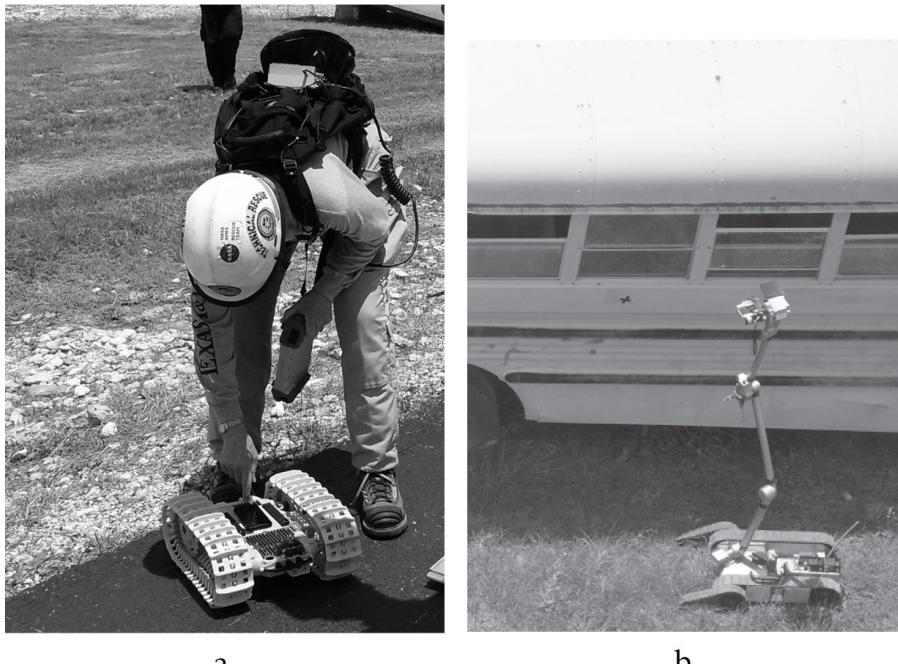
UNMANNED GROUND VEHICLES (UGV)

Robots that operate on the ground may be called mobile robots or *unmanned ground vehicles* (*UGV*), though UGV appears to becoming the term used for all ground robots. For the purposes of this book, UGV will be used for the superset of ground robots. UGVs can be divided into three categories.

- *Humanoid* or anthropomorphic robots have been popularized in movies but exist largely in research labs. The Honda P3, Sony Asimo, and Aldebaran Nao robots are examples of humanoid robots.

UGV SIZE

- *Mobile robots* are nonanthropomorphic, with the iRobot Roomba® and the National Aeronautics and Space Administration's Mars Exploratory Rovers as examples. The mobile robot category for military and public safety applications is further subdivided by *UGV size*. UGVs which fit into one or two backpacks (often one for the robot, the other for the controller and batteries) and are light enough to be carried by a person are called *man-packable* (see [figure 1.2](#)). Robots which require two people to carry (e.g., “you lift one end, I'll lift the other”) are called *man-portable*. A large robot which is too heavy or big to be lifted and carried is called *maxi*.



a.

b.

Figure 1.2 Two examples of UGVs being used in a hazardous materials incident exercise: a) a man-packable QinetiQ Dragonrunner with the operator control unit and antennas mounted to the backpack and b) a man-portable iRobot Packbot 510.

- *Motes* are miniature robots. Motes often do not have effectors for mobility, leading many roboticists to refer to them as *unattended ground sensors*.

UNMANNED AERIAL VEHICLES (UAV)

Robots that operate in the air are called *unmanned aerial vehicles (UAV)* or, in the United States, *unmanned aerial systems (UAS)*, to emphasize that they comprise a human-robot system not simply a platform. UAVs are often categorized based on form and on size, with several competing definitions. For the purposes of this book, there are three major types of UAVs.

- Fixed-wing aircraft that look and act like planes. U.S. Department of Defense Predators and Global Hawks are examples.
- Rotor-craft, or vertical take-off and landing platforms, that look and act like helicopters ([figure 1.3](#)). A commercial example is the Yamaha RMAX, an unmanned helicopter used extensively for crop dusting in small, terraced plots in Japan. Smaller platforms often have multiple rotors, such as the Parrot AR.Drone, and are sold as toys.



a.

b.

Figure 1.3 Two examples of rotor-craft UAVs in a hazardous materials incident exercise: a) a Leptron Avenger and b) a AirRobot 100B quadrotor.

- Small or micro UAVs (MAV), which are generally less than two meters in any characteristic dimension, for example, wing span, rotor blade length, or fuselage. “Micro” originated with the DARPA Micro Aerial Vehicle program, where micro meant less than 18 cm, the point at which the aerodynamic flight properties radically changed. Since the proliferation of UAVs for tactical military operations in Iraq and Afghanistan, micro tends to connote any UAV that is easy to pack and carry, similar to the UGV man-packable categorization. MAVs can be either fixed-wing or rotor-craft.

UNMANNED MARINE VEHICLES

UNMANNED SURFACE VEHICLE

UNMANNED UNDERWATER VEHICLE

Robots that operate on or under water are called *unmanned marine vehicles*. If the robot works on the surface and acts like a boat, it is called an *unmanned surface vehicle* (USV), where “surface” refers to the surface of the water. A USV never refers to a ground vehicle, even though a UGV typically operates on the surface of the ground. If the robot submerges, then it might be called an *unmanned underwater vehicle* (UUV), though it is more common to refer to it by subcategory. The two subcategories of UUVs are:

AUTONOMOUS UNDERWATER VEHICLE

- *autonomous underwater vehicle* (AUV), where the underwater robot is “free swimming,” that is, it is not tethered and is not in constant communication with an operator. AUVs are invariably preprogrammed, and, as seen from the definitions in chapter 2, would be more correctly named automated underwater vehicles. REMUS is a well known AUV used for oceanographic research, while the torpedo shaped YSI Oceanmapper (see [figure 1.4](#)) and manta ray-shaped gliders are becoming commonplace for coastal water-quality sampling.

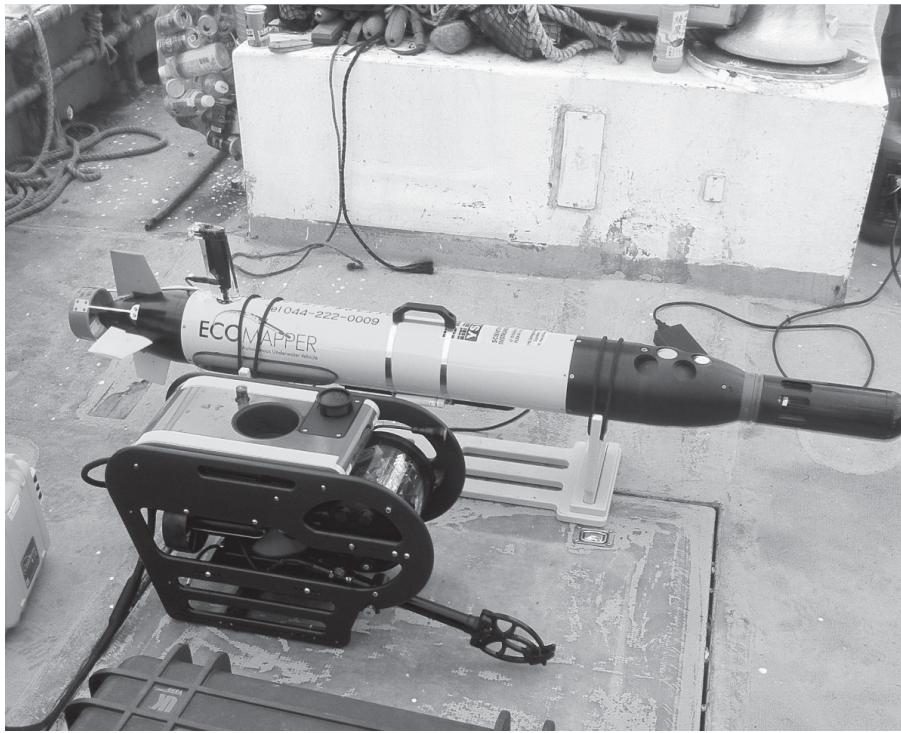


Figure 1.4 Two types of unmanned marine vehicles used at the 2011 Tohoku tsunami response: a box-like SeaBotix ROV on the bottom and a torpedo shaped YSI Oceanmapper AUV on the top.

REMOTELY OPERATED VEHICLE

- *remotely operated vehicle (ROV)*, where the underwater robot is tethered so that it can be controlled in realtime. Offshore oil and gas platforms use working-class ROVs, while smaller ROVs, such as those shown in figure 1.4, are used to inspect bridges, hulls of ships, and inlet/outlet pipes. Although any robot modality can be designed to be operated remotely by a human, the term ROV is used only for tethered UUVs, just as USV is reserved for water surface vehicles.

1.5 Motivation: Why Robots?

To date, robots have been used to for four reasons: to *replace* or substitute for humans, to allow humans to *project* themselves into a remote environment, to *assist* humans, and to *amuse*.

THE 3 Ds

Robots are often touted as a replacement for humans, captured by the well-worn joke that robots are good for the 3 Ds: jobs that are dirty, dull, or dangerous. Robots are indeed commonly used for dirty jobs, such as sewer and pipe inspection. Dull, menial tasks like factory work, vacuuming, and transporting meals and medicine in a hospital have been targeted by robot companies. Agriculture is a major area where robots have been explored as an economical alternative to hard-to-get menial labor. Tractors are using GPS and sophisticated soil sensors for precision agriculture. A robot called Shear

Magic, capable of shearing a live sheep, has been developed to compensate for the decline in the number of people available for sheep shearing. Possibly the most creative use of robots for agriculture is a mobile automatic milker developed in the Netherlands and in Italy.^{91;39} Rather than have a person attach the milker to a dairy cow, the roboticized milker arm identifies the teats as the cow walks into her stall, targets them, moves about to position itself, and finally reaches up and attaches itself. Unmanned aerial vehicles, or robot aircraft, are being used for inspection of pipelines and power lines.

But dangerous jobs, rather than the dirty or dull, especially military operations, have been a major driver in the development of intelligent robots. Humanitarian land demining is a task for robots.¹³¹ It is relatively easy to demine an area with a large bulldozer, but that destroys fields and roads and hurts the economy—smaller robots that can thoroughly, but less invasively, demine an area are desired. Robots were used for the Chernobyl, Three Mile Island, and Fukushima Daiichi nuclear accidents, and it is expected that the Fukushima Daiichi cleanup will be conducted with the extensive use of robots.

REMOTE PRESENCE

Intelligent robots do not have to replace a human because robots can be of tremendous benefit in allowing humans to project themselves into a remote environment in order to provide a *remote presence*.¹⁵³ Military drones allow soldiers to see and shoot around the world. Robot surgery can be automated, as seen with hip replacement systems, where the drilling of bones is done by a robot arm similar to that of a factory robot. But the bigger advance in surgery are robots, such as the DaVinci system, that allow surgeons to project themselves into the smaller scale of the human body and work with precision. Robots for fire rescue, law enforcement, and hazardous materials handling can enter dangerous areas and allow experts to rapidly assess, and mitigate, disasters without risking human exposure. Indeed, robots have been documented at 34 disasters between 2001 and 2013,¹⁴¹ and bomb squads generally have at least one robot. The oil and gas industries routinely use remotely operated vehicle (ROVs) to construct and inspect offshore wells. Telecommuting is another emerging industry in which colleagues attend meetings or work from remote locations without having to travel. Medical specialists are beginning to telecommute in order to provide immediate expert medical attention without the delays of traveling. As will be seen in chapter 5, robots for remote presence applications must be intelligent in order to complement the loss of sensing and to eliminate the need for a human to become a puppeteer.

An exciting new application of intelligent robots has been to assist humans. Robot assistants for eldercare, rehabilitation, and nursing is an emerging area, given an aging population. A recent movie, *Robot and Frank*, focused on the relationship of an elderly man with a robot that helped cook, clean, and even coach him on healthy habits. *Robot and Frank* featured a humanoid robot, but medical researchers have been experimenting with robot beds that help a patient get up safely and with robot walkers that detect and prevent a fall. Driverless cars are another type of assistive robotics, where the human can turn driving over to the vehicle.

The fourth application of robotics has been entertainment. Robots have been cast as toys such as Furby® and the short-lived Aibo® dog. Special effects and animatronics are moving to intelligent robots rather than relying on a team of puppeteers. Robot programming methods are being adopted by the videogame and graphics communities to create more realistic characters and avatars.

1.6 Seven Areas of AI: Why Intelligence?

Now that some possible uses of robots, and their shortcomings, have been covered, it is time to consider what the areas of artificial intelligence are, and how they could be used to overcome these shortcomings. The *Handbook of Artificial Intelligence*¹⁹ divides the field into seven main areas: *knowledge representation, understanding natural language, learning, planning and problem solving, inference, search, and vision*. In recent years, an eighth area of *distributed artificial intelligence*—how teams can work together—has emerged. The list of topics are building blocks; for example, optimal navigation through a complex building at noon requires knowledge about the building’s layout to support generating a plan; inference as to what parts of the building will be crowded with people trying to get to lunch and thus should be avoided; computer vision to perceive the building, the occupants, doors, and obstacles; some way of communicating that the robot needed to get through the crowd, otherwise it will become trapped; and perhaps learning that a corridor has been blocked off and not to try to go down that way in the future.

KNOWLEDGE REPRESENTATION

1. **Knowledge representation.** An important, but often overlooked, issue is how the robot represents its world, its task, and itself. Suppose a robot is scanning a room for an elderly person to assist. What kind of data structures and algorithms would it take to represent what a human looks like or what the human might need? How does a program capture everything important so as not to become computationally intractable? AI robotics explores the tension between the symbolic world representations that are easy for computers to create optimal paths through versus the direct perception used by animals that work directly from perception.

UNDERSTANDING NATURAL LANGUAGE

2. **Understanding natural language.** Natural language is deceptively challenging, apart from the issue of recognizing words which is now being done by commercial products such as Siri® and Alexa®.

It is not just a matter of looking up words, which is the subject of the following apocryphal story about AI. The story goes that after Sputnik went up, the US government needed to catch up with the Soviet scientists. However, translating Russian scientific articles was time consuming and not many US citizens could read technical reports in Russian. Therefore, the US decided to use these newfangled computers to create translation programs. The day came when the new program was ready for its first test. It was given the proverb: the spirit is willing, but the flesh is weak. The reported output: the vodka is strong, but the meat is rotten.

AI robotics explores the implicit and explicit communication needed for comfortable social interaction with robot.

LEARNING

3. **Learning.** Imagine a robot that could be programmed by just watching a human, or that a robot could learn by just repeatedly trying trying a new task by itself. Or that a robot experimented with a task through trial and error to generate a new solution. AI robotics is a study of the different types of learning and how learning can be applied to different functions.

PLANNING, PROBLEM SOLVING

4. **Planning and problem solving.** Intelligence is associated with the ability to plan actions needed to

accomplish a goal and solve problems when those plans fail. One of the early childhood fables, the Three Pigs and the Big, Bad Wolf, involves two unintelligent pigs who do not plan ahead and an intelligent pig who is able to solve the problem of why his brothers' houses have failed, as well as generate a new plan for an unpleasant demise for the wolf. AI robotics relies on planning and problem solving to cope with the unpredictability of the real world.

INFERENCE

5. Inference. Inference is generating an answer when there is not complete information.

Consider a planetary rover looking at a dark region on the ground. Its range finder is broken and all it has left is its camera and a fine AI system. Assume that depth information cannot be extracted from the camera. Is the dark region a canyon? Is it a shadow? The rover will need to use inference either to actively or passively disambiguate what the dark region is (e.g., kick a rock at the dark area versus reason that there is nothing nearby that could create that shadow). AI robotics techniques are increasingly engaging in inference.

SEARCH

6. Search. Search does not necessarily mean searching a large physical space for an object. In AI terms, search means efficiently examining a knowledge representation of a problem (called a “search space”) to find the answer. Deep Blue, the computer that beat World Chess master Garry Kasparov, won by searching through almost all possible combinations of moves to find the best choice. The legal moves in chess, given the current state of the board, formed the search space. Data mining or Big Data is a form of search. AI robotics uses search algorithms in generating optimal solutions in navigation or searching a knowledge representation.

VISION

7. Vision. Vision is possibly the most valuable sense humans have. Studies by Harvard psychologist, Steven Kosslyn, suggest that much of human problem solving capabilities stems from the ability to visually simulate the effects of actions in our head. As such, AI researchers have pursued creating vision systems both to improve robotic actions and to supplement other work in general machine intelligence. AI robotics relies heavily on computer vision to interpret video data and also the RGB-D cameras, such as Microsoft’s Kinect®.

It is tempting to assume that the history of AI Robotics is the story of how advances in AI have improved robotics. But that is not the case. In many regards, robotics has played a pivotal role in advancing AI. Breakthroughs in methods for planning for operations research problems came after the paradigm shift to reactivity in robotics in the late 1980s showed how unpredictable changes in the environment could actually be exploited to simplify programming. Many of the search engines on the World Wide Web use techniques developed for robotics. That the term “robot” is used to refer to pesky software agents that crawl the Web directly reflects the robotic heritage of these AI systems.

1.7 Summary

This chapter answers six common questions about robots and artificial intelligence. The most basic question is: *What is a robot?* A robot is a physically situated agent, which is an entity that can sense

and act on its operational environment. Understanding what a robot is sets the stage for answering: *What is an intelligent robot?* An intelligent robot is both a physically situated agent that can sense and act on its operational environment and an intelligent agent that can sense and act to maximize its chances of success. The physicality of a robot raises the question: *What are the components of a robot?*

Broadly speaking, a robot has five types of parts: effectors, perception, control, communications, and power. These parts loosely correspond to arms and legs; the five senses; the central nervous system; the ability to make and interpret sounds, gestures, and social displays; and the digestive system in animals. *What are the kinds of intelligent robots?* Intelligent robots are often classified by the operational environment in which a robot functions: ground, air, or water. Complementary to the question of what constitutes an intelligent robot is the question of motivation: *What are intelligent robots used for?* Robots are used for four reasons: to *replace* or substitute for humans for tasks that are dirty, dangerous, or dull; to allow humans to *project* themselves into a remote environment so that they can sense or act in real time from a distance, to *assist* humans such as with eldercare, and to *amuse* humans through the growing entertainment sector. The chapter discusses *why intelligence is needed* by introducing the seven key areas in artificial intelligence (knowledge representation, understanding natural language, learning, planning and problem solving, inference, search, and vision) and how they contribute to an intelligent agent. Each of these areas is a distinct field within artificial intelligence and contributes to intelligence.

1.8 Exercises

Exercise 1.1

What is an intelligent robot?

Exercise 1.2

A thermometer is a physically situated agent but not an intelligent agent. Name four other physically situated agents that are not intelligent agents.

Exercise 1.3

What are the three modalities of autonomous systems? Does one modality require more intelligence than the other two?

Exercise 1.4

What are the five components common to all intelligent robots? How do these correspond to parts of an animal?

Exercise 1.5

What are the four different categories in which intelligent robots have been used?

Exercise 1.6

Describe the seven areas of artificial intelligence in your own words and tell how they contribute to an intelligent robot.

Exercise 1.7

Discuss why robot applications demand more from AI than for playing chess or Jeopardy. Use your understanding of intelligent robots to give at least two reasons why this is true for each of the four applications of robots (replace, project, assist, amuse).

Exercise 1.8

Search the World Wide Web for applications and manufacturers of intelligent robots and make a list with at least 10 different entries.

Exercise 1.9

Look at the Scientific American Frontiers episode “Robots Alive!” Although this is an old episode, it is an excellent introduction as to how the different areas of AI contribute to intelligent robots. In particular, watch the “schedule a meeting” and “clean the room” competitions, and describe the role that each of the seven areas of AI play a role in the robots.

Exercise 1.10

Watch the movie *Robot and Frank*. Discuss how the robot uses each of the different areas of AI.

1.9 End Notes*The World’s End*

The robots in the 2013 movie *The World’s End* repeatedly correct the drunken protagonist who insists on defining “robot” as “slave.”

Robot name trivia

And speaking of British science fiction, the cantankerous robot Marvin in *The Hitchhiker’s Guide to the Galaxy* is widely assumed to be named after the cantankerous AI researcher, Dr. Marvin Minksy, at MIT.

2

A Brief History of AI Robotics

Chapter Objectives:

- Have a feel for the history of robotics and how the different approaches to intelligence arose.
- Be able to describe the difference between designing a robot as a *tool*, *agent*, or *joint cognitive system* in terms of the use of artificial intelligence.

2.1 Overview

This chapter provides a brief history of robotics. Understanding the history of robotics helps set the stage for defining the differences between automation and autonomy in the next chapter. It also addresses the issue of *why a robot is sometimes treated as a tool, sometimes as an independent agent capable of following its own agenda, and sometimes as a team member, similar to a service animal or sheep dog*.

Figure 2.1 gives a general timeline of the history of AI robotics, showing its relationship to other forms of robotics, which will be discussed below. One way of viewing robots is to look back on the early 1960s when there was a fork in the evolutionary path between engineering and artificial intelligence approaches. Engineering approaches were applied to building better tools, either hardware tools for the nuclear industry; “tools with really long handles,” such as teleoperated drones; or tools for manufacturing that moved in a bounded region but otherwise were similar to automatic weaving developed in the 1800s. Artificial intelligence approaches were applied to domains that needed adaptive, unsupervised mobile robots and that led to the idea of robots as agents. Today, robots are experiencing another change, that of being considered joint cognitive systems that are more natural and easier for humans to work with.

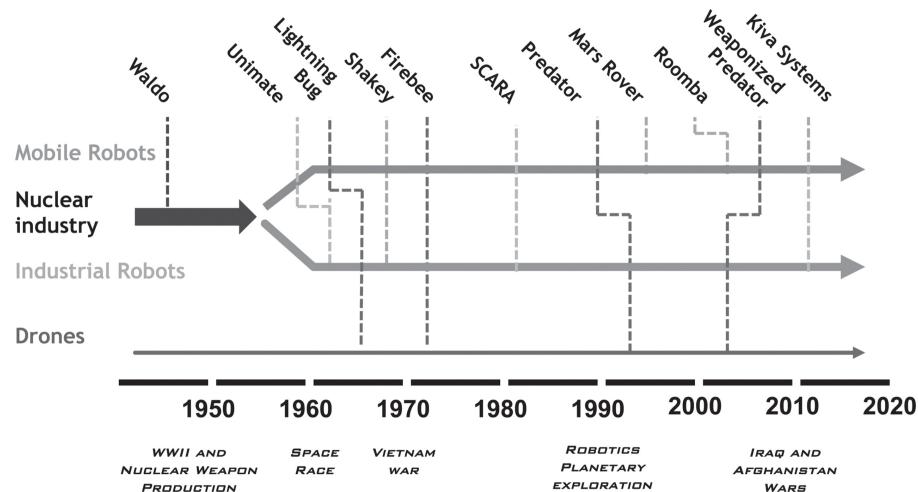


Figure 2.1 Timeline of major milestones in intelligent robotics.

2.2 Robots as Tools, Agents, or Joint Cognitive Systems

Historically, designers have assumed robots were either *tools*, *agents*, or *joint cognitive systems*. These initial assumptions impact whether artificial intelligence is used.

TOOL

A robot is often treated as a *tool*, that is a device that can perform specific, often highly limited or specialized, functions. A welding robot is an example of a robot as a tool. It is an entity that is physically situated in the world but which does not adapt to changes in the world. Parts to be welded are presented in precise configurations. Otherwise the robot does not function well—just like using the wrong screwdriver can damage the screw or be unsuccessful in turning the screw. The design process concentrates on how the robot can be optimized to perform a specific function, just like a screwdriver is optimized to turn screws and hammers to hammer nails. The process often includes designing parts and fixtures to hold the parts to make it easier for the robot. For tool design, artificial intelligence is often considered superfluous.

AGENT

The opposite of treating a robot as a tool is to treat it as an *agent*. Chapter 1 already provided a definition of an agent as a entity that can sense and effect change in the world, with robots and people being physically situated agents versus those working in simulation or on the World Wide Web. A robot vacuum cleaner is an example of an intelligent physically situated agent. In this case, the robot agent has its own goals (vacuuming) that it pursues by itself. An agent adapts to situations on its own; a vacuum cleaner works in different sizes and configurations of rooms. In contrast, a tool may be adapted to new situations, such as the handle of screwdriver being used to hammer a nail, but the tool did not think of the adaptation, and the adaptation is generally suboptimal. The agency view of robots has been reinforced by science fiction movies in which robots look and act as peers to people. The design process for an agent concentrates on how the agent will interact with the world, particularly via sensing and planning, and how it can adapt to new, but often similar, tasks or situations. As a result, artificial

intelligence is used to provide the needed functionality.

JOINT COGNITIVE SYSTEM

MIXED TEAM

A newer approach, *joint cognitive systems*, treats a robot as part of a human-machine team where the intelligence is synergistic, arising from the contributions of each agent. The team consists of at least one robot and one human and is often called a *mixed team* because it is a mixture of human and robot agents. Self-driving cars, where a person turns on and off the driving, is an example of a joint cognitive system. Entertainment robots are examples of mixed teams as are robots for telecommuting. The design process concentrates on how the agents will cooperate and coordinate with each other to accomplish the team goals. Rather than treating robots as peer agents with their own completely independent agenda, joint cognitive systems approaches treat robots as helpers such as service animals or sheep dogs. In joint cognitive system designs, artificial intelligence is used along with human-robot interaction principles to create robots that can be intelligent enough to be good team members.

2.3 World War II and the Nuclear Industry

Robotics has its roots in the history of engineering innovations and societal needs, including improvements in the way machines are controlled and the desire to have machines perform tasks that put human workers at risk. Probably the single biggest impetus for robotics was World War II and the nuclear industry, where designers thought of robots as tools.

In 1942, the United States embarked on the top secret Manhattan Project to build a nuclear bomb. The theory for making a nuclear bomb had existed for a number of years in academic circles. World War II military leaders of both sides believed the winner would be the side that could build the first nuclear device: the Allied Powers led by USA or the Axis, led by Nazi Germany.

One of the first problems that the scientists and engineers encountered was handling and processing radioactive materials, including uranium and plutonium, in large quantities. Although the immensity of the dangers of working with nuclear materials was not well understood at the time, all the personnel involved knew there were health risks. One of the first solutions to reducing those risks was the *glove box*. Nuclear material was placed in a glass box. Workers stood (or sat) behind a leaded glass shield and stuck their hands into thick rubberized gloves. This allowed the workers to see what they were doing and to perform almost any task that they could do without gloves.

TELEMANIPULATOR

But this was not an acceptable solution for highly radioactive materials, and mechanisms to more completely isolate the nuclear materials from humans had to be developed. One such mechanism was a force-reflecting *telemanipulator*, a sophisticated mechanical linkage which translated motions on one end of the mechanism to motions at the other end. A popular telemanipulator is shown in [figure 2.2](#).



Figure 2.2 A Model 8 Telemanipulator or “waldo.” The upper portion of the device is placed in the ceiling, and the portion on the right extends into the hot cell. (Photograph courtesy Central Research Laboratories.)

Nuclear workers would insert their hands into (or around) the telemanipulator and move it around while watching a display of what the other end of the arm was doing in a containment cell. Telemanipulators are similar in principle to the power gloves now used in computer games but were much harder to use. The mechanical technology of the time did not allow a perfect mapping of hand and arm movements to the robot arm. Often the operators had to make nonintuitive and awkward motions with their arms to get the robot arm to perform a critical manipulation—very much like working in front of a mirror. Likewise, the telemanipulators had challenges in providing force feedback so the operators were unable to feel how hard the gripper was holding an object. The lack of naturalness in controlling the arm (now referred to as a poor Human-Machine Interface) meant that

even simple tasks could take a long time. Operators might need years of practice to reach the point where they could do a task with a telemanipulator as quickly as they could do it with their own hands.

TELEFACTOR

TELEOPERATOR

WALDO

As the field of telemanipulation grew and was adopted for space operations, the roboticized portion of the system was called the *telefactor* and the person operating the system, the *teleoperator*. However, telefactor remains a specialized term rarely used in general practice. Instead, the nuclear industry began referring to telemanipulators and any telefactor as a *waldo* after the science fiction story about teleoperation by Robert Heinlein.

After World War II, many other countries became interested in producing a nuclear weapon and in exploiting nuclear energy as a replacement for fossil fuels in power plants. The USA and Soviet Union also entered into a nuclear arms race. The need to mass-produce nuclear weapons and to support peaceful uses of nuclear energy kept pressure on engineers to design robot arms which would be easier to control than telemanipulators. Machines that looked more like, and acted like, manufacturing robots began to emerge, largely due to advances in control theory.

CYBERNETICS

After World War II, pioneering work by Norbert Wiener allowed engineers to accurately control mechanical and electrical devices using *cybernetics*. Cybernetics originally meant *the study and creation of automatic control systems in animals and machines*, but the term has been applied to electrical circuits, mathematical logic, and other topics which “purists” believe are too distant from the original concept. Cybernetics was an attempt to move robots beyond the limits of simple tools into the realm of agents.

2.4 Industrial Manipulators

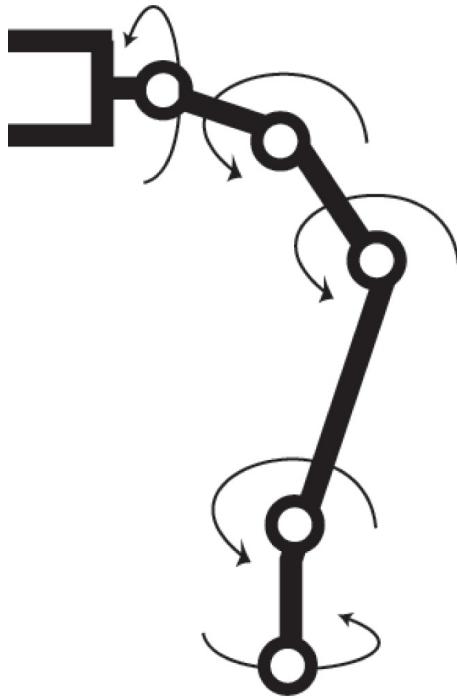
Successes with at least partially automating the nuclear industry also meant the technology was available for other applications, especially for general manufacturing. Robot arms began being introduced to industries in 1956 by Unimation (although it would be 1972 before the company made a profit).⁶¹ The two most common types of robot technology that have evolved for industrial use are robot arms, called industrial manipulators (figure 2.3), and mobile carts, called *automated guided vehicles* (AGVs), though that term has recently fallen into disuse. Industrial manipulators have typically been designed to be tools rather than agents.



Figure 2.3 An RT3300 industrial manipulator. (Photograph courtesy of Seiko Instruments.)

INDUSTRIAL MANIPULATOR

An *industrial manipulator*, to paraphrase the Robot Institute of America's definition, is a reprogrammable and multifunctional mechanism that is designed to move materials, parts, tools, or specialized devices. The emphasis in industrial manipulator design is to be able to perform a task repeatedly with a high degree of accuracy and speed. In order to be multifunctional, many manipulators have multiple degrees of freedom, as shown in [figure 2.4](#). The MOVEMASTER® arm has five degrees of freedom because it has five joints, each of which is capable of a single rotational degree of freedom. A human arm has three joints (shoulder, elbow, and wrist), two of which are complex (shoulder and wrist), yielding six degrees of freedom.



a.



b.

Figure 2.4 A MOVEMASTER® robot: a.) the robot arm and b.) the associated joints.

The key to success in industry in the 20th century was precision and repeatability on the assembly line for mass production; in effect, industrial engineers wanted to automate the workplace. Once a robot arm was programmed, it was supposed to be able to operate for weeks and months with only minor maintenance. As a result, the emphasis was placed on the mechanical aspects of the robot to ensure precision and repeatability and on the methods to make sure the robot could move precisely and repeatably, quickly enough to make a profit. Because assembly lines were engineered to mass produce a certain product, the robot did not have to be able to notice any problems. The standards for mass-production made it more economical to devise mechanisms that ensured parts were in the correct place. A robot for automation could essentially be blind and senseless.

Control theory is extremely important in industrial manipulators. Rapidly moving a large tool like a welding gun introduces interesting problems. One particular important problem is when the controller should start decelerating the welding gun so that it stops in the correct location without overshooting and colliding with the part to be welded. Also, oscillatory motion, in general, is undesirable. Another interesting problem is the joint configuration. If a robot arm has wrist, elbow and shoulder joints like a human, there are redundant degrees of freedom which means there are multiple ways of moving the joints that will accomplish the same motion. Which one is better, more efficient, and less stressful on the joint mechanisms?

BALLISTIC CONTROL

OPEN LOOP CONTROL

It is interesting to note that most manipulator control was assumed to be *ballistic control*, or *open loop control*. In ballistic control, the position trajectory and velocity profile are computed once, then the arm carries it out. There are no “in-flight” corrections, as a ballistic missile does not make any course corrections. In order to accomplish a precise task with ballistic control, everything about the device, including how it works, has to be modeled and figured into the computation.

CLOSED-LOOP CONTROL

FEEDBACK

The opposite of ballistic control is *closed-loop control*, where the error between the goal and current position is noted by a sensor(s), and a new trajectory and profile are computed and executed, then modified on the next update, and so on. Early closed-loop control requires external sensors to provide the error signal, or *feedback*.

In general, if the structural properties of the robot arm and its cargo are known, these questions can be answered and a program can be developed. In practice, the control theory is complex. The dynamics (how the mechanism moves and deforms) and kinematics (how the components of the mechanism are connected) of the system have to be computed for each joint of the robot. Then those motions can be propagated to the next joint iteratively. Propagating the motions requires a computationally consuming change of coordinate systems from one joint to the next. To move the gripper in [figure 2.4](#) requires four changes of coordinates to go from the base of the arm to the gripper. The coordinate transformations often have computational singularities, causing the equations to produce an illegal divide by zero operation. It can take a programmer weeks to manually determine the coordinate transforms and identify singularities in order to reprogram a manipulator for a new task.

TEACH PENDANT

One simplifying solution to the computational complexity is to make the robot rigid at the desired velocities, thereby reducing the contribution of the dynamics terms. This eliminates having to compute the terms for overshooting and oscillating. However, a robot is often made rigid by adding more material and thus making it heavier. The end result is that it is not uncommon for a 2,000 kg robot to be able to handle only a 200 kg payload. Another simplifying solution is to avoid the computations in the dynamics and kinematics and, instead, have the programmer use a teach pendant. Using a *teach pendant* (which often looks like a joystick or computer game console), the programmer guides the robot through the desired set of motions. The robot remembers these motions and creates a program from them. Teach pendants do not mitigate the danger of working around a 2,000 kg piece of equipment. Many programmers have to direct the robot to perform delicate tasks and have to get close to the robot in order to see what the robot should do next. This puts the programmer at risk of being struck by the robot should it hit a singularity point in its joint configuration or if the programmer makes a mistake in directing a motion. You don’t want to have your head next to a 2,000 kg robot arm if it suddenly spins around!

AUTOMATIC GUIDED VEHICLES

Automatic guided vehicles, or AGVs, are intended to be the most flexible conveyor system possible: a conveyor which does not need a continuous belt or roller table. Ideally, an AGV would be able to pick up a bin of parts or manufactured items and deliver them as needed. For example, an AGV might

receive a bin containing an assembled engine. It could then deliver it automatically across the shop floor to the car assembly area where an engine was needed. As the robot returned, it might be diverted by the central computer and instructed to pick up a defective part and take it to another area of the shop for reworking.

However, navigation, as will be seen in chapter 10, is complex. The AGV has to know where it is, plan a path from its current location to its goal destination, and avoid colliding with people, other AGVs, and maintenance workers and tools cluttering the factory floor. Navigation proved too difficult to reliably accomplish, especially for factories with uneven lighting (which interferes with vision) and lots of metal (which interferes with radio controllers and onboard radar and sonar). Various solutions converged on creating a trail for the AGV to follow. One method was to bury a magnetic wire in the floor for the AGV to sense. Unfortunately, changing the path of an AGV required ripping up the concrete floor. This did not help meet the flexibility needs of modern manufacturing. Another method was to put down a strip of photochemical tape for the vehicle to follow. The strip was, unfortunately, vulnerable, both to wear and to vandalism by unhappy workers. Regardless of the guidance method, in the end, the simplest way to thwart an AGV was to place something on its path. If the AGV did not have range sensors, then it would be unable to detect an expensive piece of equipment put deliberately in its path or a person standing in the way. A few costly collisions usually led to the AGV's removal. If the AGV did have range sensors, it would stop for anything. A well-placed lunch box could block the AGV for hours until a manager happened to notice what was going on. Even worse from a disgruntled worker's perspective, many AGVs would make a loud noise to indicate the path was blocked. Imagine constantly having to remove lunch boxes from the path of a dumb machine making unpleasant siren noises.

LUDDITES

From the first, robots in the workplace triggered a backlash. Many human workers felt threatened by a potential loss of jobs, even though the jobs being mechanized were often menial or dangerous. The backlash was similar to that of the Jacquard mechanized weaving loom during the Industrial Revolution. Prior to the industrial revolution in Britain, wool was woven by individuals in their homes or in collectives as a cottage industry. Mechanization of the weaving process changed the jobs associated with weaving and the status of being a weaver (it was no longer a skill) and required people to work in a centralized location (like having your telecommuting job terminated). Weavers, called *Luddites* after their leader Ned Ludd, attempted to organize and destroyed looms and mills. After the escalating violence in 1812, legislation was passed to end worker violence and protect the mills. The rebelling workers were persecuted. While the Luddite movement may have been motivated by a quality-of-life issue, the term is often applied to anyone who objects to technology, or "progress," for any reason.

The backlash to robots occurred, though not so violently. An engineer reported that on the first day a HelpMate Robotics cart was used in a hospital, it was discovered pushed down the stairs. Future models were modified to have some mechanisms to prevent malicious acts.

BLACK FACTORY

Despite the emerging Luddite effect, industrial engineers in the economic powers began working for a *black factory* in the 1980s. A black factory is a factory that has no lights turned on because there are

no workers. Computers and robots were expected to allow complete automation of manufacturing processes, and courses in “Computer-Integrated Manufacturing Systems” became popular in engineering schools.

But two unanticipated trends undermined industrial robots in a way that the Luddite movement could not. First, industrial engineers did not have experience designing manufacturing plants with robots. Often industrial manipulators were applied to the wrong applications. One of the most embarrassing examples was the IBM Lexington printer plant. The plant was built with a high degree of automation, and the designers wrote numerous articles on the exotic robot technology they had cleverly designed. Unfortunately, IBM had grossly overestimated the market for printers, and the plant sat mostly idle. While the plant’s failure was not the fault of robotics per se, it did cause many manufacturers to form a negative view of automation in general.

The second trend was the changing world economy. Customers were demanding “mass customization.” Manufacturers who could make short runs of a product tailored to each customer on a large scale were the ones making the money. (Mass customization is also referred to as “agile manufacturing.”) However, the lack of adaptability and difficulties in programming industrial robot arms and changing the paths of AGVs interfered with rapid retooling. The lack of adaptability, combined with concerns over worker safety and the Luddite effect, served to discourage companies from investing in robots through most of the 1990s.

While the industrial manipulator path started with robots as tools, it is now moving to robots as agents. Kiva Systems revolutionized AGVs in late 2000 by treating the AGVs as distributed agents coordinated by a centralized software agent. The Kiva Systems approach went beyond robot navigation; instead the central system planned what materials to bring where and learned what materials were used more frequently and needed to be cached on robots placed closer to the point of assembly. Later Kiva Systems was bought by Amazon, one of its earliest customers.

2.5 Mobile Robots

The mobile robot path of development focused from the beginning on robots as agents, not tools. While the rise of industrial manipulators and the engineering approach to robotics can, in some measure, be traced to the nuclear arms race, the rise of the AI approach can be said to have started with the space race. On May 25, 1961, spurred by the success of the Soviet Union’s Sputnik space program, President John F. Kennedy announced that United States would put a man on the moon by 1970. Walking on the moon was just one aspect of the ramifications of space exploration. There were concerns about the Soviets setting up military bases on the Moon and Mars and economic exploitation of planetary resources.

At the time, it was expected that there was going to be a time lag of almost a decade before humans from the USA would go to the Moon. And even then, it would most likely be with experimental spacecraft, posing a risk to the human astronauts. But unless advances were made in materials, the bulk of spacesuits would make even trivial tasks difficult for astronauts to perform. [Figure 2.5a](#) shows astronaut John Young on Apollo 16 collecting samples with a lunar rake. The photo shows the awkward way the astronaut had to bend his body and arms to complete the task.

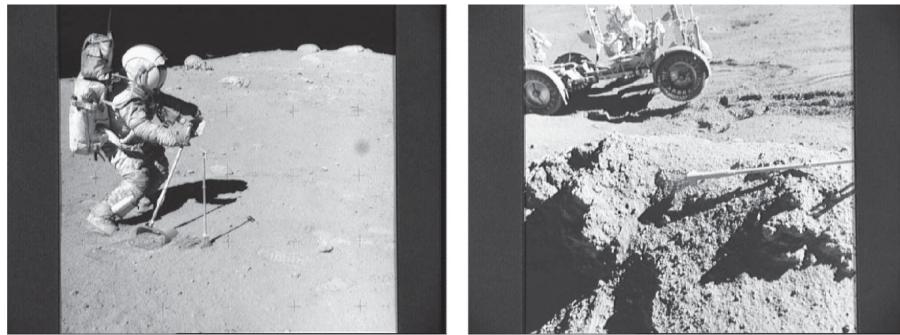


Figure 2.5 Motivation for intelligent planetary rovers: a.) Astronaut John Young awkwardly collecting lunar samples on Apollo 16, and b.) Astronaut Jim Irwin stopping the lunar rover as it slides down a hill during the Apollo 15 mission. (Photographs courtesy of the National Aeronautics and Space Administration.)

Planetary rovers were a possible solution, either to replace an astronaut or to assist. Of course, rover technology had limitations. Because of the time delays, a human would be unable to safely control a rover over the notoriously poor radio links of the time, even if the rover went very slowly. Therefore, it would be desirable to have a robot that was autonomous. One option would be to have mobile robots land on a planet, conduct preliminary explorations, conduct tests, and so forth, and radio back the results. These automated planetary rovers would ideally have a high degree of autonomy, much like a trained dog. The robot would receive commands from Earth to explore a particular region. It would navigate around boulders and not fall into canyons and traverse steep slopes without rolling over. The robot might even be smart enough to regulate its own energy supply, for example, by making sure it was sheltered during the planetary nights and to stop what it was doing and position itself for recharging its solar batteries. A human might even be able to speak to it in a normal way to give it commands.

Getting a mobile robot to the level of a trained dog immediately presented new issues. Just by moving around, a mobile robot could change the state of the world—for instance, by causing a rock slide. [Figure 2.5b](#) shows astronaut Jim Irwin rescuing the lunar rover during an extravehicular activity (EVA) on Apollo 15 as it begins to slide downhill. Consider that, if an astronaut has difficulty finding a safe parking spot on the moon, how much more challenging it would be for a rover to accomplish the same task working by itself. Furthermore, an autonomous rover would have no one to rescue it should it make a mistake.

Consider the impact of uncertain or incomplete information on the performance of a rover that did not have any form of onboard artificial intelligence. If the robot was moving based on a map taken from a telescope or an overhead command module, the map could still contain errors or be at the wrong resolution to see certain dangers. In order to navigate successfully, the robot has to sense the state of the world and compute its path with the new data or risk colliding with a rock or falling into a hole. What if the robot broke or something totally unexpected happened or all the assumptions about the planet were wrong? Ideally, the robot should be able to diagnose the problem, determine a workaround or alternative, and attempt to continue to make progress on its task. Robots seemed at first like an interim solution to putting humans into space but quickly became more complicated.

Clearly, developing a planetary rover and other robots for space was going to require a concentrated, long-term effort. Agencies in the USA, such as NASA’s *Jet Propulsion Laboratory (JPL)* in Pasadena, California, were given the task of developing the robotic technology that would be needed to prepare the way for astronauts in space. These agencies were in a position to take advantage of the outcome of the *Dartmouth Conference*. The Dartmouth Conference was a gathering hosted by the Defense Advanced Research Projects Agency (DARPA) in 1955 of prominent scientists working with computers or on the theoretical foundations of computers. DARPA was interested in hearing what the potential uses for computers were. One outcome of the conference was the term “artificial intelligence”; the attending scientists believed that computers might become powerful enough to understand human speech and to duplicate human reasoning. This, in turn, suggested that computers might mimic the capabilities of animals and humans sufficiently to enable a planetary rover to survive for long periods with only simple instructions from Earth.

As an indirect result of the need for robotics converging with the possibility of artificial intelligence, the space program became one of the earliest proponents of developing AI for robotics. NASA also introduced the notions that AI robots would be mobile rather than bolted to a factory floor and that AI robots would have to be able to integrate all forms of AI (understanding speech, planning, reasoning, representing the world, learning) into one program—a daunting task which has not yet been achieved.

SHAKY

The first AI robot was *Shakey*, funded by DARPA. Shakey was built by SRI, then the Stanford Research Institute, between 1967 and 1969 and remains a remarkable, comprehensive foray into physically situated intelligence. Shakey, shown in [figure 2.6](#), was about as tall as a small person. It used a “sense-plan-act” cycle. The robot was given a goal (e.g., go to room B) and a starting location (e.g., room A). Then it would use a TV camera and sonars to *sense* the world, in this case, a large room populated with large, brightly colored polygonal obstacles. The imagery would be analyzed by the computer to locate the obstacles, and a motion path *plan* would be generated that would move the robot to the goal without hitting the obstacles. Then the robot would *act* by moving a few feet along the path and then stop and repeat the cycle of sensing, planning, and acting. The entire process for moving across a room could take hours. The robot earned its name “Shakey” because it lurched when it moved, causing the large radio antenna on top to rock back and forth and the whole robot to shake.

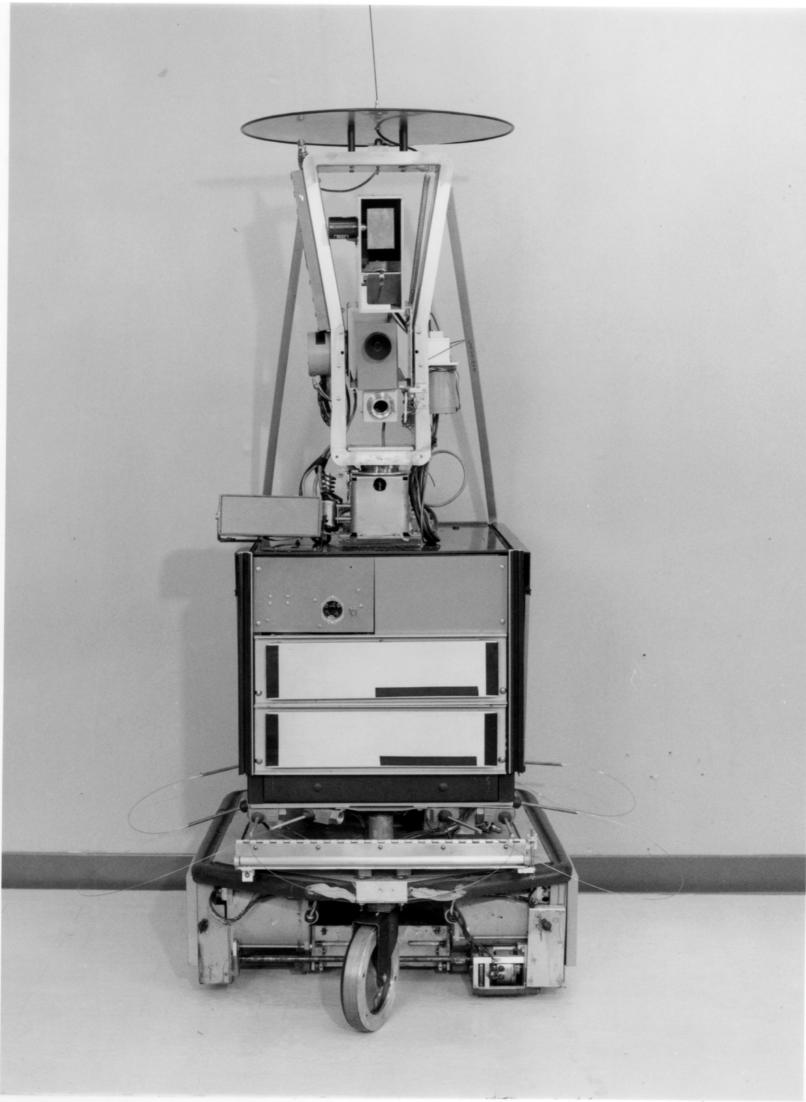


Figure 2.6 Shakey, the first AI robot. (Photograph courtesy of SRI.)

The Sojourner robot (shown in [figure 2.7](#)) which explored Mars from July 5 to September 27, 1997, until it ceased to reply to radio commands, began to realize NASA's vision of rovers that had started in the 1960s when robots were posed as interim solutions for manned exploration. While Sojourner was often teleoperated, it was capable of navigating without intervention to a rock selected in an image. The robot would determine whether it could crawl over a rock in its way or should go around. However, navigation is just one aspect of intelligence, and Sojourner illustrates why problem solving is important. When the petals of the Mars lander containing Sojourner opened, an airbag was in the way. The solution? The ground controllers sent commands to retract the petals and open them again, beating down the airbag. That type of problem solving is extremely difficult given the current capabilities of

AI, though NASA's Deep Space One probe has sophisticated fault identification, diagnosis, and recovery problem solving capabilities.⁶² The Mars Exploration Rovers, Spirit and Opportunity, arrived on Mars in 2004 and have the advantage of upgrades in AI. Opportunity, the remaining functional robot, can perform autonomous image understanding mission planning to determine what rocks to explore, what data to gather, and to perform path planning to move to the rock and reach out the arm for sampling.⁹⁷

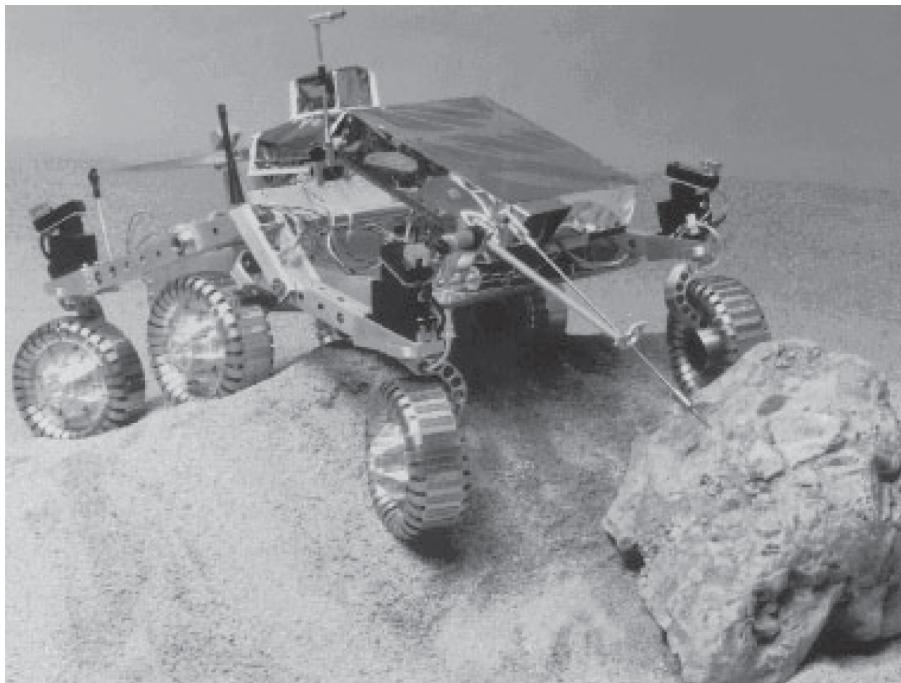


Figure 2.7 Sojourner Mars rover. (Photograph courtesy of the National Aeronautics and Space Administration.)

The iRobot Roomba® vacuum cleaner seen in figure 2.8 marked a milestone in the sustained commercialization of intelligent robots and also provided the public with an alternative view of how robots would look and act. Initial investments in cleaning robots had aimed at the janitorial services, rather than the individual consumer. Cleaning robots were typically large golf cart sized platforms that used temporary landmarks placed on walls so that the robot could accurately triangulate its location, much in the same way a GPS device using satellite signals. The robot would follow a optimum, back-and-forth sweep through a hallway or foyer, vacuuming or mopping just like a human would. Human workers would clear any trashcans or obstacles out of its path and then go clean individual offices. Given the high rate of turnover in janitorial industry, roboticization should have been a viable alternative. But early models had a few well-publicized incidents where a robot would crash through a wall due to an error in placement of a landmark or measurement of the room. The robot was mobile but not sufficiently intelligent to sense a wall or to notice when it was missing a landmark. Research began to focus on how to accurately sense an entire room or area and compute the optimal path to clean it. Progress was hampered by the size and expense of incorporating accurate range sensors, which cost

more than \$100,000 per sensor, into the robot. Computer vision was unable to extract depth reliably from imagery unless the lighting was constrained. The Roomba took advantage of principles from the behavioral robotics movement that will be discussed in later chapters to create a robot that sensed walls and obstacles like an insect and instead of planning a path, it used a random pattern of movement like a goat grazing. The expectation that a robot vacuum cleaner would look like a standup vacuum cleaner or would have a humanoid robot pushing it, à la “The Jetsons,” was dashed.



Figure 2.8 The iRobot Roomba® vacuum cleaner that uses animal-like intelligence to adapt to rooms.



Figure 2.9 Teledyne Ryan Firebee UAV (target variant, IDF designation Shadmit) at Muzeyon Heyl ha-Avir, Hatzirim airbase, Israel. 2006. (Photograph courtesy of WikiCommons under the Creative Commons Attribution-Share Alike 3.0 Unported license.)

2.6 Drones

DRONE

The drone development path is similar to the industrial manipulator path, where robots are tools. World War II created the need for a class of preprogrammed land, sea, and aerial mobile targets known as *drone*. The Ryan Firebee unmanned aerial drone was specified in 1948 and built in 1951.¹⁸² Drones provided more realistic targets to practice with or to evaluate new target acquisition systems. Aerial drones, particularly the Lightning Bug variant of the Firebee class of platforms, began carrying imaging sensors and were extensively used by the United States in the Vietnam War in the 1960s, but drones remained tools that followed simplistic preprogrammed routes or were teleoperated. Drones have remained in use since then, but the term drone has come to mean any unmanned aerial system rather than an unmanned system being used as a target. With the advent of the Predator, which did not look like a typical aircraft, and its subsequent weaponization during the 2003-2012 Iraq War, drones seem like a new, and possibly threatening, technology.

Drones contributed little to the evolution of artificial intelligence, remaining on a relatively isolated path of increasing platform sophistication rather than increasingly intelligent software. Drones have exploited engineering advances increasing mobility and range, advances in aerodynamics of aerial vehicles, lowered cost and increased reliability, and incorporation of GPS-enabled capabilities. The use of the term “drone” has so frequently been equated by the popular press with *any* unmanned system independent of whether it has machine intelligence, that saying “drone” to an AI researcher is likely to elicit a roll of the eyes or a visible wince.

2.7 The Move to Joint Cognitive Systems

The shifts in the industrial manipulator and drone paths starting in late 2000 have seen intelligent robots being designed as joint cognitive systems. There are at least three reasons for the move to joint cognitive system design for all applications. Each reason stems from the fundamental insight that the value of a robot depends on how well it works within the larger socio-technical system, not just how well it performs mechanically or electronically.

One reason is that robots are now being used for tasks that involve humans. Autonomously driven cars use a strategy of mixing what the human driver is doing and responsible for and what the car is doing and responsive for. Humans typically generate goals, such as selecting a parallel parking spot or a road to travel on, while the robot acts like a service animal and helps carry out the goal. Eldercare robots are expected to provide services to senior citizens and do so in a friendly, comforting way. Telecommuting is allowing users to project themselves into robot bodies, creating both a challenge in making it easy for the user to operate and for making the robot act and behave appropriately around coworkers present in the workplace.

A second reason for the movement to joint cognitive systems is that recent applications of robots have led to barriers and frustrations. For example, the rapid adoption of unmanned aerial vehicles by the US Department of Defense was motivated in part by the expectation that UAVs would reduce manpower; instead the clumsy tool automation schemes actually increased overall manpower.¹⁴⁴ A recent study of failure rates of robots used in disasters found that more than 50% of the failures were due to human error;¹⁴¹ this level of human error means the robots must be redesigned around the limitations of operators. The industrial manipulator market has been slowly shifting from robots as preprogrammed tools to more sensing-capable agents, but perhaps the biggest change has come in the form of *Baxter*, a factory robot that learns from the human and communicates whether it can perform the task or has a problem completing the task.

A third reason is the awareness that every conceivable application of a robot has a human in the system somewhere. As discussed above, autonomous driving and eldercare involve the human as a team member. The revolutionary value-added aspect of human-centered robots like Baxter is the human-robot interface which makes rapid retasking of the intelligent robot much faster and more trustworthy. Even planetary rovers have scientists and engineers who are consuming data provided by the robot agents and refining goals and plans based on the surprising new knowledge. The success of small UAVs, such as the DJI Phantom® and Parrot AR.Drone, over other equally capable and inexpensive brands is due to the ease of their use by a novice.

2.8 Summary

The field of AI robotics is distinct both historically and in scope, from industrial robotics and drones. The vast majority of robots in use, starting with the nuclear industry and drones during World War II, were designed as *tools*, with limited scope and utility. The robots-as-tool approach continued with industrial manipulators, AGVs, and drones; these had little artificial intelligence. Cybernetics was an early movement towards intelligent robots by using biological control principles, but the term began to be synonymous with general control principles and thus is rarely used in AI discussions. The more speculative robots for planetary rovers were conceived of as independent agents with their own agenda;

planetary rovers motivated fundamental advances in artificial intelligence. The resulting fundamental advances in robots as agents have begun to influence industrial robotics, most notably the Kiva Systems (now Amazon Robotics®) success in materials handling. Commercial marketplace drivers, such as ease of use, rapid reconfigurability, and user acceptance are driving all types of robots towards joint cognitive system design. The sizable manpower requirements and high human error rate in unmanned systems are motivating designs that incorporate agency for increased functionality and are expressed as joint cognitive systems in order to reduce manpower and errors.

2.9 Exercises

Exercise 2.1

Describe the difference in intelligence needed for a robot to serve as a tool, an agent, or a joint cognitive system.

Exercise 2.2

Define cybernetics.

Exercise 2.3

Search the Web for information on the early Kiva Systems (now Amazon Robotics®). Which of the seven areas of artificial intelligence does it use?

Exercise 2.4

Search the web for information on the Baxter robot. Which of the seven areas of artificial intelligence does it use?

Exercise 2.5

Is there a robot application in which a joint cognitive system design philosophy would not be useful?

2.10 End Notes

AGVs in hospitals

A 2008 study of AGVs in hospitals by Mutlu and Forlizzi¹⁵⁵ suggests that the history of poor design persists. In this case the robot carts were able to move around obstacles in the hall—that were predominately people. However, the type of ultrasonic range sensors the carts used were coarse and could not detect thin objects, thus the robot was able to avoid healthy people but often sideswiped, or came very close to, crutches or wheelchairs which have a thin profile. When the robot reached the nursing station on a ward, if the nurses did not immediately acknowledge the cart, it would sound a loud siren to alert the nurses that it was there (as if a large robotic cart was easily missed). The staff in the Intensive Care Unit found that they had to stop critical care activities in order to shut off the loud siren and prevent it from disturbing the ward.

For the Roboticist's Bookshelf

The Human Use of Human Beings by Norbert Wiener, Houghton Mifflin Harcourt, 1950, is an astonishingly cogent and timeless argument for robots.

Managing Martians by Donna Shirley, Broadway Books, 1999, gives a behind-the-scenes look at the development of NASA's Sojourner rover.

3

Automation and Autonomy

Chapter Objectives:

- Describe *automation* and *autonomy* in terms of a blending of four key aspects: generation or execution of *plans*, deterministic or non-deterministic *actions*, closed-world or open-world *models*, and signals or symbols as the core knowledge *representation*.
- Discuss the *frame problem* in terms of the *closed-world assumption* and *open-world assumption*.
- Describe *bounded rationality* and its implications for the design of intelligent robots.
- Compare how taking an automation or autonomy approach impacts the programming style and hardware design of a robot, the functional failures, and human error.
- Define the *substitution myth* and discuss its ramifications for design.
- List the five trade spaces in human-machine systems (*fitness*, *plans*, *impact*, *perspective*, and *responsibility*), and for each trade space give an example of a potential unintended consequence when adding autonomous capabilities.

3.1 Overview

This chapter addresses autonomy and what makes it different from building and controlling tools. In particular, the chapter will try to answer *what is the difference between automation and autonomy?* As seen in the previous chapter, automation historically concentrated on creating tools for industrial manipulation. Industrial manipulators measured success in terms of how well the robot could *execute* preplanned precise, repetitious actions or sequences. Automating preplanned motions for an industrial manipulator typically uses *deterministic* algorithms and relies on a well-understood environment that is completely and precisely modeled in advance.

On the other hand, those interested in developing mobile robots measured success in terms of how well the robot acted as an agency; this chapter introduces important concepts in agency. Research and development concentrated on a robot that could *generate* plans that adapted the appropriate actions or sequences to the environment based on sensed data. It is important to emphasize that autonomous capabilities, distinct from automation, use *non-deterministic* algorithms in an open world to overcome

potential omissions and errors in modeling the world. While non-deterministic systems are harder to test, this does not mean that robots can spontaneously exceed their programmed functions; the chapter introduces the principle of *bounded rationality* which states that a robot will not do anything it was not programmed to do. Often the generation of plans requires manipulating *symbols* (e.g., red car, person walking, open door, etc.) as well as *signals* (e.g., lidar point cloud, vibration, etc.), requiring advanced perceptual processing and reasoning to translate sensor data into a semantic understanding of the objects and activities going on in the world.

This chapter also addresses a more fundamental question: *Why does it matter that there is a difference between autonomy and automation?* The distinction between automation and autonomy may appear subtle but it has major ramifications: it affects the style of programming a robot, the hardware design of building a robot, and the kinds of failures produced by a robot. Ill-considered designs of autonomous capabilities often produce failures stemming from the *substitution myth*: that a robot can replace a human with no impact on the larger socio-technical organization or resilience of the system.

Because it does matter that automation is different from autonomy, this determination raises questions: *What are the advantages of autonomy over automation? Can you tell me when to use one over the other? How much autonomy do I need?* A Holy Grail of unmanned systems has been a taxonomy or rating system that can be used to prescribe the “right amount” of autonomy for a particular robot and application. Unfortunately, robot design is just that: a design problem with complex tradeoffs. In most cases, artificial intelligence offers techniques for overcoming the negative aspects of a tradeoff, but these autonomous capabilities must be incorporated into the system design from its inception. Essentially a designer needs to consider the autonomous capabilities needed to accomplish the mission, then consider the additional autonomous capabilities needed to ensure that the robot really does accomplish the mission, and finally make sure that the requisite sensors and processors are incorporated into the platform design.

3.2 The Four Sliders of Autonomous Capabilities

Rather than try to define a particular robot capability as being either “automation” or “autonomy,” it may be more useful to think of a capability as drawing from the spectrum of techniques representing four key aspects: *plans*, *actions*, *models*, and *knowledge representations*. One way to visualize a capability is as a set of sliders over the spectrum for each aspect shown in [figure 3.1](#). Note that each of the aspects associated with automation and autonomy contribute to the overall capability. An “autonomous robot” may have aspects associated only with the automation side of the spectrum or only from the autonomous side, but, in reality, most robot systems programmed with artificial intelligence methods have a mixture of both.

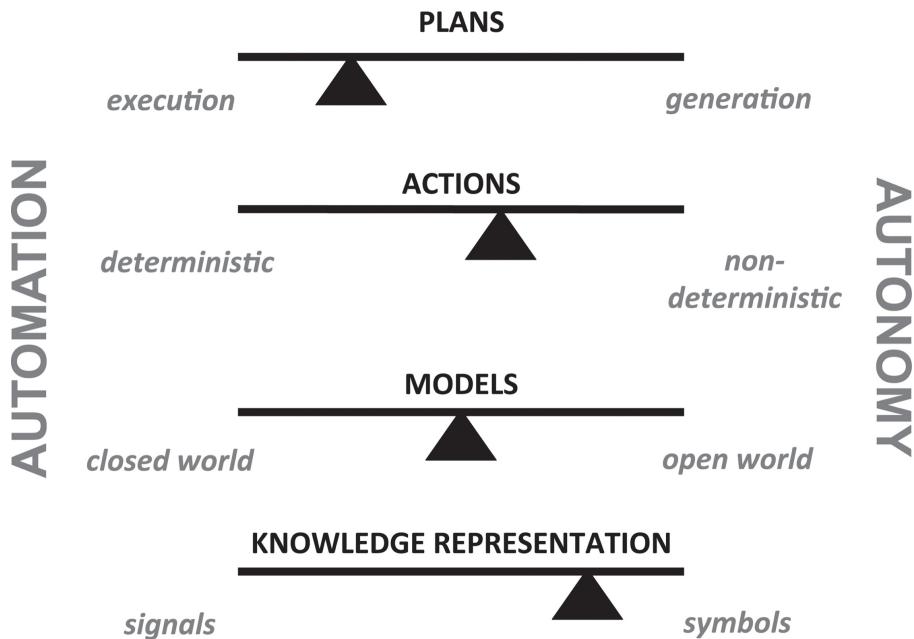


Figure 3.1 Visualizing how the four aspects of automation and autonomy combine to create the DNA for a particular intelligent system.

3.2.1 Plans: Generation versus Execution

Generating and executing a plan are closely related, as the robot cannot execute a plan unless one has been generated. In factory automation applications, the plan may be constructed by a human using a teach pendant to show the robot what the person wants it to do. The computing effort and intelligence is therefore not in generating the plan, but restricted to the robot's ability to perform, or execute, those actions exactly the same way on each pass and as fast as possible. In contrast, autonomous applications allow the robot to construct or adapt the plan itself because the variations in the mission or the environment or even wear and tear in the robot's mechanical joints precludes it being able to create a complete, full-proof plan in advance. Planning has been one of the most active areas in the development of artificial intelligence, and the community has proven results for mission planning, contingency planning, optimal resource allocation and scheduling, and determining when to use highly optimized plans or more flexible generalized plans. Control approaches view executing a plan as the hard or interesting problem; artificial intelligence approaches view generating the plan as where the effort should be focused.

3.2.2 Actions: Deterministic versus Non-deterministic

The spectrum of deterministic and non-deterministic actions merits a more detailed discussion, given that a 2012 study by the Defense Science Board that found that this dichotomy has been a stumbling block in creating effective test and evaluation procedures.¹⁴⁴ *Deterministic* and *non-deterministic* are terms used in computer science to distinguish types of algorithms—usually applied to finite state

automata (FSA).¹¹⁶ In a deterministic FSA, if the robot is in a given state and receives a specific set of inputs, there is only one possible output. In a non-deterministic algorithm, there are multiple possible outputs, and the chosen output depends on some other factor or event.

Non-determinism can result from the nature of the inputs: if there are many inputs, the inputs are from noisy sensors, or the range of values provided by sensors is very large, then it becomes computationally intractable to explicitly express the output for all the combinations of inputs. Another source of non-determinism is the condition of the hardware. If the robot is running out of power, it may not move as far out of the way of an obstacle. For example, the robot was supposed to move 1m to the right but moved only 0.75 m. If the robot has multiple processes operating concurrently, as is likely, then the timing between the processes introduces non-determinism. For example, a sensor processing algorithm in Trial 1 may update a buffer a second or two after a motor-control algorithm accesses the buffer to control the next move. On Trial 2, the buffer may be updated earlier. Thus, in the two trials, the robot has slightly different motions, just like people who rarely step in exactly the same place every time they walk down a hall. People have a pattern of walking down a hall and can be counted on to successfully walk down a hall without hitting other people or walls. Non-determinism is similar: a robot can go down a hallway and not hit a person, though whether the robot swerves to the left or right may depend on when it sensed the person, any sensor noise that might make it think it was closer to a wall than it was at that moment and thus it should swerve to the left, not the right, and any other sensory inputs or timing of internal computation.

The examples above suggest that deterministic and non-deterministic systems should be tested differently. Guidance, navigation, and control have historically relied on deterministic algorithms, leading engineers to create elegant mechanisms for provably correct, complete control guaranteed to generate a specific set of outputs. These traditional methods are difficult to apply to artificial intelligence algorithms because expressing a system deterministically becomes harder as the robot becomes more complex and consists of many actions working in parallel. Although in theory, a non-deterministic finite state automaton can be converted to a deterministic one, in practice, the size and complexity of the inputs, outputs, and timing considerations prevent this from being a viable option. The scale hampers the designer's ability to visualize or test for all possible situations, which hinders acceptance of intelligent robots.

Autonomous capabilities for AI robotics have historically been non-deterministic, and performance has been measured as an average or statistical probability. An example is the work of Pattie Maes and Rod Brooks¹¹⁹ in which a six-legged robot, Genghis, learned to walk. Genghis' legs have to move at the same time and slight variations in the terrain or the hardware can change the maneuverability of the platform. Rather than try to encode all possible conditions, Genghis essentially woke up with two urges and thirteen behaviors. It had the urge to move forward, as measured by a trailing wheel, and not to be lying on its belly, as measured by two touch sensors. Twelve simple behaviors were for swinging a leg forward or backward and one was for balancing. Genghis learned to walk, on average, in less than two minutes by trying different leg movements and receiving positive and negative feedback from the sensors. But the robot's overall actions were non-deterministic as it did not produce exactly the same leg movements each time it woke up because the conditions were slightly different.

Testing non-deterministic algorithms is an open research question. Existing testing techniques often rely on statistically sampling the space of inputs. Many users, especially military users, have trouble

with a robot being non-deterministic because there is not 100% certainty that the robot will do exactly the same thing every time even though this non-determinism is part of how the robot adapts to the environment, computational load, or condition of its hardware. User discomfort is often compounded if the robot cannot display why it is doing what it is doing or if it cannot meet its goals, which is a common human-robot interaction design error discussed in chapter 13. Even if the robot meets the goal, that it is not completely predictable as to *how* it meets the goal is disturbing to some. The AI robotics philosophy is “if you want a robot to be as smart as an insect, dog, or human, then you have to accept that the robot will be as non-deterministic as an insect, dog, or human.”

3.2.3 Models: Open- versus Closed-World

WORLD MODEL

In order for a computer system to understand the world, there has to be a computationally relevant representation called a *world model*. If the robot only has navigational capabilities, the world model may consist of a map of which areas are occupied and which are empty. A representation of the world is not necessarily spatial. If the robot interacts with other agents, such as is seen in social robotics, it may need to represent what it thinks are the beliefs, desires, and intentions of the other agents. The world model can be a collection of world models; a chemical plant robot might have a map-like representation for spatial reasoning, a model of all the types of valve handles that would be in a chemical plant and how to turn them, and a belief model for social interactions with other workers.

A world model may be preprogrammed into a robot, may be learned by the robot, or some combination of both. The preprogrammed portions of a world model are often maps or lists of objects or conditions that the robot can expect to encounter.

CLOSED-WORLD ASSUMPTION

World models are classified as being closed-world or open-world. The *closed-world assumption* says that everything possible is known a priori, that there are no surprises. In formal logics, this means that any object, condition, or event that is not specified in the database is false.

In the best-selling book and movie, *Jurassic Park*, the intelligent monitoring system designed to keep track of the dinosaurs failed because of the closed-world assumption. The system did not notice that the dinosaur population was increasing because it was assumed that the dinosaurs could not reproduce; as a result, the object recognition system would alert only if there were fewer dinosaurs than expected and never alert if there were more.

OPEN-WORLD ASSUMPTION

The algorithms for a robot operating under the *open-world assumption* assume that the list of possible states, objects, or conditions cannot be completely specified. These algorithms may use machine learning to add new objects or events to the model. If the world is being represented with a formal logic, a more advanced logic is used which allows the system to retract assertions about the world when an object is moved or if something changes; otherwise, correct logical inference could not be guaranteed.

FRAME PROBLEM

Related to the closed-world assumption is the frame problem. The *frame problem* refers to the problem of correctly identifying what is unchanging in the world, and thus does not require constant updating, thereby reducing computation. A system that requires representing and reasoning over all the items or states of the world will likely be unable to manage the resulting computational complexity. For example, consider trying to list all the items in a room (walls, ceilings, desk, chair, etc.) The list gets very long very quickly. One of Shakey's contributions to AI was that it helped uncover the frame problem. The robot accrued significant computational costs because it was programmed to find and identify everything in the room with each update of its camera.

Another aspect of the frame problem is determining what is relevant for a particular capability. In *Jurassic Park*, the programmer had used the closed-world assumption that the dinosaurs could not reproduce to reduce the requisite computational resources; otherwise the observation program would have to look for all dinosaurs in all imagery. In Shakey's case, if the robot is trying to navigate around a desk, is it necessary to identify the object blocking its path as a desk or a chair? On the other hand, if the robot is searching for a desk, then identifying desks is important.

The frame problem means a robot designer has to be aware of how assumptions about the world impact the adaptability of the robot and what the role of monitoring the validity of these assumptions is. The cognitive robotics field, a small group within AI robotics, addresses the frame problem by exploring temporal reasoning and temporal logic. However, this book concentrates on mainstream behavioral robotics, which tries to avoid the frame problem by minimizing the use of world models.

The dangers of the closed-world assumption and the associated frame problem impact both engineering and AI approaches to robotics. The difference between the two approaches is that a good control system is meticulously designed to capture everything that needs to be in the world model, while an AI system is designed to adapt to an imperfect world model. In practice, both control-theoretic and AI robots may use a closed-world model. Fortunately, an open-world model does not mandate sophisticated machine learning of new objects or states; as will be seen in the next chapter, animals use simple open-world models.

3.2.4 Knowledge Representation: Symbols versus Signals

A distinction can be made between automated and autonomous capabilities based on the form of information that the robot processes, either signals or symbols. Automation connotes that the robot is responding to signals or raw data, for example, move until the encoder signals that location L1 has been reached, activate grasp movements, move to location L2, release. Recalling the seven areas of artificial intelligence from chapter 1, artificial intelligence generally operates over data that have been transformed into information or symbols, for example, move to the COFFEE POT, grasp, move to MY COFFEE CUP and ignore YOUR COFFEE CUP. In robotics, it is often difficult for a robot to reliably recognize objects or understand a situation, and thus there is a barrier to converting raw sensor signals into labeled objects or areas. Fortunately, in AI robotics, the “lower” level autonomous behaviors described in chapter 4 do not use symbols and instead work with sensor signals called affordances while concurrently the “upper” level deliberative capabilities work with symbols.

3.3 Bounded Rationality

Autonomy may induce fear or distrust on the part of users, despite the principle of bounded rationality. Distrust often arises from two sources: the political definition of autonomy and its non-deterministic nature. Autonomy is defined as the quality or state of being self-governing, of having self-directing freedom and especially moral independence.¹²⁷ In everyday usage, this means people are free to do as they choose without having politically imposed boundaries. Many people worry that “autonomy” means the robot has a similar self-actualized freedom of choice and thus will be unpredictable or will exert initiative beyond what was intended by the designer. In AI robotics usage, the term autonomy is much more limited. The use of the term “autonomy” is derived from “self-governing” portion of the definition, and stems from the invention of the self-governing flyball governor in the 1700s. The flyball governor would spin with the amount of steam being produced by a steam engine. The faster the spin, the higher a set of balls would fly. If the balls rose to a specified height, too much steam was being produced and this would trip the production system thus preventing a boiler explosion. A flyball governor is not politically autonomous, just mechanically autonomous.

BOUNDED RATIONALITY

One of the founders of AI, Herbert Simon, showed that the decision-making capability of all agents, human or artificial, is limited by how much information they have, their computational abilities, and the amount of time available to make a decision.¹⁹⁵ This principle of *bounded rationality* means that, while a robot may dynamically adapt or replan to overcome the occurrence of unmodeled events in the open world, it cannot go beyond what it was programmed to do. Note that while Genghis learned the new capability of walking, it did not exceed its computational bounds; it could only learn combinations of its 13 initial behaviors.

Bounded rationality and understanding a robot to be self-governing in the sense of a flyball governor may not help people trust robots. A major barrier in the acceptance of autonomous capabilities is the question *How can you trust a non-deterministic system?* Given the difficulties of visualizing and testing non-deterministic algorithms, it is easy for designers or users to be surprised by the actions the robot takes to maximize its success in accomplishing a specific goal or mission.

3.4 Impact of Automation and Autonomy

The previous section described how the artificial intelligence and engineering approaches to machine intelligence represent fundamentally different paradigms. The choice of paradigm in designing an autonomous system impacts the programming style and hardware design for a robot, and the likelihood of certain types of functional failure and human error. Autonomous capabilities can be so primitive as to be indistinguishable from automation, but the differences between the two become more pronounced as the number and sophistication of capabilities increases. Thus the impact of choosing a characteristic from the automation end of the slider spectrum or from the autonomy end will become more pronounced in the future.

3.5 Impact on Programming Style

The programming style for autonomy and even the programming languages can be different. As noted in the previous section, autonomous capabilities are generally computationally non-deterministic. In AI

robotics, the overall response of the robot emerges from the interaction of independent modules rather than is explicitly specified. As will be seen in the remaining chapters of the book, basic functionality for robot control is expressed in modules called behaviors. Behaviors are often programmed in C ++ or another procedural language. More advanced deliberative functions are captured in more sophisticated modules that may run slower than the behaviors and thus are asynchronous with the fast real-time control of the robot. The deliberative functions may be programmed in Lisp, a functional programming language that is especially well-suited for planning and reasoning.

3.6 Impact on Hardware Design

Autonomous capabilities must be considered in the design of the hardware. Recall that autonomy requires sensing in order to adapt to the key elements of the dynamic world. Design has typically focused on enabling robots to act, concentrating on effectors, powers, and wireless communications but ignoring the control and sensing subsystems. As will be emphasized throughout this book starting in chapter 6, intelligent robots should be viewed from an ecological design perspective: What capabilities does the robot need to perform its functions in a particular environment?

Users commonly request a wireless robot to have a built-in return-to-home capability. If wireless communications are lost, the robot would automatically come back to a starting position. However, many ground robots are designed with no onboard computing; all computations are done at the operator's control unit. Without onboard computing, there is no way to add a return-to-home capability. However, not having a computer onboard reduces the power draw of the robot, helps reduce the weight and size of the robot, reduces the cost, and increases the reliability of the robot given that a computer may not survive the rough and tumble environment.

Adding new sensors to an unintelligent robot later may be impossible due to the change in size and weight of the robot, the lack of capacity on the internal electrical and power buses, or even the lack of a surface for mounting. In many types of manned and unmanned aircraft, a human cannot maintain stable flight whereas a computer can react faster and perform nonintuitive but critical movements of multiple control surfaces. But these sensors required for flight stability are not sufficient for autonomous capabilities, such as obstacle avoidance. Recently, as a result of the difficulty of retrofitting sensors in UAVs, there have been efforts to reuse the camera on a robot for capabilities such as obstacle avoidance, leveraging the AI area of computer vision (see chapter 9).

If a human has difficulty in directing a robot due to lack of sensing, neither automation nor autonomy may be able to do better. The Inuktun Mine Cavern Crawler, used for search and rescue at the 2007 Crandall Canyon, Utah, mine collapse, was difficult for a human to control and impossible to add any autonomous capabilities to it.¹⁴¹ The robot platform consisted of two major components: a tracked robot base with a pan-tilt-zoom camera and lights module. The camera module was packaged in a cylindrical mast that would lie flat behind the base while the robot was being lowered through a borehole and then raise up 90 degrees above the robot base. The robot was created in three days with the intent of making the smallest, waterproof robot possible. As a result of design decisions, there was no sensor or mechanical stop for raising the mast due to its size and the additional electronic complexity. As a consequence, if the mast was raised beyond 90 degrees, there was the possibility that it would become stuck in that position and not be able to fold back down and allow the robot to re-enter

the borehole. The robot operators were instructed to count as they raised the mast and never to raise the mast for longer than a count of 10, assuming the mast was always lifted from the neutral flat position. But the robot would frequently have to change the angle of the mast in order for the robot to navigate in the highly confined spaces of a collapse, thereby requiring the operators to mentally maintain an estimate of the mast pose. Without encoders on the mast, there was no reliable and accurate way to automate the rise to stop before 90 degrees from any starting position or create a warning system.

3.7 Impact on Types of Functional Failures

Automation and autonomy have similar failure modes but diverge in types of solutions. Two major failure mode of intelligence in robots are *functional failures*, where the robot does what it was programmed to do but not what was intended, and *failures stemming from the substitution myth* that a machine can replace a human without changing anything else in the system, such as displays, procedures, and training. In theory, a system with autonomous capabilities should have fewer functional failures than an automated system working under a restrictive set of closed-world assumptions. But in practice, a robot with autonomous capabilities may not have the problem solving and reasoning capabilities that would enable it to go beyond a de facto closed world. Cognitive and industrial engineers who study human-machine systems have a long list of the ways in which systems fail. Many of these are attributed to human error by the user of the machine or the robot, but are actually human errors by the designers. A recent study of failure modes in disaster robotics found that about 50% of the failures were attributed to human error, though it was noted that it was unclear what the human could have done differently.¹⁴¹ Autonomous capabilities can be added to improve the robot's performance in these areas or to intelligently assist the user.

3.7.1 Functional Failures

If the robot's expectation of the environment is incorrect in a complex world, it may get the equivalent of "tunnel vision" and miss things. As discussed earlier, in automation of industrial manipulators, the representation of the world may include the assumption that the object to be manipulated, welded, or painted is in a precise place and orientation. If this assumption is violated, such as when a conveyor belt is stopped, then the robot will manipulate, weld, or paint thin air.

A similar phenomenon can happen in more autonomous systems. An early U.S. military autonomous image processing system was considered a major failure during an actual engagement because it could not identify enemy troop movements. Upon further analysis, it was discovered that the object-detection and scene-understanding modules had been correctly identifying troop movements. However, the algorithms for the module typically generated false positives, so the system forwarded the output to another module which would reason about the images using its knowledge of the best practices of military operations. Unfortunately for the artificial intelligence system, in that particular engagement, the enemy troops lost discipline quickly and did not fight using any principles of military doctrine. Thus while the object-detection and scene-understanding modules were accurately identifying troop movements, another module was disregarding the correct identification because of a formalized expectation that troops would never fight in a disorganized way.

Returning to the discussion of closed and open worlds, both the industrial manipulator and the

software agent were operating in a closed world but the image processing system was operating in an open world. The design decision to formalize expectations about troop movements without adding any mechanisms to notice (or display to a human) that the two modules were disagreeing in surprising ways imposed a de facto closed-world assumption on what should have been an open-world system. As a result the design imposition, the overall intelligence of the agent was reduced. Artificial intelligence strives for systems that can work in the open world, but implementations may fall short of those lofty expectations.

3.7.2 Impact on Types of Human Error

SUBSTITUTION MYTH

While automation and autonomy are intended to eliminate or reduce human involvement, these capabilities often introduce human or operator error because any robot is part of a larger human-machine system. “Autonomous” often implies to a user that the robot is more capable than it is, particularly in the subtle ways that make humans so adaptable. This assumption is referred to as the *substitution myth* in cognitive engineering:²²⁰ *the myth that a machine will perfectly replace a human for that particular task.*

HUMAN OUT-OF-THE-LOOP (OOTL) CONTROL PROBLEM

From a larger human-machine perspective, no matter how independent of a human the robot is, if something goes wrong, a human is expected to take over for the robot or to intercede. But it may not be cognitively possible for a human to react fast enough or to comprehend what the robot has been doing and what is going wrong and to determine why and then to correctly solve the problem. This phenomenon has been well-known in cognitive engineering since the 1980s and is generally referred to as the *human out-of-the-loop (OOTL) control problem*.^{125;118;99;101} The human OOTL problem proved deadly in early autopilot systems where commercial airline pilots would be attending to paperwork or other duties when the autopilot would suddenly exit due to the occurrence of a situation it was not programmed for. The pilots might have only a few minutes to determine what had led the autopilot to exit; worse yet, since the autopilot worked well in routine situations, by definition it should exit on its own only if an abnormal event had occurred. The fact that the autopilot exited meant that the situation would be challenging to diagnose. Pilots were often caught unaware and could not make the transfer from what they were currently doing back to a full understanding of what was going on, leading to crashes. The OOTL phenomena is especially pernicious in small unmanned aerial vehicles that fly at lower altitudes where the human operator may have only seconds to diagnose and recover the system.^{171;154}

In general, a human will be able to overcome the OOTL control problem if the system is designed to allow the human to easily visualize the internal state of the robot and to facilitate smooth transfer of control. In the case of manned aviation, autopilots began indicating when the plane was approaching the limits of its control regime, giving the pilots a warning to start focusing on the plane and get ready to take over.

Human-machine studies have shown that the OOTL problem also occurs in factory automation and that it may be more cost effective to have less automation or autonomy. Endsley and Kaber in⁷¹ found that for highly automated factories, it took longer for a person who had not been directly engaged with

the operators (perhaps in an adjacent office doing paperwork) to diagnose and fix a problem that shut down the production line. Time and loss of revenue was reduced if the person was given jobs that required staying involved with the automation; by being involved, the person had a higher level of situation awareness of what the system was doing, whether it was heading into trouble, and what the alternatives might be if a problem arose. The conclusion was that it was cheaper to pay to keep a worker engaged full time to handle rare, but challenging problems, rather than eliminate a full-time person and call them in only when a problem occurred. A third alternative to add artificial intelligence capabilities for problem solving and reasoning was not considered.

Returning to the substitution myth, the point is that while both automation or autonomy provides capabilities that may appear to replace human capabilities, machine intelligence is different from human intelligence. Designers must expend extra effort to make sure the machine has the implied intelligence that a human would exhibit in a particular situation, so that the robot really does substitute for the key attributes of a human or that the boundaries of where the robot will not behave as a human are understood. Designers must also recognize that, as relatively little is known about the interaction between human intelligence and machine intelligence, that failures will occur, and thus mechanisms to allow the human to assume control are essential in practice.

3.8 Trade-Spaces in Adding Autonomous Capabilities

The issue of how to design autonomous capabilities has been examined by the US Department of Defense. A 2012 study by the US Defense Science Board (DSB) found that unmanned systems, regardless of whether they are being used on the ground, air, or sea, have problems stemming from initial design decisions.¹⁴⁴ The study found that systems have often been programmed with capabilities that are too narrow to be really useful, have cost overruns of software and hardware for re-engineering, and have increased, rather than reduced, the number of people needed to accomplish a mission. While the unmanned systems reduced the number of people in harm's way, large numbers of people are needed to maintain the physical platforms and to supervise the operations and take over when the software fails. In addition, the study found that a major problem was that autonomous capabilities were not being used for the right applications or were not being implemented with best practices from artificial intelligence and cognitive science. The mismatch of applications and clumsy implementations created distrust among the users.

TRADE SPACES IN HUMAN-MACHINE SYSTEMS

The study made numerous recommendations; one was that designers and acquisition officers consider the design of the unmanned system in terms of *five trade spaces in human-machine systems fitness, plans, impact, perspective, and responsibility*. The five trade spaces are described below:

- *Fitness*, which represents the tradeoff between optimality and resilience. Industrial manipulators are designed to use internal position sensors to execute optimal paths over a set of expected actions. This design enabled the robot to be optimized for rapid execution and be less expensive because it did not require any external sensors. To achieve this optimality most designers assumed the object being manipulated was in the correct location and nothing was in the robot's workspace. The result was a robot that was engineered for a particular process line but was hard to adapt to new set of

objects to be manipulated or acted upon and could kill a worker who entered the workspace.

- *Plans*, which represents the tradeoff between efficiency and thoroughness. An example of this tradeoff was the algorithm discussed earlier that was used to count dinosaurs in *Jurassic Park*. It was very efficient but at the cost of thoroughness. More intelligent systems take advantage of advances in AI methods for reducing the dependence on assumptions and for monitoring plan execution. In navigation, one approach is to plan the complete path for the entire mission at a high degree of spatial resolution, which is thorough, while another approach is to plan at a low degree of spatial resolution and react to obstacles as they are encountered, which is efficient.
- *Impact*, which represents the tradeoff between centralized and distributed perspectives. As will be described in chapter 17, teams of robots can be controlled from either a centralized computer or with control distributed among the team. Centralized control introduces time lags in communication and vulnerabilities if the centralized computer fails. AI researchers have shown the efficacy of distributed control when the robots coordinate among themselves, either using simple behaviors to act as a swarm or to bid on more complex tasks.
- *Perspectives*, which represent the tradeoff between local and global views. A common dilemma in ground vehicle navigation is the box canyon problem, where the robot turns into a dead end. If the robot is not simultaneously mapping the world and keeping up with its position in the world, that is, creating a global view, the robot may not notice that it is in a dead end. A robot acting with only a local view may be vulnerable to a “fly-at-the-window effect.” Flies are attracted to light, and, once in a room or a car, move to the window. But they are unable to see the glass and continue to try to fly through it, often missing an open window elsewhere. However, creating and maintaining an accurate global world map is computationally expensive, requires range sensing for spatial modeling, and scene understanding.
- *Responsibility*, which represents the tradeoff between short-term and long-term goals and the decision about which agent is ultimately responsible for reaching those goals. Autonomous capabilities may apply to a particular phase of a mission. For instance, some aerial vehicles have autonomous take-off capabilities but not autonomous landing capabilities. In almost every case, a human operator is expected to take over if the autonomous capability cannot be used or fails, so the human has responsibility even for autonomous take-off or landing. In contrast, NASA’s 1998 Deep Space 1 probe was designed to be fully autonomous for its entire mission with onboard problem solving and reasoning capabilities to work around onboard failures (Engineers did have reconfigure its navigation system during the initial trial run but after that the robot probe was on its own).

Returning to the discussion in the previous section of the differences between automation and autonomy, it is easy to see how design problems might emerge and how deterministic testing might miss problems. The fitness and plan trade spaces exemplify the tensions between the open-world and closed-world assumptions, where algorithms and actions can be very precise for well-understood situations but inappropriate for anything outside of expectations. The claims of optimality and thoroughness are based on some bounded representation of the world, which means the system could be vulnerable to an overly restrictive closed-world assumption and poor handling of the frame problem. Without the right balance of fitness, planning, perspective, and responsibilities, a system can be stuck

executing the wrong plan or have difficulty revising the plan. Yet a human might not notice or have access to the detailed workings needed to modify the plan.

As will be discussed in chapter 13 on Human-Robot Interaction, these trade spaces also capture the potential causes of problems between human operators and robots. For example, the impact and perspectives spaces include the design of interfaces, where the interface needed for one person to direct and see all the information gathered by the robot might be very different from an interface that allows multiple people in different locations to use the information relevant to them. The fitness, impact, perspectives, and responsibility trade spaces are particularly vulnerable to the substitution myth and introduce human out-of-the-loop control conundrums. The tradeoffs between centralization and distributed systems also involve coordination between humans using the robot, where one team of operators may be responsible for interpreting imagery and another for acting on the imagery. If the human is working in a local view but is sent all the data for the global view, the human can be overloaded by the data and take longer to make decisions. In terms of the responsibility trade space, success with short-term goals builds up trust in the system which can overcome the general reluctance to delegate long-term goals to the robot.

3.9 Summary

This chapter answers four common questions about automation and autonomy. The basic question is *What is the difference between automation and autonomy?* One way to describe the difference is to say that automation is about robots as tools and autonomy is about robots as agents, following the definitions in chapter 2. Another way is to discuss the differences in terms of four main characteristics. One characteristic is *plans*: Does the system focus on the execution of plans versus generation of plans? Another characteristic is actions: Does the system use deterministic or non-deterministic algorithms? The type of *world model* also helps distinguish between tools and agents. Tools typically operate in a closed world with large amounts of a priori information while agents work under an open-world assumption to accomplish variable tasks in dynamically changing environments. The fourth difference is knowledge representation, whether the system manipulates signals or symbols. In practice, intelligent robots have a mixture of characteristics, for example, one robot might have more characteristics from the automation end of the spectrum, while another robot would have more characteristics from the autonomy end of the spectrum.

If intelligent systems are a mixture of attributes of automation and autonomy, *Why does it matter that there is a difference between autonomy and automation?* Automation and autonomy are often indistinguishable for relatively unintelligent tasks or they may be used together, such as navigating to a waypoint where an AI algorithm may be used to select the waypoints for an unmanned aerial vehicle and control theory to keep the UAV flying on course. However, these approaches reflect different design paradigms that require different amounts of effort and will introduce different tradeoffs for more intelligent capabilities. The choice of paradigm is similar to choosing coordinate systems in calculus, where a problem can be solved by posing the solution in either a Euclidean or spherical coordinate system. If done correctly, both will give the same answer but one will require significantly different effort and increase the probability of a math error. In robotics, the choice of approach impacts the programming style, the hardware design, the kinds of functional failures and the types of human error

to prepare for. Thus it is important for a designer to consciously choose the correct end of the automation/autonomy spectrum in providing a specific intelligent capability and to defend against the different types of vulnerabilities associated with the chosen approach.

What are the advantages of autonomy over automation? Can you tell me when to use one over the other? How much autonomy do I need? The advantages, and suitability, of autonomous capabilities over automated capabilities depend on the application. The lack of design rules for when to use autonomy are exacerbated by the lack of non-deterministic testing methods; good judgment is needed to decide when and how to add autonomy and traditional means of testing are not guaranteed to uncover any mistakes in judgment. While there is no hard and fast rule for what capabilities work best for which robotic system, there are at least five trade spaces for a designer to consider: fitness, plans, impact, perspective, and responsibility. Classic automation in manufacturing robots optimizes the fitness of the physical robot for precision, repetition, and expense but at the cost of requiring parts to be perfectly placed for the robots and risking the robot's adaptability to new tasks. Adding an autonomous capability, such as path planning, independent of other related autonomous capabilities, such as mission execution monitoring, to make sure that the planned path is still valid, risks system failures or rejection by users who deem the system too difficult or demanding to use.

Designers can underestimate the amount of autonomy needed for an intelligent robot to meet implicit expectations. This stems, in part, from the substitution myth, that a machine or software system can directly replace a capability provided by a person. Usually a person provides both an actual capability, such as planning where to go, and *additional implied capabilities*, such as exception handling or problem-solving. Consider an example from the plans trade space. Smartphones and mobile devices can currently generate optimal routes based on maps and, in some cases, traffic information. However, they rely on the human driver to notice that the map or route has an error or, in the case of a military operation, that the blockage in a road is odd and could signal an ambush. Adding a path-planning capability to an unmanned ground vehicle without adding a plan-execution monitoring capability may create a robot that is too narrowly “autonomous” to be used without constant human supervision or without risk of failure. While artificial intelligence has algorithms for most intelligent capabilities, it is still up to the designer to determine which set of capabilities to include in a system.

3.10 Exercises

Exercise 3.1

Define automation.

Exercise 3.2

Define autonomy.

Exercise 3.3

What is the closed-world assumption? How is it different from closed-loop control?

Exercise 3.4

What is the open-world assumption? How is it different from open-loop control?

Exercise 3.5

True or false. The open-world assumption states that the list of possible states, objects, or conditions cannot be

completely specified and thus designers must impose a closed-world model on the problem space.

Exercise 3.6

True or false. The frame problem refers to the problem of correctly identifying what is unchanging in the world and thus does not require constant updating, thereby reducing computation. The frame problem is related to a closed-world assumption.

Exercise 3.7

Fill in the question marks with either “deterministic” or “non-deterministic.” Automation typically relies on ? algorithms, while autonomy relies on ?

Exercise 3.8

Consider an autonomous system with four slider bars. The position of each slider indicates the contribution to autonomy or automation. For example, one robot can be mostly on the generating (autonomy) side of the plans spectrum but more on the closed-world (automation) side of the models spectrum and so on. Look up each of these robots and sketch the slider positions for each:

- Roomba
- Shakey
- Unimate
- Genghis
- DaVinci surgical robot
- Deep Space One
- Predator
- Remus

Exercise 3.9

Recall the four different categories of applications of robots from chapter 1. Is one category more “open world” than the others? One more “closed world” than the others? Give examples of how a closed-world assumption might impact a robot in each category. Give examples of how an open-world assumption might impact a robot in each category.

Exercise 3.10

What are three ways in which a world model can be created?

Exercise 3.11

What are the three items that lead to *bounded rationality*?

Exercise 3.12

Which is NOT one of the three items that lead to bounded rationality?

- a. the amount of time the robot has to make the decision
- b. how much information the robot has
- c. the number of problems or goals the robot is trying to solve or satisfy
- d. the robot’s computational abilities

Exercise 3.13

Choose the right answer. The substitution myth states:

- a. that a robot will eventually replace people
- b. that a robot can replace a human with no impact on the larger socio-technical organization or resilience of the system

- c. that a person can substitute for a robot's superior endurance and attention to detail
- d. that artificial intelligence cannot be substituted for biological intelligence.

Exercise 3.14

Consider factory automation: Does the substitution myth emerge? If so, under what circumstances?

Exercise 3.15

Search the Web for reports about unmanned aerial systems crashing, flying away, or doing other surprising things. Speculate on the balance between automation and autonomy and why the system failed.

Exercise 3.16

Search the Web for a description of the Baxter robot which advertises itself as a new approach to factory automation. List the autonomous capabilities. How does it appear to balance the trade spaces and avoid the substitution myth?

Exercise 3.17

Consider the 2011 Fukushima Daiichi nuclear accident and clean up efforts. What autonomous capabilities would be useful for a ground robot? List any tasks that could be automated and why.

3.11 End Notes

For the roboticist's bookshelf

"The Role of Autonomy in DoD Systems," the July 2012 report from the Defense Science Board (available online), provides an overview of US military unmanned systems summarizing what autonomous capabilities have been used, the missed opportunities to use autonomy, and recommendations for increasing the effective use of autonomy in the future.¹⁴⁴

Sci-fi Movies

An example of non-determinism occurs in the 1972 science fiction movie, *Silent Running*. Three robots are outside a space station conducting maintenance when they are ordered to come inside due to an impeding collision with debris from Saturn's rings. All three robots were working near each other when the order was given, so in a deterministic system, all three should have chosen the same route to the same airlock and re-entry. Robots Huey and Dewey choose one route, but Louie, presumably based on the variation in its local perception, chose another route.

4

Software Organization of Autonomy

Chapter Objectives:

- Apply Levis' definition of architectures to conceptualizing software in an intelligent robot and express the different levels of abstraction in *operational*, *systems*, and *technical* architectures.
- Name the layers (*reactive*, *deliberative*, *interactive*) within a canonical operational architecture of an intelligent robot and describe them in terms of the value of the five common attributes (*primitives*, *perceptual ability*, *planning horizon*, *time scale*, *use of models*).
- Describe the four categories of deliberative functionality (*generating*, *monitoring*, *selecting*, *implementing*) and explain how these functions differ from reactive functionality in terms of *world model*, *time scales*, and *planning horizon*.
- Identify a systems architecture as implementing either a hierarchical, reactive, or hybrid deliberative/reactive paradigm if given the two distinguishing traits of robotics systems architectures: i) the pattern of the relationship of the three AI robot primitives and ii) the route of sensing.
- Label and define the five subsystems that appear in most systems architectures: *Planning*, *Cartographer*, *Navigation*, *Motor Schema*, and *Perception*.
- Associate reactive and deliberative operations with each of the five subsystems and explain how the operations would use those subsystems.
- Describe the similarities and differences of *execution approval* and *task execution* and give two reasons why they do not ensure safe or correct actions of a robot.

4.1 Overview

The previous chapter established that autonomy is conceptually different from control theoretic approaches—that AI researchers view robotics as an issue of increasing adaptability (doing the “right thing” in an open world), whereas many engineering researchers view robotics as extending control (creating a larger world model and guaranteeing actions). The discussion of these differences was general and did not provide any real guidance as to how to design an intelligent robot. This chapter

attempts to broadly answer the questions: *Given that autonomy has a different programming style, what is it?* and *Can intelligence be added in layers, like upgrading to a “pro version” or downloading “apps” as needed?*

This chapter begins by discussing programming style in terms of three types of software engineering architectures and then focusing on the operational and systems architectures that dominate AI robotics programming. In order to understand software organization, the chapter begins by defining “architecture.” Levis’ definitions of *operational*, *systems*, and *technical* architectures¹¹⁴ are used to help frame the discussion of software design. Operational architectures are analogous to describing a house in terms of important features. For example, a real estate listing may describe a house in terms of style, for example, ranch, cottage, mansion, mobile home, and so forth, or function, for example, duplex, single family, and so forth, which gives a prospective buyer an indication of whether it would fit the buyer’s family and lifestyle. Systems architectures capture what the house consists of, such as having three bedrooms, two baths, a large kitchen, a covered garage, and a swimming pool. Technical architectures are the actual instances of the design, where the floor plan for a house is the declaration of the function. Consider that a three-bedroom, two-bath house can have different configurations of the rooms and the ways in which they are connected, such as whether the house has a second story, the number and extent of hallways, any shared bathrooms, and so on. Technical architectures also consist of the algorithms to implement the functions, which are similar to the actual framing, plumbing, painting, landscaping, and decorating details.

Next, the chapter presents the canonical operational architecture used in AI, which groups intelligence into three broad layers: Reactive, Deliberative, and Interactive. This canonical operational architecture will be the framework for organizing the rest of the book, with part II on the systems and technical architectural details of the Reactive Layer, part III on the Deliberative Layer, and part IV on the Interactive Layer. The canonical operational architecture is not well known outside the AI robotics community and thus the chapter describes three other operational architectures that are used by the industrial automation, controls, and general AI community.

The canonical operational architecture is, by definition, high-level and abstract, and thus the chapter goes on to describe the next more tangible architecture, the systems architecture. There are five common subsystems used in AI robotics implementations: *Planning*, *Navigation*, *Cartographer*, *Motor Schema*, *Perception*. These correspond to the types of rooms in a house. There are three paradigms, or floor plans, with the hybrid deliberative/reactive systems architecture, often shortened to *hybrid architecture*, being dominant. The hybrid architecture will be the backbone of the remainder of the book as it delves into the subsystems and algorithms that comprise a technical architecture, or specific house.

The architectural perspective highlights that programming an intelligent robot requires many hardware and software components to work perfectly together. If not, the robot is at risk of failing. One approach to reducing risk is to insert the human into the execution cycle to review and approve what the robot intends to do before it does it. As will be discussed in this chapter, human factors research suggests that human approval does not actually yield better robot performance.

Looking ahead, the next chapter will cover telesystems, which are often thought of as substitutes for designing and building a “complete” AI robotics system or “fully autonomous” robot. That will conclude the Big Picture overview of AI robotics. The remainder of the book will survey the different

algorithms and programming methods for various software modules that comprise the technical architecture view, similar to capturing the plumbing, framing, and landscaping details for a specific house. With this chapter, a person could begin a top-down design of an AI robot; however, design methodologies and philosophies are deferred to chapter 19.

4.2 The Three Types of Software Architectures

The term “architecture” is frequently used in robotics to refer to the overall style of design or organization. Arkin offers several definitions in his book, *Behavior-Based Robots*.¹¹ Two of the definitions he cites from other researchers capture how the term will be used in this book. Following Mataric,¹²⁴ an architecture provides a principled way of organizing a control system. However, in addition to providing structure, an architecture imposes constraints on the way the control problem can be solved. Following Dean and Wellman,⁵⁷ an architecture describes a set of architectural components and how they interact. This book is interested in the common core of components in robot architectures, as these are the basic building blocks for programming a robot, and in the principles and rules of thumb for connecting these components together. To see the importance of an architecture, consider building a house or a car. There is no “right” design for a house, although most houses share the same components (kitchens, bathrooms, walls, floors, doors, etc.) and infrastructure (electricity, plumbing, heating and air conditioning, etc.). However, the size of the rooms, sophistication of the infrastructure, along with the floor plan, influence whether it is “right” for the buyer.

4.2.1 Types of Architectures

Architectures distill prior experience into templates for creating new intelligent robots. The term “architecture” was used so frequently with different meanings in the US Air Force that in 2001 the Chief Scientist Alex Levis formalized a taxonomy¹¹⁴ of architectures: operational, systems, and technical. These are described below.

OPERATIONAL ARCHITECTURE

- *operational architecture*: describes *what the systems does*, or its functionality, at a high level, but not how it does it. This description is similar to the real estate listing for a new house describing the house in commonly used terms such as the style of house (e.g., bungalow, Colonial, ranch, etc.) and the number of bedrooms and bathrooms. An operational architecture specifies how the seven distinct areas of artificial intelligence, each with their own algorithms and data structures, are knitted together and can be used to determine if a design provides the intended functionality.

SYSTEMS ARCHITECTURE

- *systems architecture*: describes *how a system is decomposed into major subsystems*. This description is similar to the floor plan for the house. The systems architecture states the specific software systems (rooms) that supply the desired intelligent functionality and describes how they are connected (hallways? open floor plan? a second story or basement?). The systems architecture can be used to determine if a design meets good software engineering principles, especially modularity and extensibility.

TECHNICAL ARCHITECTURE

- *technical architecture*: describes *how the implementation details of the a system*. This is similar to describing how the house is actually built, where the builder places the pipes, what building codes apply to the roof or foundation, what materials will be used, how expensive the light fixtures and cabinets will be, and so forth. The technical architecture specifies the algorithms and even the languages that the modules are programmed in and it can be used to determine if a design is using the most appropriate algorithms. Technical architectures are in constant evolution, reflecting advances in artificial intelligence.

4.2.2 Architectures Reinforce Good Software Engineering Principles

Artificial intelligence is a software enterprise, and good software engineering is necessary for a successful software enterprise. Thinking of the design of an autonomous robot in terms of the three architectures encourages the creation of designs that meet the four general principles of software engineering.⁸¹ One is *abstraction*, where the operational and systems architecture portals allow designers to ignore details in order to focus on thinking about general organization of intelligence. Another principle is *modularity*, where the systems and technical architectures encourage the designer to think in terms of object-oriented programming. Modules should have high cohesion, where each module or object does one thing well and unrelated functions are put elsewhere, and low coupling, where modules or objects are independent. High cohesion and low coupling support unit testing and debugging. The third principle is *anticipation of change with incrementality*, as the systems and technical architectures provide the designer with insights into whether the code is engineered to support upgrading an algorithm with a newer one and adding new capabilities without requiring the designer to rewrite the code for existing modules and possibly to introduce bugs. This principle is related to modularity. The fourth principle is that of *generality*. The operational, systems, and technical architectures provide the designer with frameworks to determine if the basic organization, subsystems, and implementation will allow the code to be used for other applications. In robotics, this principle is often called *portability*.¹¹ The intent of this chapter is to give templates for the operational and systems architectures so that they do not have to be re-invented each time.

NICHE TARGETABILITY

ROBUSTNESS

In addition to the four general principles of software engineering, Arkin¹¹ adds two more software engineering principles for AI robotics: *niche targetability* and *robustness*. Niche targetability captures how well the robot works for the intended application. Note that niche targetability and ease of portability are often at odds with each other. This is where a technical architecture may commit to algorithms that are highly specific to a particular application. For example, the robot may be designed to work only during the day. But if the technical architecture was well organized into subsystems within a systems architecture, the daylight-specific algorithms could easily be replaced with algorithms that worked at nighttime. Robustness identifies where the system is vulnerable, and how the system design intrinsically reduces that vulnerability. This vulnerability may emerge at the operational or systems architecture level of abstraction, where the designer can see if there is a subsystem that will monitor execution. Or it may emerge through the technical architecture serving as a portal through

which the designer can examine how the code handles exceptions and failure conditions.

4.3 Canonical AI Robotics Operational Architecture

THREE LAYERS

After a series of “paradigm wars” in the years between 1967 (Shakey) and 2000, AI robotics has converged on the hybrid deliberative/reactive operational architecture. Organizationally, the operational architecture consists of three layers, *reactive*, *deliberative*, and *interactive*, following the biologically-inspired metaphor in figure 4.1. Biological intelligence can be thought of as consisting of four major functions, *reaction*, *deliberation*, *conversion of signals into symbols*, which bridges reaction and deliberation, and *interaction* of the robot with other external agents. This abstraction of biological intelligence leads to three distinct purposes and styles of computing. The three layers not only encapsulate philosophically different categories of functionality, they have five attributes which influence computing: *primitives*, *perceptual ability*, *planning horizon*, *time scale*, and *use of models*. As will be discussed in more detail in chapter 19, the layers may use significantly different programming structures and languages. This section first introduces five attributes for describing the layers, discusses each layer individually building to a summary diagram.

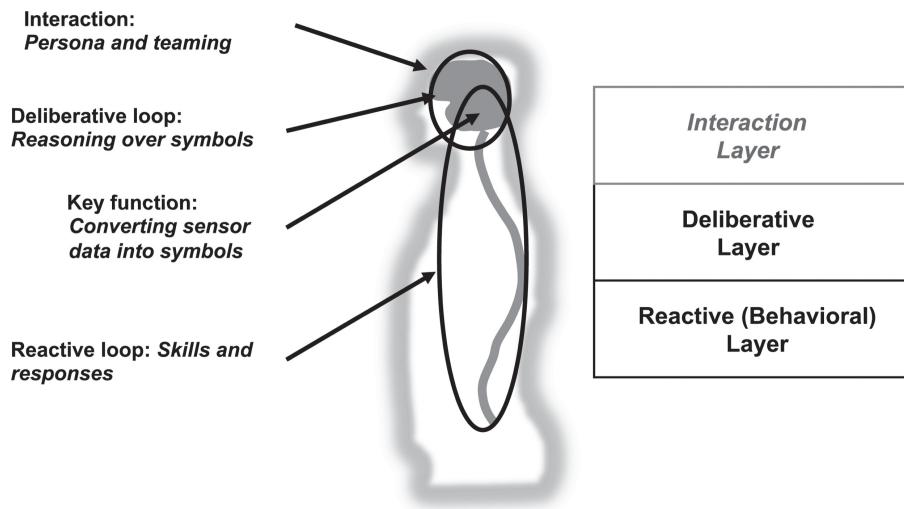


Figure 4.1 Central nervous system represented as an operational architecture.

4.3.1 Attributes for Describing Layers

The description and justification for what software functions go into each layer are based on five attributes: *primitives*, *perceptual ability*, *planning horizon*, *time scale*, and *use of models*.

FOUR PRIMITIVES

Primitives. Similar to functions in the LOA and ACL architectures, AI robotics views autonomous capabilities as consisting of four primitives: SENSE, PLAN, ACT, and LEARN. If a function is taking

in information from the robot's sensors and producing an output useful by other functions, then that function into the **SENSE** category. If the function is taking in information (either from sensors or extracted from its own knowledge about how the world works) and producing one or more tasks for the robot to perform (go down the hall, turn left, proceed three meters and stop), that function is in the **PLAN** category. Functions which produce output commands to motor actuators fall into **ACT** (turn 98°, clockwise, with a turning velocity of 0.2 mps). Intelligence is generally viewed as a process of **SENSE**, **PLAN**, and **ACT**, similar to the OODA loop; see [table 4.1](#). **LEARN** is an important mechanism by which an agent maximizes its chances for success, but it transcends the other three primitives; an agent can learn to sense or plan better, acquire more acts or skills, or even learn for a specific mission what to sense, what to plan for and how to act. As with ACL, more intelligent systems have increasingly sophisticated instantiations of one or more of these four primitives.

Table 4.1 Three robot primitives defined in terms of inputs and outputs.

ROBOT PRIMITIVES	INPUT	OUTPUT
SENSE	Sensor data	Sensed information
PLAN	Information (sensed and/or cognitive)	Directives
ACT	Sensed information or directives	Actuator commands

PERCEPTUAL ABILITY

Perceptual ability. AI robotics views the perception needed for a particular capability as being either *direct* or requiring *recognition*. This divides capabilities into those that can work directly with incoming stimuli and those requiring a conversion of the stimulus into a symbol.

PLANNING HORIZON

Planning horizon. AI robotics views a capability as also having a planning horizon of either *present; past, present; or future, past, present*. For example, the reactive layer was concerned only with stimulus from the present, while deliberative capabilities require reasoning about the past or projecting how the world will be in the future as a result of an action. The planning horizon constrains the choice of data structures (e.g., if the robot is reasoning about the past or future, then it has to store previous and current states), as well as algorithms (e.g., if the algorithms are working only in the present, the robot cannot use reasoning methods that project future consequences).

TIME-SCALE

Time-scale. AI robotics also considers the time-scale of a function—whether it needs to be very fast (a reflex), fast (selection of actions), or can be relatively slow (reasoning about a problem). Time-scale

helps determine the asynchronous operation of the software components. One technical architecture, RCS,⁵ specifies the time-scales for different functions creating layers of functions executing over similar time-scales.

World Model. AI robotics divides capabilities into those that use information collected only for that function, called a *local world model*, and those that rely on all sensor information to first be processed into a *global world model*. This division helps determine the degree of sophistication of a capability.

4.3.2 The Reactive Layer

Reactive, also called *behavioral*, functionality corresponds to the reactive loop in the central nervous system. This loop is associated with functionality that occurs in the spinal cord and “lower brain,” especially responses and skills based on motor memory. Involuntary reflexes, such as the knee-jerk response to being tapped on the patella, and learned skills, such as picking up a coffee cup or riding a bicycle, allow the agent to execute actions quickly. The responses and skills are patterns of action generally referred to as *behaviors*. Behavioral functionality is sufficient for animal intelligence, such as exhibited by fish and insects.

In robotics, the reactive layer, as shown in figure 4.2, consists of functionality constructed from the SENSE and ACT primitives with no PLAN. This layer corresponds to animal intelligence where action is generated directly by sensed or internal stimuli, and there is no awareness of the larger world and no mission monitoring. SENSE and ACT are tightly coupled into constructs called *behaviors* which produce capabilities. One or more behaviors are triggered either by external or internal stimuli and the overall behavior emerges based on the mechanism for combining the outputs from each behavior. The perceptual ability of the behaviors is called *direct perception*, which will be described in later chapters. While a behavior does not create and maintain a global world model, a behavior may have a local world model, which serves as short term memory. There is no planning, and thus the planning horizon is the present. The time-scale for the functions is very fast; stimulus-response behaviors typically have a fast update cycle on the order of 15–30 cycles per second, matching control needs and sensor update rates.

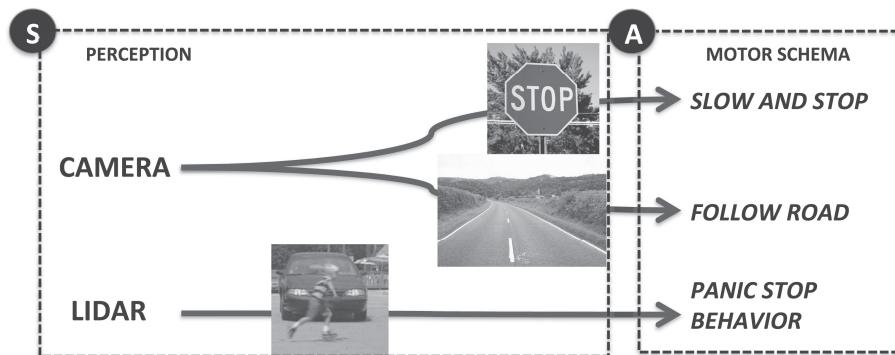


Figure 4.2 Organization of the primitives in the Reactive Layer.

4.3.3 The Deliberative Layer

Deliberative functionality corresponds to the cognitive loop associated with the cortex in the brain. The brain independently takes as inputs the same signals used by the lower central nervous system and adds additional sensor processing to make conscious decisions. The cognitive loop can modify the reactive loop, either to instantiate new actions that would be carried out by reactive functionality (e.g., walk up the stairs to the building) or to modify them (walk slower than normal because there may be unseen ice on the stairs). Deliberative functionality is sufficient for the more sophisticated problem solving and reasoning aspects of intelligence as well as planning. For example, the deliberative loop is where reasoning about where to find a coffee cup occurs and it activates the reactive loop in order to move the robot to the kitchen and then to pick up the coffee cup.

The two loops differ in processing speed and the content of what is being processed. Reflexes and motor skills are very fast, while deliberating about a problem may be slow; this might explain why the loops are asynchronous. A office worker can catch a falling coffee cup, yet, at the same time, mull over how to balance a financial account. Reactive functionality appears to rely on signals, that is, direct sensor stimuli, while deliberative functionality appears to rely on symbols, that is, stimuli that have been synthesized, labeled, and combined with memory and knowledge. How the brain converts signals into symbols, either how it recognizes an object that should be assigned a symbol (e.g., a coffee cup) or how it assigns a semantic label to the object (e.g., my coffee cup), is not well understood. In AI, this is referred to as *object recognition* and constitutes the *symbol grounding problem*.

As shown in figure 4.3, the Deliberative Layer hosts the PLAN activities of the robot, has a Planner that generates a plan using a World Model and then instantiates appropriate SENSE-ACT behaviors in the reactive layer to execute the plan. These behaviors run until the next step in the plan is reached and a new set of behaviors is instantiated or the deliberative monitoring detects a problem with the plan. The perceptual ability of the behaviors is recognition, where the robot builds up global or persistent world models. The planning horizon uses information from the past and the present to project consequences of a plan into the future. The time it takes to plan is slower than to react, and functions in the Deliberative Layer may update with a frequency ranging from on the order of 15 cycles per second to update the World Model to several minutes to construct complex plans. Even if a planning algorithm runs nearly as fast as the behaviors in the Reactive Layer, there may be no advantage in continuously replanning and reinstating behaviors.

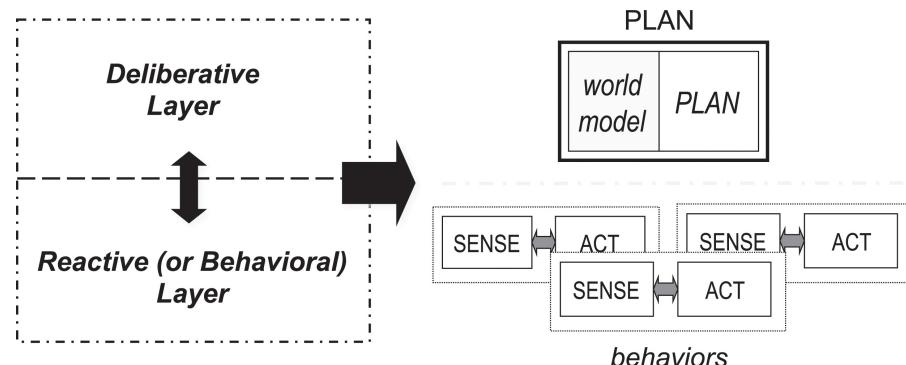


Figure 4.3 Primitives in the Deliberative Layer.

FOUR DELIBERATIVE FUNCTIONS

The Deliberative Layer is divided into two sublayers connected by the World Model that contain the *four deliberative functions*:

GENERATING

- *generating* plans which corresponds to planning, reasoning, and problem solving in AI

SELECTING

- *selecting* specific resources to accomplish the plan, which corresponds to planning, resource allocation, and knowledge representation of capabilities in AI

IMPLEMENTING

- *implementing* the plan, which corresponds to execution

MONITORING

- *monitoring* the execution of the plan to determine if it is meeting the goal, learning what is normal, and anticipating potential failures, which corresponds to planning and reasoning in AI.

As shown in [figure 4.4](#), the upper deliberative layer contains mission generation and monitoring functionality which work over past, present, and future time horizons. The lower deliberative layer contains the selection and implementation functionality that converts the plan into behaviors. The lower sublayer works over past and present time horizons. The sublayers operate asynchronously. The World Model is bounded, representing what is important in the world for the overall intended capability of the robot. For example, many robots create and maintain a 2D or 3D map of the physical layout of the world for autonomous navigation but do not incorporate changes over time to support the semantic understanding needed for plan monitoring (e.g., “Why is the road suddenly blocked? Am I about to be ambushed?”). Functions within the deliberative layer are often programmed in a functional language, such as Lisp, which simplifies planning and reasoning.

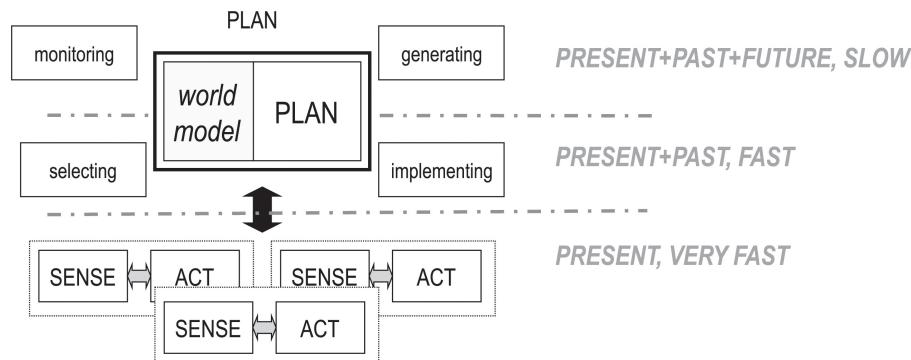


Figure 4.4 Sublayers in the Deliberative Layer with the four key deliberative functions.

4.3.4 The Interactive Layer

PERSONA

SWARM BEHAVIORS

Interactive functionality is needed for interaction with other agents, either people, other robots, or software agents. Interaction may be regulated by reactive or cognitive loops. Internal reactions to situations or to others may generate negative emotions such as fear or anger, which may manifest themselves as changes in posture and gestures (faster and more extreme movements), proximity to other agents (step closer to be threatening), and verbal communication (word choice and speaking louder); this is sometimes referred to as *social intelligence* and will be described in chapter 18. Interaction can also be regulated deliberatively, such as with *security and privacy* decisions. Displays of interactive functionality essentially surround the person and produce their *persona*. However, interactions may arise when working in teams as described in chapter 17. These interactions include *swarm behaviors* where colonies of ants find food, birds flock, and fish school, or, for more sophisticated teams, where members explicitly negotiate what portions of a task they will take responsibility for.

The Interactive Layer is essentially a wrapper on individual robot programming that enables the robot to work with other agents—either humans, robots, or software agents. The functionality within the interactive layer is a topic of active research into how robots communicate to, and work with, each other and with humans, how they maintain security and privacy, and what are good user interfaces. There is no figure for the internal organization of the Interactive Layer because there is currently no clear pattern of the attributes of the layer. For example, natural language and other forms of communication can be produced reactively¹⁵⁶ or deliberatively.⁹⁴ Likewise, emotions that provide feedback about the robot’s state can be generated reactively or deliberatively, depending on the circumstances.¹⁴⁰

The Interactive Layer began to emerge as a topic of interest around 2006, driven by research into how teams of robots could cooperate explicitly with each other and by research into human-robot interaction for the specific applications of social robots for entertainment, assistance, and health care. The Interactive Layer may be programmed in procedural, functional, or ontological languages, such as OWL.

4.3.5 Canonical Operational Architecture Diagram

The canonical operational architecture can be represented as seen in [figure 4.5](#), which shows the organization of the primitives, the perceptual ability, the planning horizon, the time scale, and the use of models in the Reactive and Deliberative layers. Returning to [figure 4.1](#), the layers correspond to the processing loops, and the World Model handles the conversion of signals into symbols.

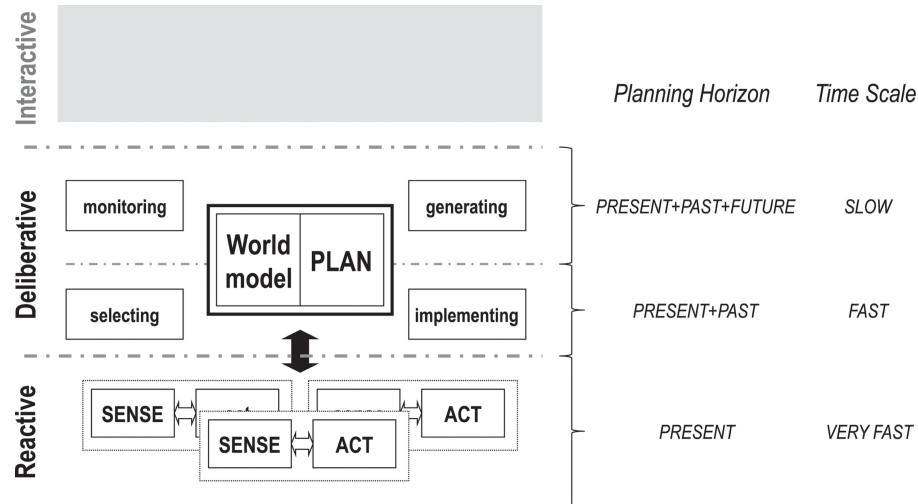


Figure 4.5 Canonical AI robotics operational architecture.

4.4 Other Operational Architectures

The canonical organizational architecture favored by the AI robotics community is based on a biological metaphor, but there are at least three other distinct styles of operational architectures providing different, thought-provoking ways to think about organizing intelligence. The automation community has created taxonomies defining how functions can be assigned either to a machine or a human, leading to *levels of automation*. However, these levels have been treated by some developers as an operational architecture. The controls community, especially in aerospace, has attempted to categorize increasing levels of understanding and control over a situation, creating a variant of levels of automation called *autonomous control levels*. The multi-agent community within AI has begun to think of autonomy in terms of increasing levels of understanding and control over a situation, more specifically as the amount of initiative that a robot is allowed to exert, captured as *levels of initiative*.

This section reviews these other architectures and discusses how they relate to the canonical AI robotics architecture. The levels of automation and the autonomous control levels framework appear throughout the robotics literature, so it is especially helpful to understand them and their limitations for AI software organization.

4.4.1 Levels of Automation

LEVELS OF AUTOMATION (LOA)

LEVELS OF AUTONOMY

While biological intelligence provides insight into the general organization of intelligence, the human-machine systems community offers a different viewpoint. The human-machine systems community has looked at automation for space exploration, control of processing plants, and autopilots in aircraft. Starting with the seminal work of Thomas Sheridan,¹⁹¹ the community has traditionally labeled the state of automation at any given time in terms of what functions a human has currently delegated to the

computer. This classification of the state in a system naturally leads to hierarchies of where the machine has responsibility for more of the functions, and thus would be considered more autonomous. These taxonomic hierarchies are often used as operational architectures or as *de facto* measures of how automated a system is, where Level B is more automated than Level A. The hierarchies are generally referred to as levels of automation or *levels of autonomy* (LOA), as automation and autonomy are often used synonymously.

The book will focus on the general themes common to the variants of the levels of automation. A common way to express the state of automation in a system is to divide functional capabilities into four broad categories.

SHARED CONTROL

TRADED CONTROL

A function can be performed by a human, a computer, both working together (referred to as *shared control*), or portions can be performed by either a human or computer and then the remaining portions by the other agent (referred to as *traded control*).

The state of automation for a process is defined by whether the human or the machine is performing the function. **Table 4.2**, derived from,⁷¹ provides an example of a taxonomy of the human-machine relationships arranged in a hierarchy of increasing levels of responsibility by the machine. No particular levels of automation is accepted by all researchers, with different experts arguing over the functions and the ordering of the hierarchy. However, the different variants of levels of automation all share the same tenet that there are a set of functions and the more functions delegated to a computer means higher automation. The resulting *levels of automation* approach is often used as an operational architecture, with the expectation that an intelligent robot would be built incrementally in the layers corresponding to the rows in the table. An unintended consequence of using levels of automation as an operational architecture is that it implies full automation is the design goal, rather than matching the appropriate level of automation for the mission and the robot's capabilities.

Table 4.2 Representative levels of automation hierarchy, adapted from Endsley and Kaber.⁷¹

Level of Automation	Generating		Selecting		Implementing		Monitoring	
	Human	Computer	Human	Computer	Human	Computer	Human	Computer
1 Manual Control	✓		✓		✓		✓	
2 Active Support	✓		✓		✓	✓	✓	✓
3 Batch Processing	✓		✓			✓	✓	✓
4 Shared Control	✓	✓	✓		✓	✓	✓	✓
5 Decision Support	✓	✓	✓			✓	✓	✓
6 Blended Decision Making	✓	✓	✓	✓		✓	✓	✓
7 Rigid System	✓			✓		✓	✓	✓
8 Automated Decision Making	✓	✓		✓		✓	✓	✓
9 Supervisory Control	✓			✓		✓	✓	✓
10 Full Automation	✓			✓		✓	✓	✓

The major advantage of using the levels of automation organization is that the four functions and levels offer more specific modularity than the three layers in biological intelligence. The taxonomy acknowledges that a mission may require both robots and humans working together. It suggests that functionality can be added to a robot to reduce its dependence on humans, essentially that an individual robot can be upgraded over time from an unintelligent, totally dependent on a human operator form of automation to an intelligent, totally independent of a human operator form.

The major disadvantage of this organization is the levels of autonomy were initially intended to be a set of definitions for labeling the current state of an active capability, not an operational architecture for coordinating all capabilities. If the taxonomy is applied as a designation of the intelligence for the overall system, it may interfere with designing capabilities of a robot that change dynamically during a mission. For example, an unmanned aerial vehicle may have an autonomous take-off capability that does not require a human pilot, yet, later in the mission, a human pilot may be needed in order to recognize targets. For the former capability, the robot is responsible for all functions while, for the latter capability, the functions are shared by the robot and the human. The state may change dynamically, as well, due to a failure which requires human intervention, thus decreasing the number of functions the robot is responsible for. However, designers will often refer to the entire robot being at the level of the highest state of automation, rather than the lowest. This unfortunate situation leads users to have unrealistic expectations about the overall sophistication of the robot and enables designers to ignore ways to support the interplay between levels.

There are two other disadvantages. One is that treating the levels of autonomy taxonomy as an operational architecture results in a system that considers only the four deliberative functions, ignoring reactive behaviors. Another disadvantage is that levels of autonomy taxonomy considers the mode of interaction with the human supervisor but does not consider interaction with other agents or nonsupervisory interactions.

4.4.2 Autonomous Control Levels (ACL)

AUTONOMOUS CONTROL LEVELS (ACL)

The Autonomous Control Levels (ACL) operational architecture is sometimes used by the US Department of Defense as a roadmap for developing unmanned systems.⁴⁸ It is based on the OODA loop pronounced “ewe-dah,” which John Boyd, a famous Air Force pilot and strategist, used to codify how a successful pilot makes decisions. Boyd hypothesized that a pilot goes through a constant cycle of *observe, orient, decide, and act* (OODA) in order to complete a mission.²⁸

Like LOA, ACL divides autonomy into functions and levels, but, unlike LOA, the levels do not explicitly consider human/machine delegation. ACL has ten levels reflecting an ad hoc ranking that combines both increasing sophistication of which portions of each phase of the OODA loop are executed autonomously by an unmanned aerial system and their value for categories of military missions. Each level can be divided into three functions: *perception/situational awareness, analysis/decision making, and communication/cooperation*. The levels are:

- 9 *Battlespace Swarm Intelligence*. A group of robots can assign themselves roles and apply distributed tactics for the current situation.
- 8 *Battlespace Cognizance*. A robot understands the locations and trajectories of other agents and can opportunistically select targets.
- 7 *Battlespace Knowledge*. A robot or group of robots can detect and project enemy actions.
- 6 *Real-Time Multi-Vehicle Cooperation*. A group of robots work together and optimize their group activities to meet the tactical goals.

- 5 *Real-Time Multi-Vehicle Coordination*. A group of robots work in close proximity to each other to follow a tactical plan with human supervision.
- 4 *Fault/Event Adaptive Vehicle*. The robot is given the mission plan and rules of engagement (i.e., policies) and is allowed to adapt the tactical mission plan to enemy actions or to system failures.
- 3 *Robust Response to Real-Time Faults/Events*. The robot monitors its health, detects problems, and decides whether to adapt the plan or abort the mission.
- 2 *Changeable Mission*. The robot is given a tactical mission plan and is allowed to adapt the plan within bounds.
- 1 *Execute Preplanned Mission*. The robot executes a mission in an environment away from other agents.
- 0 *Remotely Piloted Vehicle*. The human is essentially in charge of all but flight stability.

The primary advantages of ACL are that it covers the wide range of military missions and relates functionality associated with intelligence to the familiar OODA loop formulation. The three functions have increasing levels of sophistication; for example, there is no analysis/decision-making function of a robot at level 0 but at level 3, the analysis/decision-making functionality includes fault identification, detection, and recovery. A secondary advantage is that ACL does take into consideration teams of robots, although it does not consider the interaction of robots working with humans. The upper levels do share functionality with entertainment applications, in particular, understanding where the other agents are and inferring their intent.

The disadvantages of ACL are similar to those of LOA, with two additional disadvantages. One is that the justification for the ranking of the levels is unclear and may delay applying robots to missions because the levels imply preconditions on the sophistication of the requisite intelligence. For example, consider swarming for applications such as humanitarian demining an area of land mines or sampling air or water quality. Many animals swarm. Wasps can swarm and coordinate actions to attack an intruder, which would correspond to levels 5, 6, and 9. But they also die while repeatedly trying to exit through a closed window because a wasp does not monitor its progress, which would require functionality that the ACL places at level 3 and 4. Biology suggests that agent swarming can be successful with less sophisticated intelligence, while ACL views as swarming as requiring more intelligence than task monitoring. This difference between biology and ACL is important because under ACL, swarming would not be designed and implemented until technology had advanced to the point of monitoring mission execution. Yet there may be missions where hundreds of cheap robots will be able to complete a mission without mission monitoring even if a large percentage of them fail. A second problem is that ACL does not appear suitable for assistive applications as there are no levels or functions that imply helping, not replacing, the pilot or people co-located with the robot.

4.4.3 Levels of Initiative

Another style of operational architecture is based on the amount of initiative that a robot is given to conduct a task. In this case autonomy is based on the political connotation of autonomy as self-governing rather than the mechanical control connotation discussed in chapter 3.3. Unlike the

previously discussed operational architectures, initiative is conceptualized by *roles* rather than by functions.

LEVELS OF INITIATIVE

Colman and Han⁴⁹ provide one example of a *level of initiative* operational architecture. Their work considers how a robot might be an intelligent soccer player. The robot would have at least two roles, one to be, say, a forward and the other to be a good teammate. At the lowest level, a soccer-playing robot with minimal intelligence would be able to use its perception and skills to fulfill its role, but nothing more; in general, the robot would not be allowed any initiative to change strategies. At a higher level, a robot midfielder would be able to apprehend that a teammate with the role of defender is injured and spontaneously change its role to cover both midfielder and defender roles.

Five levels of initiative are:

- *No autonomy*. The robot follows rigid programming for performing a task or achieving goal. A robot may always cover an opposing player in the same way or always pass the ball to another player. This level is similar to stimulus-response behaviors in a biological architecture and the 10 levels in the levels of automation architecture.
- *Process autonomy*. The robot can choose the algorithm or process to achieve its task goals. The robot may adopt a different strategy for covering different players. This choice is similar to the *selecting* function in a levels of automation architecture.
- *Systems-state autonomy*. The robot can generate and choose between options to achieve its goals. The robot might decide to keep the ball rather than pass the ball according to the playbook. This level is similar to the *generating* function in a levels of automation architecture but implies increased unpredictability and nondeterminism.
- *Intentional autonomy*. The robot can change its goals to meet the intent of its roles within the team, such as filling in for another player, where the intent to be a good team player is a higher priority than the intent to be a good midfielder. To do this, the robot may negotiate with others. This functionality introduces the notion of being aware of others and explicitly interacting with them, similar to the interface layer in biologically-inspired architecture.
- *Constraint autonomy*. The robot can create its own roles and goals, which, in AI, may involve relaxing constraints on its goals or how it accomplishes its goals. At this highest level of initiative, a robot may ask itself if it even wants to play. Constraint autonomy reflects the level of initiative associated with robots in movies, where they suddenly decide to take over the world. However, constraint autonomy is still subject to bounded rationality.

The levels of initiative operational architecture is intuitively appealing, with many conceptual advantages but at least one major disadvantage. Grouping functionality by initiative resonates with the definition of intelligence from chapter 1 as maximizing success; more intelligent agents try new ways of achieving success. The levels of initiative approach shares similarities with ACL in prioritizing the adaptability of a robot to emerging situations, yet is more focused on the robot's one-to-many relationship to other agents and the goals and tasks. Initiative involves selecting and generating alternatives, so there is an influence from LOA. However, the functions used to describe the ACL

(perception/situational awareness, analysis/decision making, and communication/cooperation) are required, in some form, for each level of initiative, and thus they are not a discriminator. One disadvantage of the levels of initiative architecture is that it lacks the specificity of LOA and ACL regarding which functions are needed to achieve the requisite autonomy for each level.

4.5 Five Subsystems in Systems Architectures

Operational architectures focus on the general style, while systems architectures focus on the general components. A house can be expected to have a kitchen, one or more bedrooms and bathrooms, a living room or den, and closets. Likewise the software for an intelligent robot normally has at least five subsystems, which are encapsulated as object libraries or similar reusable programming repositories. Systems architectural design encourages designers to think in terms of creating libraries of algorithms and data structures for each subsystem, similar to the modules and functions in MATLAB, Mathematica, and Maple, the ISML libraries in Fortran, and the Standard Template Library in C ++. These libraries serve as general clearinghouses from which the designer can pick specific functions for a particular robot in order to customize it for a new application.

The most common subsystems in the AI robotics literature are listed below. A typical configuration of the subsystems is also shown in [figure 4.6](#) in a datagram/actigram-like arrangement. Note the systems architecture diagram is very different than the canonical operational architecture; the former is a high-level diagram of how to program the robot. The subsystems are listed below.

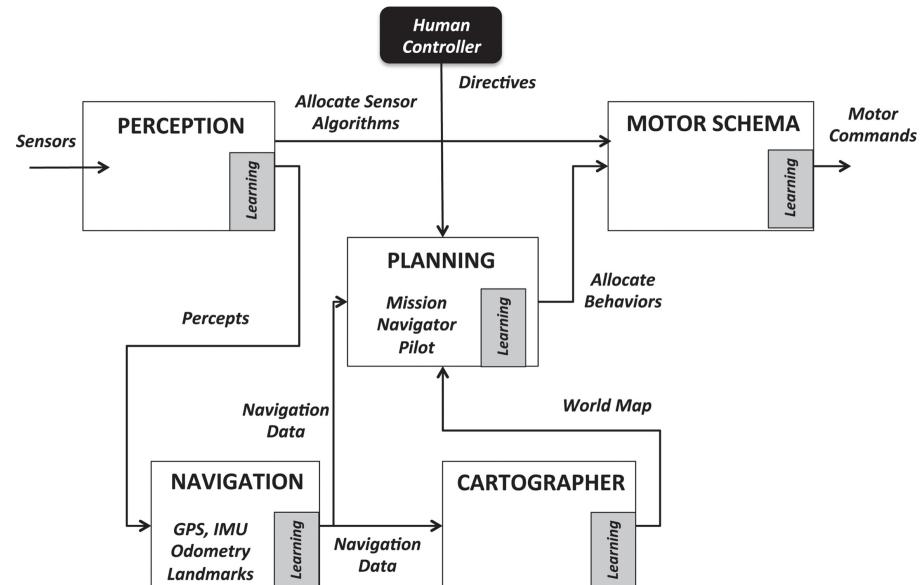


Figure 4.6 The five common subsystems in intelligent robotics, adapted from Technology Development for Army Unmanned Ground Vehicles.¹⁵⁸

- The **Planning** subsystem is associated with *generating* and *monitoring* overall mission objectives and passing along the geospatial component of those objectives to the navigation subsystem,

selecting the motor and perceptual resources, implementing or instantiating those resources, and monitoring the performance of the mission. This subsystem is host to the numerous classical planning and resource allocation algorithms.

- The **Navigation** subsystem is associated with *generating* and *monitoring* paths of movements and selecting the resources to accomplish the movements. This subsystem typically contains navigational knowledge needed to select the resources, get from waypoint A to waypoint B, decide on implementation parameters, such how fast to safely travel, and monitor progress. The subsystem may contain the path planning algorithms discussed in chapter 14, most notably the topological planners, the A* and D* metric path planning algorithms, or simultaneous localization and mapping algorithms.⁴³ Alternatively these algorithms may be stored in the Cartographer subsystem because they are coupled to the data structures for the World Model.
- The **Cartographer** subsystem, also known as the World Model or World Map, is associated with the construction and maintenance of knowledge representations about the world. For navigation-oriented robots, this subsystem is the key data structure that enables generating, monitoring, selecting, and implementing actions. These knowledge representations are often geospatial maps but can include more abstract symbolic information, such as the beliefs about the state of the world or the intent of other agents. In more deliberative robots, this subsystem bridges the gap between signals and symbols. In interactive robots, the subsystem also monitors the robot's mental models of itself and the beliefs, desires, and intents of other agents described in chapter 12.
- The **Motor Schema** subsystem, also known as the Motor Schema Library¹¹ or more generally as the Behaviors subsystem, is associated with *selecting* the best motor routines and *implementing* actions. This subsystem contains the functions that connect the deliberative and reactive functions with the requisite guidance, navigation, and control algorithms for the actuators and effectors. This subsystem typically does not have deliberative functionality but is rather about execution. It typically does not contain algorithms that monitor the overall success of an action in AI terms, for example, whether the robot is stuck in a “fly-at-the-window” loop. That type of monitoring is handled by the Planning subsystem, which, as the “home” of the plan for a robot mission, is the normal place for determining if the plan is being followed. The algorithms in the Motor Schema subsystem may be programmed to monitor their execution within a control loop.
- The **Perception** subsystem, also known as the Sensing, Perceptual Schema, or Perceptual Schema Library, is associated with *selecting* the best sensors for the motor actions and *implementing* sensor processing algorithms. As will be described in chapter 10, this subsystem contains the functions that control the sensors and extract the percept, that is, what is perceived. This includes algorithms that extract a special type of percept called affordances or those that perform object recognition and image understanding.

There are two important notes about the subsystems. First, the subsystems are not independent and do not represent a sequence of programming actions. Instead they are essentially the major objects or classes in object-oriented programming that reflect software engineering decisions about how to group similar functions and data. The “main” program (part of the technical architecture) employs subsets of the functions and data structures contained in the subsystems to produce capabilities. For example, a

specific behavior, such as avoid obstacles, takes the output of a function from the Perception subsystem (e.g., looming object) for use by a motor schema (e.g., turn in the most open direction) from the Motor Schema subsystem.

Second, the list of subsystems is not necessarily complete. The five common subsystems highlight the historical focus within the robotics community on autonomous navigation of a platform and mapping. These subsystems might be sufficient for an autonomous car but not for a surgical or entertainment robot. The five subsystems described above have no clear component for interacting with other agents or for manipulation. An intelligent robot in healthcare, entertainment, or even driving in city traffic would likely have additional subsystems.

4.6 Three Systems Architecture Paradigms

If the five subsystems correspond to the types of rooms and spaces in a house, architectural paradigms correspond to the basic floor plans connecting the subsystems. Historically, systems architectures for AI robotics fall into three categories called paradigms: *hierarchical*, *reactive*, or *hybrid deliberative/reactive*. These paradigms arrange the flow of data and control within the robot. This data and control flow for each paradigm can be uniquely described using two traits: the *pattern of interaction between the three robot primitives* of SENSE, PLAN, and ACT, and the *route of sensing*. This section begins by defining the two traits, then uses those traits to formally characterize the three specific styles of system architectures.

4.6.1 Trait 1: Interaction Between Primitives

Recall from chapter 3 that there are three basic primitives, SENSE, PLAN, and ACT, which are summarized in [table 4.3](#). Intelligent robots use one of three cyclic patterns of interaction between these primitives. One is to SENSE, then PLAN, and finally ACT, at which point acting changes the state of the world which leads to a new iteration of SENSE, PLAN and ACT and so on. This is signified as SENSE, PLAN, ACT where the “,” between SENSE, PLAN, and ACT indicates a sequence of primitives. But chapter 3 also described reactive animal intelligence, which uses a pattern of SENSE and ACT coupled into behaviors with no PLAN. The behavior is signified as SENSE-ACT where the “-” indicates coupling or concurrency between primitives. Neurophysiological models favor a different pattern of intelligence where higher brain functions PLAN asynchronously while SENSE and ACT behaviors both carry out plans and react to the environment. This pattern of intelligence is signified as PLAN, SENSE-ACT where there is both a sequence (plan, then instantiate sensing and acting) and coupling (execute sensing and acting until terminated or a new plan is instantiated). The three styles of interaction are shown in [figure 4.7](#).

Table 4.3 The three robot primitives and their inputs and outputs.

ROBOT PRIMITIVES	INPUT	OUTPUT
SENSE	Sensor data	Sensed information
PLAN	Information (sensed and/or cognitive)	Directives
ACT	Sensed information or directives	Actuator commands

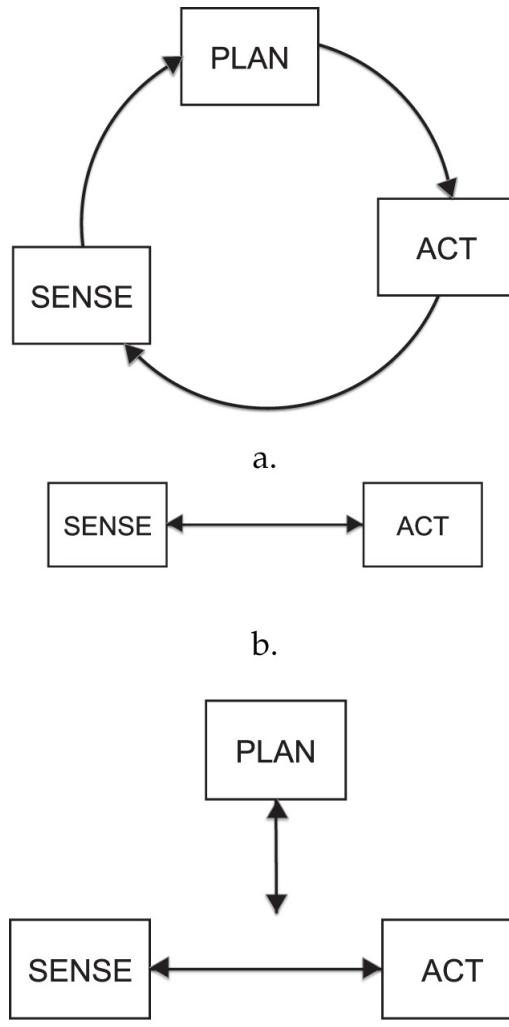


Figure 4.7 The three styles of interaction: a) SENSE, PLAN, ACT, b) SENSE-ACT, and c) PLAN, SENSE-ACT.

4.6.2 Trait 2: Sensing Route

Intelligence relies on sensing, and systems architectures typically use one of three routes described below by which sensing gets to the reactive, deliberative, or interactive functions that constitute an autonomous capability. The three routes represent design choices in computational sophistication and costs, latency, and whether to mimic sensing organization in animals.

The *local sensing* route goes directly from the sensor to the behavior or function using the sensor data. The receiving function is responsible for any transformations or conversions of the raw data into a suitable data structure. The transformations or conversions stay local, that is, within the programming scope, of that function. Local sensing can be a one-to-one or a one-to-many mapping, where the data

from one sensor can go to one or more independent functions.

Figure 4.8a shows the local sensing route from two sensors to three outputs for a hypothetical driverless car. The same camera data go to two different and independent functions, one to look for stop signs, which would enable the robot to stop, and the other to look for white lines for following the road. This is a *one-to-many* relationship. The lidar output goes to a third function that triggers a panic, or emergency, stop if anything is in front of the car. This is a one-to-one relationship.

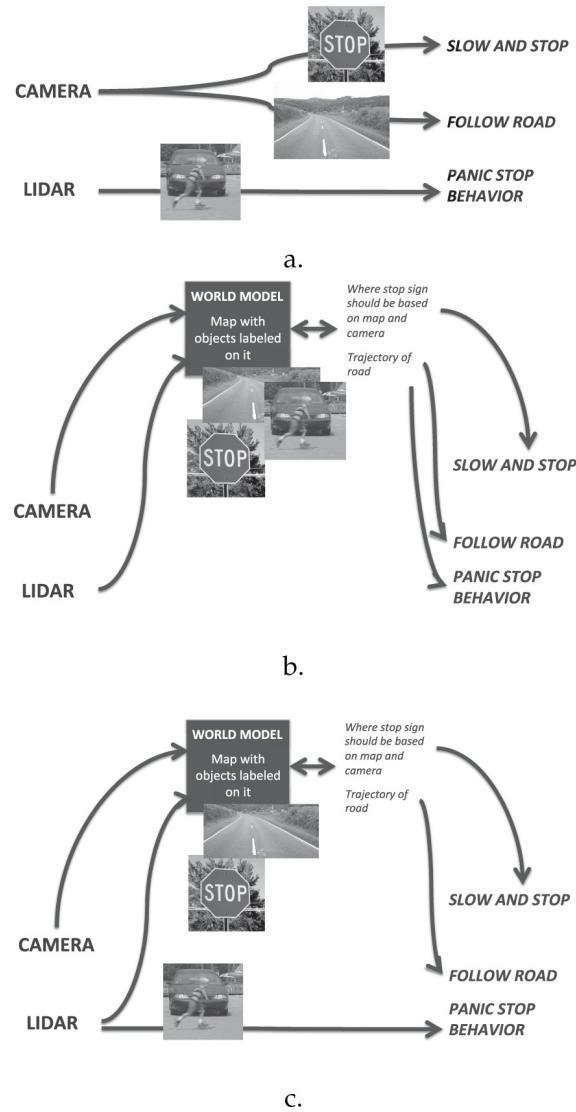


Figure 4.8 Example of the three sensing routes for a driverless car: local, global, and hybrid.

The *global sensing* route is how data originating from all the sensors is transmitted to a function that transforms and fuses the data into a global World Model or unified data structure. Global sensing can

be thought of as a *many-to-one* mapping, where the data from many sensors go to one function.

[Figure 4.8b](#) shows the sensing route from two sensors being fused into a World Model and then the World Model providing the location of the stop sign, the trajectory of the white lines, and the presence of anything in front of the car. This is a many-to-one relationship. Fusion into a World Model adds another step and can be computationally significant, thus adding a delay in sensor processing.

The *hybrid sensing* route is a combination, where the same data from a sensor may go to one or more functions that perform local transformations and to a global sensing function. Hybrid sensing can be thought of as a *many-to-many* mapping, where the data from many sensors can go to many independent functions.

[Figure 4.8c](#) shows the hybrid sensing route where the data from two sensors are being fused into a World Model but are transmitted directly to functions, such as a panic stop, which cannot tolerate the delay in processing

a world model or do not need a world model. This is a many-to-many relationship. This hybrid sensing route most closely mimics human sensing processes where neural signals go up the afferent paths and split.

4.6.3 Hierarchical Systems Architecture Paradigm

Hierarchical systems architectures organize the five subsystems to support a SENSE-PLAN-ACT pattern that relies on a global sensing route. As noted in Albus and Mystel,⁴ hierarchies are a natural way to organize functionality, and, if the priorities and goals are clear, they can be computationally efficient because they can reduce computation by specifying the frame and defining a closed world. Hierarchical systems are used for implementations where the mission or application is well-understood and further additions of capabilities or major upgrades are not expected. Automated manufacturing and guidance, navigation, and control systems tend to be hierarchical, trading off working in a broadly open world for being able to tightly control the execution flow. Hierarchical systems architectures are often used where there is little expectation of reuse of code or later expansion of functionality. Thus the goal is to optimize the programming.

The flow of a typical hierarchical architecture uses the subsystems in a sequence. The SENSE phase collects data from all sensors using routines from the Perception subsystem and then fuses sensor data into a World Model using routines from the Cartographer subsystem. The fusion process does more than fuse the current set of readings; it fuses the current readings with older readings and any a priori knowledge, such as maps. The PLAN phase applies routines from the Planner subsystem to the World Model to determine any changes to the missions or constraints and then plans navigational routes and movements using routines from the Navigation subsystem. The output is a set of Actions. The ACT phase executes the Actions using routines from the Motor Schema subsystem. Actions may also involve the Perception subsystem if sensors have actuators to be moved. The SENSE-PLAN-ACT sequence executes in a continuous loop.

[Figure 4.9](#) illustrates how an idealized hierarchical systems architecture might work for a driverless car. The program flow would begin with the SENSE phase. The data would be collected from the camera and LIDAR sensors and then fused with a priori knowledge into a World Model. From the World Model, the system would extract relevant perception, in this case, where the road was, that there was a stop sign present, and that there was an unexpected object in the road. The PLAN phase would

use the World Model to generate, update, or modify the mission plan, where the constraints were to reach point P , obey all traffic laws, and not hit anything in the road. It would then plan navigational actions, in this case to immediately stop to avoid hitting the object in the road but to stay on the road. The navigational subsystem would also note that the stop sign indicated that the car was at point P on the route and making good progress on the planned route. The ACT phase would execute the planned actions, stopping to avoid the unexpected object in the road.

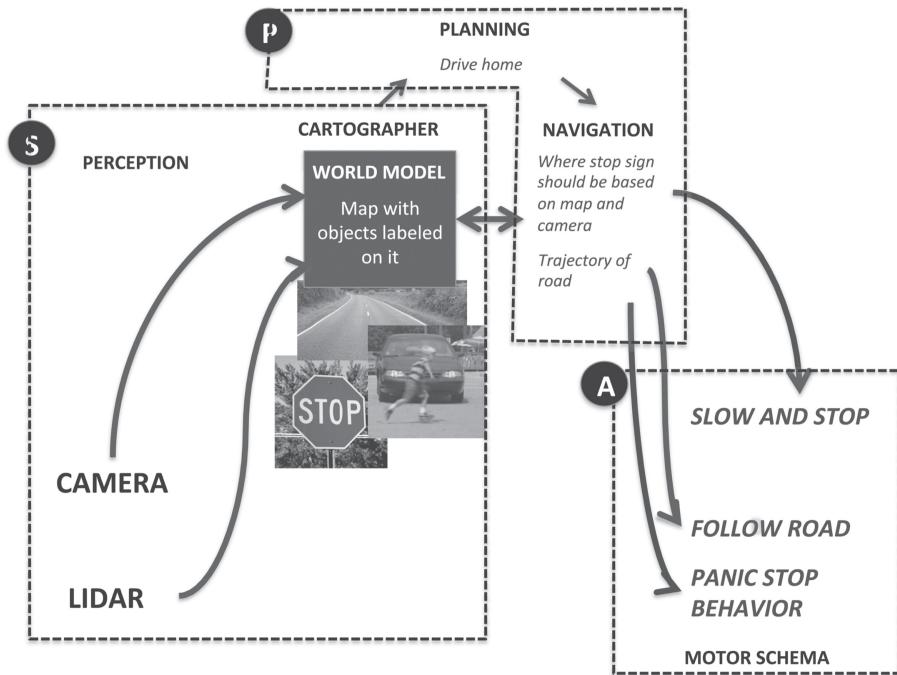


Figure 4.9 Example of a Hierarchical Systems Architecture for a driverless car.

The NIST Real-Time Control System (RCS) is the best known of the hierarchical architectural systems. Jim Albus at the National Bureau of Standards (later renamed the National Institute of Standards and Technology or NIST) anticipated the need for intelligent industrial manipulators, even as engineering and AI researchers were splitting into two groups. He saw that one of the major obstacles in applying AI to manufacturing robots was that there were no common terms and no common set of design standards. This made industry and equipment manufacturers leery of AI for fear of buying an expensive robot that would not be compatible with robots purchased in the future. He developed a very detailed architecture called the Real-Time Control System (RCS) Architecture in the early 1980s to serve as a guide for manufacturers who wanted to add more intelligence to their robots. It was adopted by the U.S. Army for several major projects, influencing how defense contractors organize code to this day.

The primary advantage of the Hierarchical Paradigm was that it provided an ordering of the relationship between sensing, planning, and acting. It was intuitively appealing and consistent with OODA loop control.

The primary disadvantage was planning. Every update cycle, the robot had to update a global World Model and then do some type of planning. The sensing and planning algorithms of the day were extremely slow (and many still are), so this introduced a significant bottleneck. Notice also that sensing and acting are always disconnected. This effectively eliminated any stimulus-response types of actions (“a rock is crashing down on me, I should move *anywhere*”) that are witnessed in nature.

4.6.4 Reactive Systems Paradigm

Reactive systems use only two subsystems, Perception and Motor Schema, of the five canonical subsystems to support a SENSE-ACT pattern that relies on a local sensing route. Programming by behavior has a number of advantages, most of them consistent with good software engineering principles. Behaviors are inherently modular and easy to test in isolation from the system (i.e., they support unit testing). Behaviors also support incremental expansion of the capabilities of a robot. A robot becomes more intelligent by having more behaviors. The behavioral decomposition results in an implementation that works in real-time and is usually computationally inexpensive. Generally, the reaction speeds of a reactive robot are equivalent to stimulus-response times in animals, although if the technical architecture choices are implemented poorly, then a reactive implementation can be slow. Purely reactive robots are useful for simple autonomous systems, such as the iRobot Roomba vacuum cleaner, robot mowers, and entertainment robots, where the device uses biomimicry to emulate insect or very basic animal behaviors where planning and optimality are not essential. Reactive systems architectures can also be used for adding autonomous capabilities to an existing human-machine system where a rapid response is important, such as a car autonomously breaking to avoid hitting another car or obstacle. Researchers tend to view reactive systems as a lower layer in the broader hybrid deliberative/reactive architecture.

The flow of a typical reactive architecture uses multiple instances of the Perception and Action subsystems in parallel. The SENSE phase collects data from all sensors using routines from the Perception subsystem and directly supplies them to the Motor Schemas which produce Actions (the ACT phase). There may be multiple SENSE-ACT pairs, or behaviors, and these behaviors run concurrently and independently. Each behavior is a local loop and may have a different computation time, and thus the behaviors work asynchronously. Sensing is routed to each behavior without any filtering or processing and each behavior extracts the direct perception that is unique to it. A sensor can supply the same sensor reading simultaneously to different behaviors, and then each behavior compute a different percept. The sensor processing is local to each behavior with no global model of the world. There is no PLAN phase as behaviors produce Actions only if the sensing produces a reaction.

Figure 4.10 illustrates how an idealized reactive system architecture might work for a driverless car. The program flow would begin with the SENSE phase. The sensor data from the camera would be directed to the “stop at stop signs” behavior, and the behavior would determine whether the brakes should be applied. At the same time, the sensor data from the camera would be split off and directed to the “follow road” behavior, and the behavior would compute the steering correction. The sensor data for the “panic stop” behavior would also be collected from the LIDAR and a braking action computed if needed. The ACT phase would apply the coordination function, as two behaviors both contribute brake actions. For example, the “stop at stop signs” behavior might be saying “no brakes” because no stop sign is visible while the “panic stop” might be saying “hit the brakes!” because there is an obstacle

in the road. The coordination function would resolve the contention between behaviors and allow the panic stop to execute.

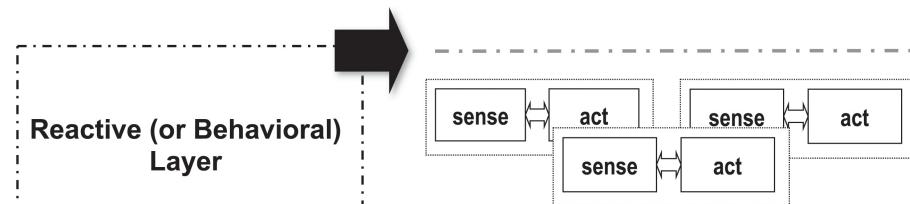


Figure 4.10 Example of a Reactive Systems Architecture for a driverless car.

A robot constructed only with the reactive layer is often called either a reactive or behavioral robot. The iRobot Roomba is possibly the best known example of a reactive robot. The reactive layer, championed by Rodney Brooks³⁰ and Ron Arkin,¹¹ was the major focus in AI robotics research from 1986–1996. Researchers explored how an agent could maximize its chances of success using only the information directly perceivable in the world and without planning. Brooks' Subsumption Architecture³⁰ is the best known style of implementing reactivity as a stand-alone systems architecture. Subsumption organizes behaviors into layers of competence and utilizes subsumption and inhibition mechanisms to serve as the coordination function for the multiple layers. Subsumption will be discussed further in chapter 8. Potential fields are another popular coordination mechanism for behaviors, and they were made popular by Arkin,¹¹ based on Khatib's original work.¹⁰³ The reactive layer is generally programmed in a procedural language such as C, C++, or Java, but, it can be programmed in Lisp.

A behavioral robot has at least three advantages. First, direct perception is usually simple to program. Second, the behaviors are highly modular. This modularity means that new behaviors can be added without reprogramming existing working behaviors. Behavioral robots tend to degrade gracefully as a failure on one behavior does not cause the other behaviors to fail, though this depends on the technical architecture. Third, a reactive or behavioral robot can work well in cooperation with a human in telesystems, because the robot can react to stimuli, such as proximity, to trigger panic behaviors (stop!) or guarded motion behaviors (robot won't move closer than is safe regardless of human input). Behaviors often appear as automatically triggered “macros” such as a wireless robot returning to home after loss of signal, self-righting, and recharging.

A robot with only a reactive layer has three disadvantages. One disadvantage is that the inability to PLAN leads to the “fly-at-the-window” effect introduced in chapter 3. The effect occurs because the robot is sensing and acting locally over what it can perceive at that instant in time. It has no way to monitor itself over time and discover it is stuck in a loop. A second disadvantage is the emergence of the overall behavior from individual behaviors using direct perception makes it difficult to precisely predict what the robot will do; that is, it adds to the non-determinism of the system. For example, an **avoid** behavior may drive the robot either to the left or the right of an obstacle, depending on the sensor readings. Third, the behavior may not be optimal because it is based on local, instantaneous sensing rather than on a plan generated from a global world model.

4.6.5 Hybrid Deliberative/Reactive Systems Paradigm

HYBRID DELIBERATIVE/REACTIVE SYSTEMS

Hybrid deliberative/reactive systems architectures organize the five subsystems to support a “PLAN, then SENSE-ACT” pattern coupled with hybrid sensing. The idea of the PLAN, then SENSE-ACT pattern evolved both from biomimicry and from two software engineering assumptions. First, planning covers a long time horizon and requires global knowledge, so it should be decoupled from real-time execution on the software engineering principle of coherence (dissimilar functions should be placed in different objects). Second, planning and sensor fusion algorithms are computationally expensive, so they should be decoupled from real-time execution of behaviors because they would slow down the reaction rate. The World Model is constructed by processes independent of the behavior-specific sensing. Hybrid systems use hybrid sensing routes. In many hybrid architectures, the Cartographer or model making processes can access not only the same sensors as the behaviors but also the results of local sensing. The Cartographer can also have sensors that are dedicated to providing observations which are useful for world modeling but are not used for any active behaviors. Hybrid systems are the most flexible and have become the default systems architecture for researchers and for applications where the robot’s functionality is expected to be modified or extended in the future.

[Figure 4.11](#) illustrates how an idealized hybrid system architecture might work for a driverless car. The program flow would begin with the SENSE phase. The sensor data from the camera would be split into two streams, one contributing to the formation of the World Model in the Cartographer, the other directed to the “follow road” behavior in the Motor Schema subsystem. Similarly, the sensor data from the lidar would be split and streamed to the Cartographer and “panic stop” behavior. The “follow road” and “panic stop” behaviors would execute faster because they are using local sensing reflexively. The Cartographer would be slower because it has to take the extra steps of processing the sensing and then integrate the extracted perception into a global world model. From the World Model, the system would extract the relevant perception needed for Planning in the PLAN phase and for use in Navigation and action. The output of the World Model can serve as a virtual sensor stream to the behaviors.

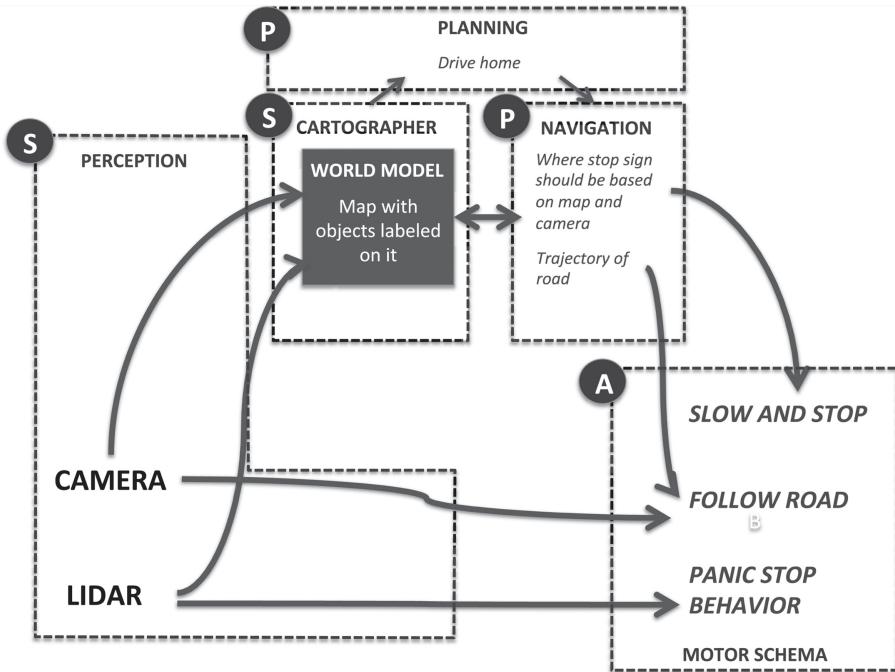


Figure 4.11 Example of a Hierarchical Systems Architecture for a driverless car.

The 3T, or 3-Tiered, mobile robot architecture is the best example of a state-hierarchy system and is predominately used at NASA. As the name suggests, 3T divides itself into three layers, one clearly reactive, one clearly deliberative, and one which serves as the interface between the two. 3T has been used primarily for planetary rovers, underwater vehicles, and robot assistants for astronauts.

Hybrid systems are the most flexible and have become the default systems architecture for researchers and for applications where the robot's functionality is expected to be modified or extended in the future.

A disadvantage of hybrid systems is the added programming complexity of managing asynchronous processes. The SENSE-ACT behaviors run independently and very fast while planning routines and global sensing update at a slower rate.

4.7 Execution Approval and Task Execution

The previous sections demonstrate that building an intelligent robot is a large undertaking. The many facets of deliberation and reaction pose the concern as to whether we have algorithms and coordination methods to actually build a fully autonomous robot that can work in the open world. One strategy is to build the robot but have a human supervisor check the robot's planned actions before it executes them; this strategy is known as *execution approval* and *task rehearsal* in the human factors literature. For example, the DARPA Robotics Challenge expected the teams to simulate the execution of a plan to confirm that it should be successful when deployed.¹⁴² Essentially it changes the hierarchical SENSE-PLAN-ACT cycle to the SENSE-PLAN-APPROVE-ACT arrangement shown in figure 4.12,

introducing the human into the loop to view the simulation and approve the next action. Unfortunately, SENSE-PLAN-APPROVE-ACT has not been shown to be very effective, though it is appealing.

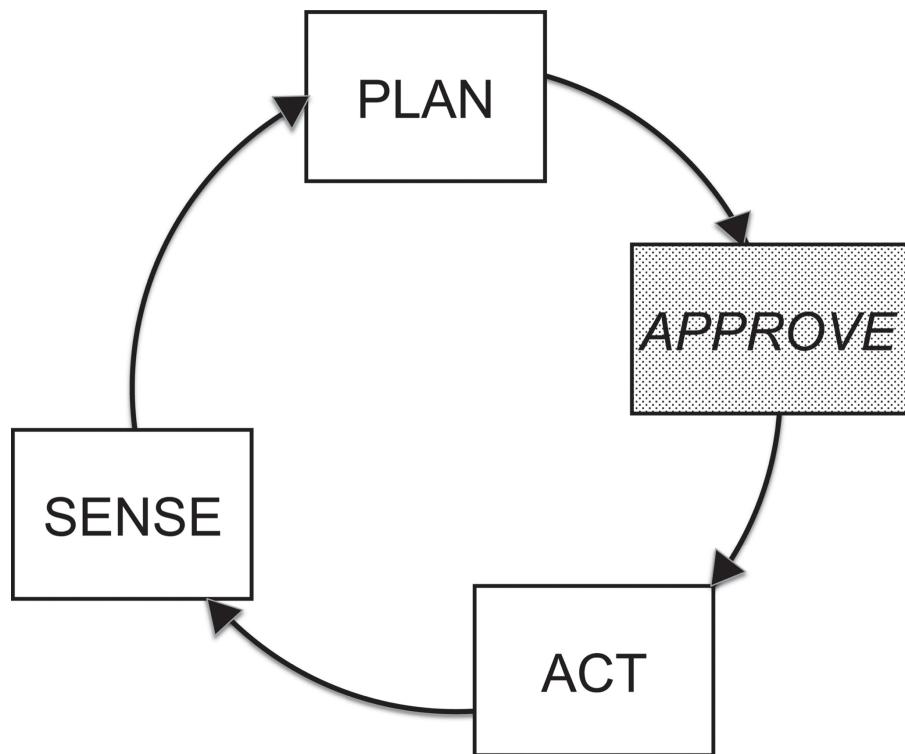


Figure 4.12 SENSE-PLAN-APPROVE-ACT cycle.

EXECUTION APPROVAL

This type of checking found in figure 4.12 is often implemented as *execution approval*,¹⁴² where the cognitive agent, either human or computer, considers the next step and decides whether it is safe. As described by Klein,¹⁰⁴ human execution approval typically results in low performance because humans are not good at considering all the ways the planned action can go wrong. Computers can be more comprehensive, but generally execution approval has been the purview of a human.

TASK REHEARSAL

A similar approach for verifying the plan is to consider an entire sequence of actions, which is known as *task rehearsal*.¹⁰⁴ A notable example of task rehearsal is a nuclear processing cell, where the layout of the building and every piece of equipment is known. That knowledge is used to generate a computer simulation of whether a particular path for a robot handling nuclear material would collide with other equipment and cause a spill. The operator can view the simulation and confirm that there is no collision or unexpected behavior. Unfortunately, people are not good at catching problems with task rehearsal. Klein¹⁰⁴ shows that people cannot manage more than three variables or six transitions.

The low performance of humans at imagining all the ways things can go wrong or rehearsing what

to do when things go wrong suggests that the robot should ultimately be the one to verify its plan, or at least create more trustworthy plans. Fortunately, execution approval and task rehearsal may not be needed, or even appropriate, for all actions. If a piece of the ceiling is falling on the robot, it needs to get out of the way immediately or risk coming up too late with the best place to move in order to avoid being crushed. The latency in projecting the consequences can literally be the death of a robot.

4.8 Summary

This chapter has considered autonomy in terms of desired capabilities and discussed how those capabilities fit within a general programming framework. The programming framework is presented as an operational architecture, a high level description of what a robot does, and a systems architecture, specifying the general systems needed to implement the functionality. An architectural framework has several conceptual advantages. It describes the general arrangement of functions in an intelligent robot using the software engineering principles of abstraction and modularity, and it serves as a semi-formal design specification and checklist for completeness addressing the question: *Are all the functions accounted for in a specific robot design?* If some are missing, is that an acceptable vulnerability or does it have to be addressed?

The AI robotics community has converged on a canonical operational architecture consisting of three layers of reaction, deliberation, and interaction. The canonical operational architecture separates reactive behaviors (reaction) from the cognitive functions of generating, selecting, implementing, and monitoring plans and resources (deliberation) and adds an interaction layer for communication and coordination with other agents. This software organization merges biological insights with the Levels of Automation and Autonomous Control Levels and Levels of Initiative styles of architectures favored by manufacturing, the US military, and the multi-agent communities.

The systems architecture that best expresses this operational architecture in terms of subsystems is the *hybrid deliberative/reactive architecture* or “hybrid” for short. Hybrid systems architectures use a PLAN then SENSE-ACT flow with distributed sensing and reactive and deliberative functions that operate at different time scales. There is a temptation to insert a human into the loop to check the robot’s plans before execution, moving the flow to a SENSE, PLAN, APPROVE, ACT sequence, but there is evidence from the human factors community that this rarely works as expected.

The chapter overview posed two questions, each of which has been partially answered. *Given that autonomy has a different programming style, what is it?* The programming style for autonomy is to think in terms of biological metaphors, capturing the reaction, deliberation, and interaction needed for the robot to be successful as a complete system. The biological metaphor leads to a canonical operational architecture consisting of three layers: a Reactive Layer made up of SENSE-ACT behaviors often programmed in the technical architecture using a procedural language, such as C++; a deliberative layer that maintains a world model, mapping symbols to ground and providing generating, selecting, implementing, and monitoring functions, which are often programmed in a functional language, such as Lisp; and an Interactive layer that enables the robot to communicate and work with humans and other robots or software agents and often programmed in OWL or XML to take advantage of ontologies and relationships.

The answer to the second question, *Can intelligence be added in layers such as upgrading to a “pro*

version” or downloading “apps” as needed? is both “yes” and “no.” On the “yes” side of the discussion, general purpose intelligence is organized in layers of abstraction and modular functions. A robot may not need all the layers for a particular task; for example, a Reactive robot may not need deliberation and interaction. However, the answer is “no” because a successful autonomous capability, such as navigation, generally requires all the layers and functions be considered in the design. For example, either the robot internally handles plan monitoring because monitoring isn’t important for that application (e.g., we don’t mind that the robot vacuum cleaner gets stuck from time to time), or the human has to monitor manually. A major class of mission failures stem from the designer neglecting to explicitly consider the entire set of deliberative functions, for example providing plan generation but not plan monitoring. Not only does this result in mission failures per se but the human has to try to prevent or recover from those missing capabilities. This leads to increased manpower demands and there is no guarantee that a human can perform those functions or can manually recover from a mission failure.

Sadly, the three layers in the hybrid architecture, or any operational, systems, or technical architecture, are not a roadmap for incrementally building a robot. The software engineering elegance of modular operational and systems architectures is not the same as a roadmap. The robotics industry and military procurement agents have typically treated operational architectures as an incremental development model, essentially saying the industry should first focus on robots that can handle one layer, then progress to the next set of modules or layers, and so on. This mindset undercuts the idea that even a “simple autonomous” capability may require reaction, deliberation, and interaction. The layers reflect functionality, and the subsystems and coordination reflect the general layout, similar to types of rooms and general floor plans. The layers do not suggest that one subsystem should be built first, then the robot used (or house occupied), then the next subsystem should be build, and so on. People want a working robot or a complete house. Robots, like houses, are generally not designed for ease of addition —however, if the owner knows that the house will be expanded over time, then it is designed and constructed differently than a typical tract home.

This chapter may have raised a third question, which will be deferred to chapter 19 on design: *How much artificial intelligence does a robot need?* How much artificial intelligence depends on the desired capabilities for the robot. An inexpensive vacuum cleaning robot can be successful by emulating simple animal grazing behaviors while a multimillion dollar robot expected to travel to deep space may have to employ sophisticated path and mission planning. This chapter has helped set the stage for chapter 19, which will pose five questions to help focus the design on providing sufficient intelligence to match the expectations for the robot, the complexity of the environment, and abilities of the robot. One question is *what functions does the robot need to provide?* These functions are generally behaviors, deliberative functions (generation, monitoring, selecting, implementing) or learning? A second question is *What planning horizon do the functions require?* This chapter has described the Present (work only with immediate information), the Past and the Present (use knowledge of the past with immediate information), and the Future (projecting). Another question is *How fast do the algorithms have to update?* That depends on whether the function is for real-time control, which may need guaranteed execution rates, or for planning which may take longer and does not need to be performed as frequently. The fourth question is *What type of model does the robot need?* Models can be local to a specific task or require the construction and maintenance of a larger, task-independent

model. An often overlooked question is *Where does the human come in?* The chapter has described interactions but also has introduced the presumption of a human performing fine-grained monitoring of the robot.

The layout of the canonical operational architecture and major subsystems in the hybrid systems architecture will provide a basis, or skeleton, for anchoring the remainder of the algorithms covered in this book.

4.9 Exercises

Exercise 4.1

Define the difference between *operational*, *systems*, and *technical* architectures.

Exercise 4.2

Name the layer in an operational architecture that each function below is most likely to reside in:

- Robot nods as a person speaks directions.
- Robot loses wireless communication with its controller and returns home.
- Robot plans an optimal path to search a building.

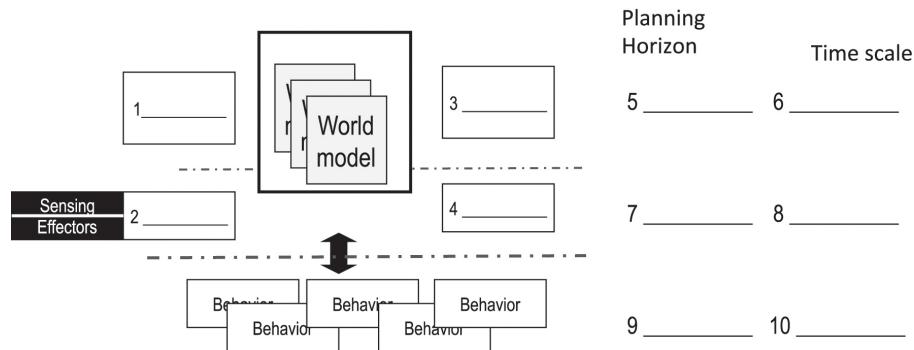
Exercise 4.3

Name the three systems architecture paradigms, and draw the relationship between the primitives.

Exercise 4.4

Label the components and attributes of the canonical hybrid architecture below:

For each layer/sub-layer:



Exercise 4.5

Name and describe the three routes of sensing in a software systems architecture.

Exercise 4.6

What is the difference between *execution approval* and *task execution*?

Exercise 4.7

Which one is not a reason why execution approval and task execution do not ensure safe or correct actions of a robot:

1. It is hard to imagine all the ways a plan or action can go wrong.

2. People cannot effectively rehearse tasks with large numbers of variables and transitions.
3. People inherently trust a computer.

Exercise 4.8

Why does the functional decomposition and layers in operational architectures typically fail as roadmaps for incrementally building autonomy?

4.10 End Notes

Robot paradigm primitives.

While the SENSE, PLAN, ACT primitives are generally accepted, some researchers are suggesting that a fourth primitive be added, LEARN. There are no formal architectures at this time which include this, so a true paradigm shift has not yet occurred.

H.A.L. got it right.

H.A.L., the artificial intelligent ship in *2001: A Space Odyssey* (1968), is one of the most accurate examples of designing for intelligence in science fiction. Dave Bowman, the mission commander, only shuts down the “higher consciousness” portions of H.A.L., leaving the autonomic functions that run the ship’s air recycling, navigation, and other essential operations intact. Not only is H.A.L. programmed modularly, each module has its own hardware processor. See a short article in *Science Robotics* on the 50th anniversary of *2001: A Space Odyssey*.

Science fiction and why it’s better not to add intelligence to your robot at the very end.

A common theme in bad science fiction movies is to have a scientist build a robot that is perfect in every way except for having intelligence, which is then supplied by downloading a copy of a human brain. And not just any brain, a flawed human brain. Two particularly so-bad-it’s-good examples of these are *Saturn 3* (1980), where it is almost impossible not to hope that the robot will kill Farrah Fawcett, and *Eve of Destruction* (1991), where the plot twist is that the scientist, who downloads her brain into an uninhibited robot with a small tactical nuclear bomb in its thigh, has a long history of failed relationships and desire for revenge.

Ender’s Game.

An article in WIRED Magazine (November 4, 2013) argues that Ender’s Game is an example of how to successfully use Boyd’s OODA loop, which is the motivation for the Autonomous Control Levels.

5

Telesystems

Chapter Objectives:

- Define *telesystem*, *supervisory control*, *teleoperation*, *shared control*, *traded control*, *remote control*, *telepresence*, *proprioception*, *exteroception*, and *exproprioception*.
- Compare and contrast *taskable agency* and *remote presence*.
- Label the seven components of a telesystem.
- Describe the *teleoperation time heuristic*.
- Define the three types of perception and why exproprioception is critical for robust telesystems.
- Justify whether or not a domain is suitable for a telesystem using the six characteristics of a telesystem domain.
- If given a description of a human supervisory control scheme, classify it as manual, traded, shared, or autonomy.
- Define *guarded motion* and explain its relationship to traded and shared control.
- Write out the formula for the *safe human-robot ratio* and explain what each term means.
- Describe the *human out-of-the-loop (OOTL) control problem* and why it is a concern for human supervisory control.

5.1 Overview

TELEOPERATION

To recap, artificial intelligence is good at lower-level behaviors and skills in the Reactive layer and at symbolic reasoning in the Deliberative layer but not so good at converting sensor data into symbols. In addition to this cognitive limitation, robots are mechanically disadvantaged and have trouble performing manipulation tasks that require hand-eye coordination. Given that humans still exceed machines at sensing and at dexterous manipulation, it is often desirable to have systems that enable humans and robots that are physically separated to interact to accomplish the task. This human-robot arrangement is called a *telesystem*, where *tele* means “remote,” and the general activity of the human

operator controlling the robot from a distance is called *teleoperation*. The two terms, telesystem and teleoperation, are used interchangeably. The connotation of telesystems is that the human *must* interact with the world *through* the robot and would prefer to delegate tasks to the robot rather than micromanage every movement. More recently, tasks such as telecommuting, surveillance, warfighting, and search and rescue have emerged that involve a person who *wants* to work through a robot in order to understand the remote environment or situation in realtime. These human-wants-to-be-in-the-loop tasks are a form of remote presence, which will be discussed in the next subsection.

Teleoperation and the use of telesystems is the state of the practice in most military and public safety robots. Use of telesystems is also a hot new trend in consumer robotics, given the increase in telecommuting robots, such as the Double Telepresence Robot for office workers and RP-Vita for doctors, and the proliferation of small unmanned aerial vehicles as toys and for commercial applications. Thus, it is important to understand teleoperation both as a legitimate, stand-alone approach to robotics for remote presence applications and as a transitional path to autonomous taskable agents. Autonomous capabilities can assist operators in remote presence applications and can enable taskable agency.

This chapter addresses questions on teleoperation, in particular, *What is a telesystem?* Having a human work through a robot instead of totally delegating a task to a robot presents the question, *Is teleoperation a “temporary evil” on the path to autonomy or is it a different style of AI?* The short answer is that teleoperation is a different style of AI, called remote presence. Unlike taskable agency, the purpose of remote presence is to allow the human to perceive and act in real-time at a distance. This human-machine system pairing is one that involves considerable thought as to *What task domains are telesystems good for?* The chapter will delve into more technical issues, answering *What is human supervisory control?, What is semi-autonomy?, and How are they different?* Human factors are often overlooked in telesystems even though the human and robot work intimately together, so the chapter covers the relevant human factors that must be taken into account to ensure that the operator(s) cognitive limitations are not exceeded and that the human out-of-the-loop control problem does not occur.

5.2 Taskable Agency versus Remote Presence

Telesystems can be either an alternative to full autonomy for taskable agents or an “end state” for remote presence applications, so it is helpful to distinguish between these two concepts.

TASKABLE AGENT

A *taskable agent* domain is one where the robot is given a complex task or mission, executes it without supervision, and then returns or informs the human. One example is the Deep Space One probe. In fiction, an example is the Terminator. In both cases, once the robot begins execution, the human has little control over the robot and must trust that the robot will do the right thing.

REMOTE PRESENCE

A *remote presence* domain is one where human and robot share the task, and the task execution is blended. This type of application is also called a *joint cognitive system*²²⁰ because the human and robot must jointly divide and share the cognitive activities to accomplish the mission. An example is a

surgical robot which is directed by a doctor but uses artificial intelligence to sense the body and prevent cuts penetrating too deeply or venturing into the wrong place.

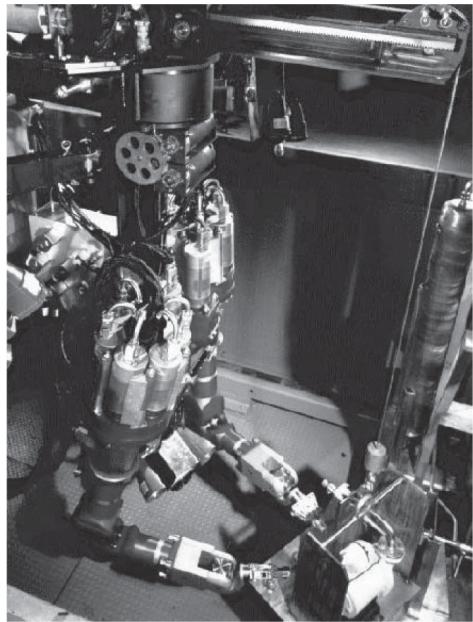
The two objectives are different. Recall the four foci of robotics introduced in chapter 1: Replace, Project, Assist, and Amuse. Taskable agents are designed to Replace humans. Remote presence robots are designed to allow humans to Project themselves into distal environments.

5.3 The Seven Components of a Telesystem

A telesystem has seven components, two at the local workstation, sometimes called the *master*, four at the remote robot or *slave*, and a communication component to connect the local and remote.²¹¹ See figure 5.1.



Master



Slave

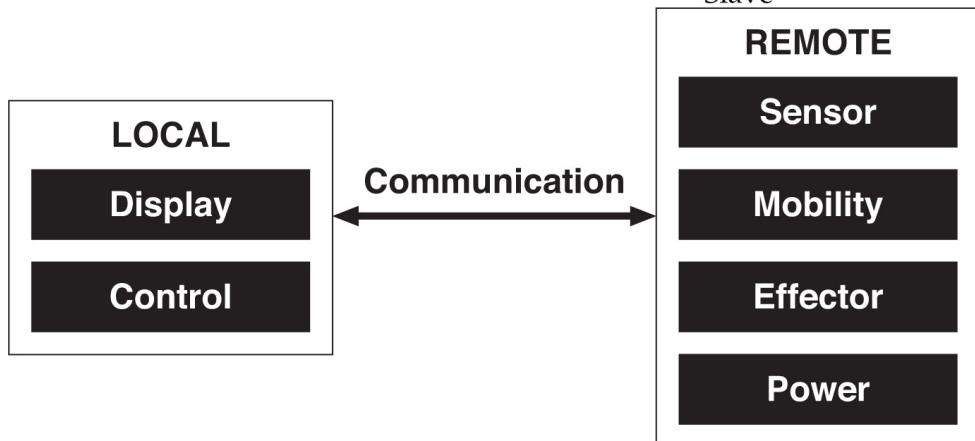


Figure 5.1 Organization of a telesystem. (Photographs courtesy of Oak Ridge National Laboratory.)

LOCAL

DISPLAY

The *local* is where the human operator, or teleoperator, works. The local must have some type of *display* to see the state of the robot and *control mechanisms* to direct the robot. The display and control mechanisms are usually configured as a workstation. The control mechanisms may be a keyboard, a joystick, a joystick with force and tactile feedback, such as the PHANTOM haptic interface,¹²¹ or a more complex manipulator which translates hand and arm motions into robot manipulator motions (see figure 5.2).



Figure 5.2 Manipulator interface used in nuclear hot cell work. (Photograph courtesy of Central Research Laboratories.)

REMOTE

SENSORS

DISPLAY

COMMUNICATION LINK

The *remote* robot, sometimes called the *telefactor*, must have *sensors*, *effectors*, *power*, and *mobility*. Effectors and mobility may seem redundant as mobility is enabled by effectors, but originally teleoperation was limited to manipulators, and thus, when mobile robots were created, mobility was considered a new component. Another change over time in technology is that control may be

distributed between the local and the remote, allowing the remote telefactor to have some intelligence and autonomous capabilities. The teleoperator generally cannot look at what the remote is doing directly, either because the robot is physically remote (e.g., on Mars) or the local has to be shielded (e.g., in a nuclear or pharmaceutical processing plant hot cell). Therefore, the *sensors* which acquire information about the remote location, the *display* technology for allowing the operator to see the sensor data, and the *communication link* between the local and remote are critical components of a telesystem.²¹¹

5.4 Human Supervisory Control

HUMAN SUPERVISORY CONTROL

Now that what a telesystem is has been established, the next question is *How is a telesystem controlled?* Telesystem control generally falls under the heading of *human supervisory control*. Human supervisory control is defined as when one or more human operators are intermittently giving directives and continually receiving information from a computer that, itself, closes an autonomous control loop through artificial effectors and sensors to the controlled process or task environment.¹⁹¹ The term is not limited to unmanned vehicles; it can be applied to airplanes with autopilot and fly-by-wire capabilities as well as to factory automation.

The definition of human supervisory control implies that a human is *always* involved with a robot, if only to set objectives for an autonomous taskable agent. Even if the robot is autonomous, it is providing information because even the lack of information is informative (e.g., nothing interesting has happened). The definition also indicates that a computer is always involved. The computer may project the effect of future actions for execution approval or task rehearsal, as described in chapter 18. It may compensate for time delays, which will be discussed in section 5.5.2. The computer certainly provides the inner-loop control, enabling actual actions. An example is the fly-by-wire ability of quadrotor UAVs to hover in place without a human touching the controls. It is desirable for a robot to have safety/self-protection reflexes, also known as *guarded motion*, as described in Pratt and Murphy.¹⁷⁰

In teleoperation, the human generally supervises a robot via one of three modes: *manual control*, *traded control*, and *shared control*. The modes will be described in more detail below. However, it is important to keep in mind that the mode may change with the timeline or phase of a mission, thus the modes were intended to be labels for the *current state of the joint cognitive system*, not a fixed, unchanging operating scheme. For example, an operator may lightly guide a unmanned aerial system during takeoff and landing (*shared control*) and then directs the robot to fly autonomously with no supervision to a waypoint. Then the human takes over to look at the situation (*traded control*), then re-engages full autonomy to let the UAV fly itself back home, finally guiding the UAV to the best landing spot (*shared control*).

5.4.1 Types of Supervisory Control

The types of supervisory control can be represented as a matrix with two orthogonal axes. The axes frame human supervisory control as the answer to two questions: *Can you see the robot in its environment?* and *Where is the major part of the intelligence: the local (operator) or the remote (robot)?* This leads to four quadrants as shown in [figure 5.3](#).

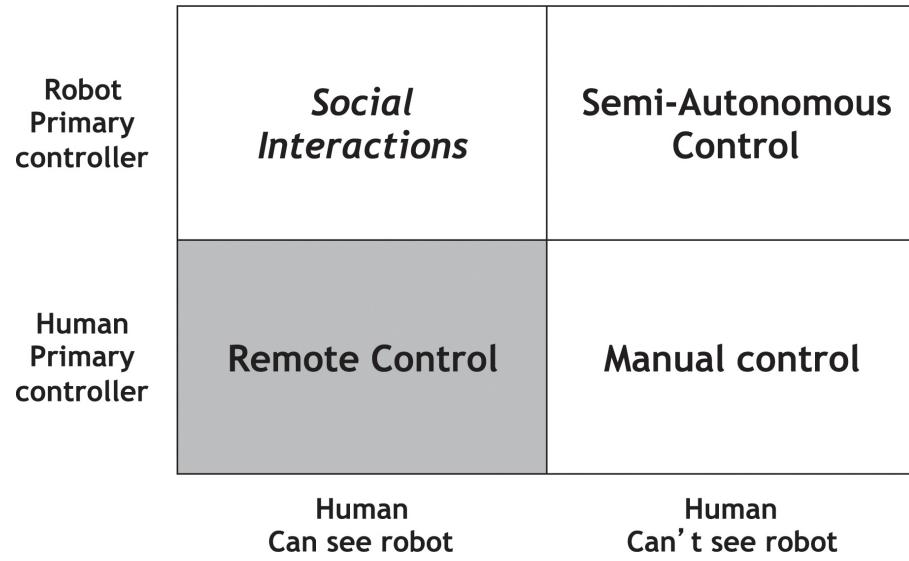


Figure 5.3 The types of supervisory control represented as quadrants.

REMOTE CONTROL

The lower left quadrant in [figure 5.3](#) shows that one means of supervising a robot is by *remote control*. In remote control, the operator can see the robot, and the robot has little, if any, intelligence. Remote control, also called radio control, is standard for “robot war” competitions and flying toy UAVs. Remote control is generally *not* considered teleoperation, as teleoperation connotes the operator cannot see the robot.

MANUAL CONTROL

The *manual control* quadrant is the classic teleoperation regime. In this quadrant, the operator cannot see the truly remote robot, and the robot has little intelligence.

SEMI-AUTONOMY

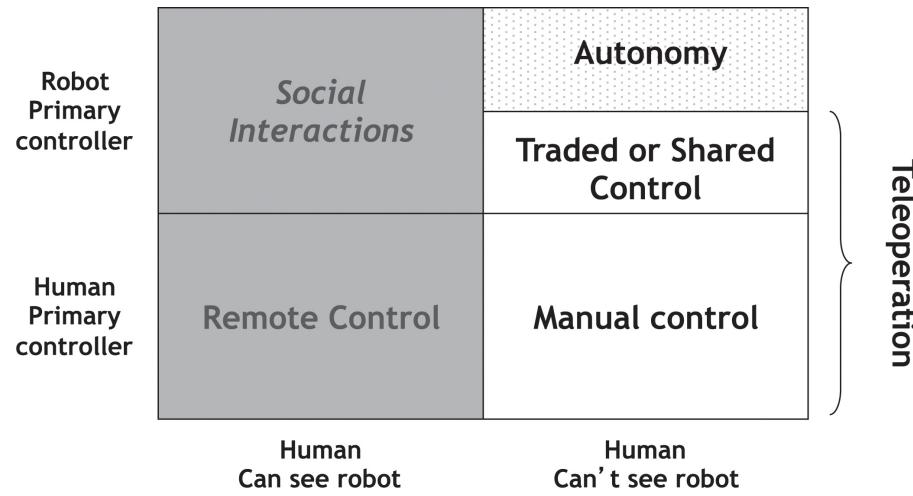
The *semi-autonomy* quadrant also includes teleoperation. In Semi-Autonomy, the robot has intelligence, but it may not have a set of capabilities that can totally substitute for human intelligence. Likewise, even if it has such a set of capabilities, there is a human giving at least initial directives to the robot, so the term “semi” still applies. Much of human supervisory control concentrates on the rich interactions between humans and robots that occur in this quadrant.

SOCIAL INTERACTIONS

The fourth quadrant, *social interactions*, is the newest form of human supervisory control. It covers situations where the human can see the robot and must interact with it, even though the human does not control the robot. The controller could be a coworker who has to talk to a colleague through a telecommuting robot. Human supervisory control has historically been about the human “behind” the robot, in effect, the Remote Control, Manual Control, and Semi-Autonomous Control quadrants, but now it has to consider the human “in front” of the robot. This quadrant will be the subject of chapter 18 on Human-Robot Interaction.

5.4.2 Human Supervisory Control for Telesystems

The term “human supervisory control” generally refers to the column in [figure 5.3](#) consisting of the Manual and Semi-Autonomous Control quadrants. The Semi-Autonomous Control quadrant can be subdivided into Traded or Shared Control and Autonomy as shown in [figure 5.4](#), reflecting flavors of autonomy. Manual, Traded, and Shared Control are sometimes referred to as *human-in-the-loop control*, while autonomy is referred to as *human-on-the-loop* to differentiate the flavor of human engagement. The connotation is that supervisory control covers the range of interactive control styles from manual out-of-sight control through full autonomy.



[Figure 5.4](#) Human supervisory control matrix expanded for telesystems.

In teleoperation, human supervisory control is normally associated with Manual, Traded, or Shared Control, while the term “autonomy” is associated with taskable agents. The human supervisory control, manual control, traded control, and shared control are often used interchangeably, adding to the confusion over definitions. A robot mission can have portions that are manual, traded, shared, or autonomous so the type of human supervisory control may dynamically change. Another confusing aspect is that some designers have tried to arrange manual, traded, shared, and full autonomy in a hierarchy of increasing artificial intelligence. Traded, shared control, and full autonomy require different intelligent capabilities, and the complexity may depend on the application, so it is difficult to say that one scheme inherently requires more intelligence than another. Shared control of a swarm of a hundred ground, aerial, and marine robots on the battlefield may require more intelligence for both the remote robots and for the local to assist the human in comprehending the situation than in a fully autonomous robot vacuum cleaner.

5.4.3 Manual Control

As shown in [figure 5.5](#), the robot has no onboard control component in Manual Control. There may be an inner control loop to stabilize the flight of a UAV but nothing that would be considered artificial

intelligence. Robots built for teleoperation will have exteroceptive sensing and may or may not have proprioception but will almost always rely on the human to infer exoprorioception.

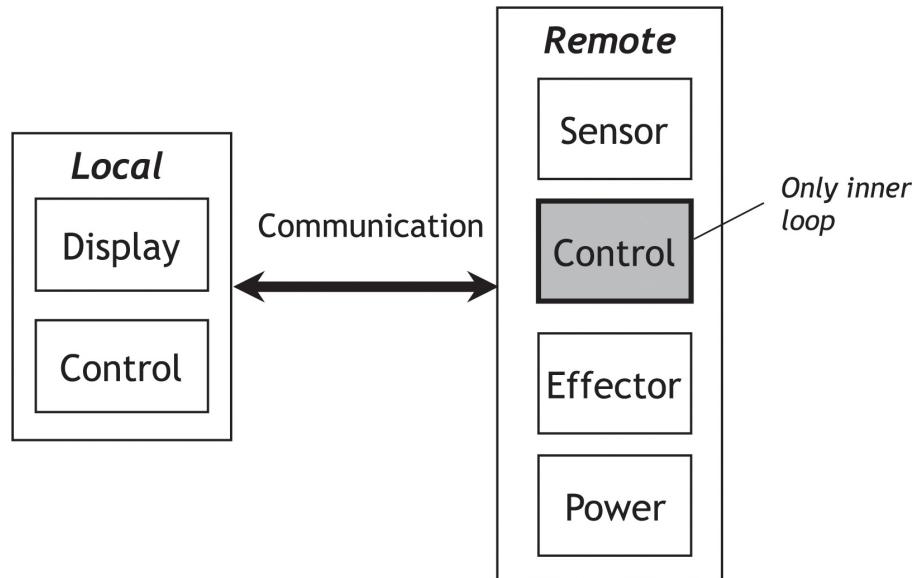


Figure 5.5 Manual control in teleoperation, showing that the remote may have some onboard control.

The teleoperator cannot see the robot and is totally responsible for constructing and maintaining an understanding of what is going on with the robot and the world. The teleoperator must create the understanding based on the Display, or user interface, component in the telesystem. Unfortunately, the robot interfaces are usually insufficient to allow the teleoperator to be aware of where the robot is because the robot hardware rarely incorporates exoprorioceptive sensing mechanism.

These types of perception are defined as follows:

PROPRIOCEPTION

- *Proprioception*: measurements of the movements relative to an internal frame of reference. In biology, this is usually some measure of how far an arm or leg has been extended or retracted. In robotics, actuators are generally motors. Many motors come with a *shaft encoder*, which measures the number of turns the motor has made. If the gearing and wheel size are known, then the number of turns of the motor can be used to compute the number of turns of the robot's wheels, and that number can be used to estimate how far the joint or robot has traveled.

EXTEROCEPTION

- *Exteroception*: measurements of the layout of the environment and object relative to the robot's frame of reference. Exteroception can prevent a robot from falling down stairs, such as the Roomba, which uses a type of range sensor to detect voids and a bump sensor to detect obstacles. Sensors, such as the Kinect originally used for videogames, can provide a 3D reconstruction of the environment.

EXPROPRIOCEPTION

- *Exproprioception:* measurement of the position of the robot body or parts relative to the layout of the environment. This is essentially an external view of the robot. For example if you are hiking and get your foot stuck in a pile of rocks, you can use your physical flexibility to bend down and turn your head to see how the foot is trapped. Proprioception tells you that your foot is stuck at an uncomfortable angle, exteroception tells you that rocks are pressing against your foot, but exproprioception helps you plan to straighten your foot, pull backward, then up in order to extricate it.

[Figure 5.6](#) gives an example of a typical military bomb squad with proprioception and exteroceptive sensing plus some exproprioceptive support for the teleoperator. Data from proprioceptive shaft encoders are used to display an icon of the approximate position of effectors in a robot. The display shows the data from the exteroceptive sensor (the cameras) but the human is expected to mentally convert the data into distances to obstacles, clearances, and so on. The accelerometer proprioceptive data hints at exproprioception as the icon shows that the robot is tilted, implying that the terrain is tilted. The robot comes with several cameras, and the teleoperator is expected to be able to discover a camera or camera/arm combination that will give any needed exproprioceptive viewpoints; as will be discussed in section 5.5, this viewpoint may not be possible to obtain even for “routine” terrain, such as going down stairs.



[Figure 5.6](#) Local display from an iRobot Packbot showing perception.

5.4.4 Traded Control

In traded control, the phases of a mission are divided into discrete tasks that the teleoperator will do and those that the telefactor will do, as shown in [figure 5.7](#). For example, the teleoperator delegates a phase of the task to the robot, the robot executes that phase, and then the human takes over. This is

common in tasks which require dexterous manipulation as the robot can move the arm and manipulator close to the object and then the human can take over and perform the fine movements. Traded control is also used for waypoint navigation to a goal location, where the robot navigates to the goal and then the human takes over to perform a search or inspection based on what is at the location. Trading can occur in any order. Consider that UAVs may be manually flown during search and rescue missions and then put on autonomous return to home when the teleoperator has seen enough.

***Teleoperator and Telefactor
can trade places on who is
in control for a task***

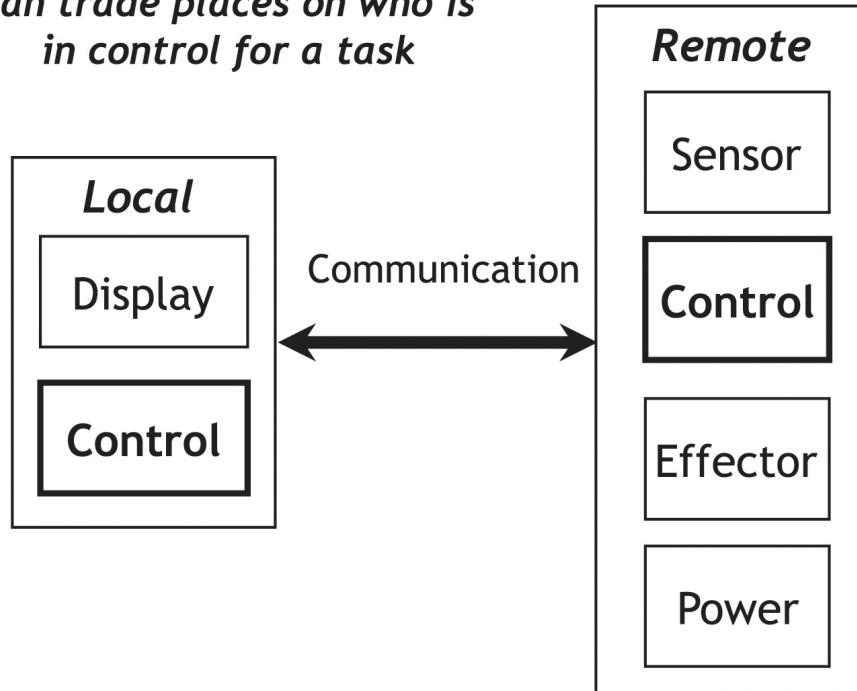


Figure 5.7 Traded control between a teleoperator and a telefactor.

In practice, traded control is implicitly part of almost every fully autonomous system for exception handling and emergency operations, where the robot trades the control to the human. One implicit use of traded control is due to the expectation that, if something goes wrong, the robot will “fail upwards” to the “smarter” agent and thus give control to the human. For example, autopilots in planes will disengage if the situation exceeds their parameters and the human pilot must take over. Another example is that many UAVs have a lost-link procedure where the UAV will autonomously return to home if communication with the operator is lost; here control has been traded to the robot with the human’s implicit permission. As will be covered later in section 5.5.4, emergency or exception-driven traded control creates significant human factors problems.

5.4.5 Shared Control

In shared control, both the teleoperator and the telefactor contribute simultaneously to the control. Typically, the teleoperator provides deliberative inputs and the telefactor provides reactive control, essentially splitting responsibilities along the layers of the canonical software organizational architecture in chapter 4. Antilock brakes can be considered a form of shared control where the car can react to a skid faster than a human can and can execute the correct response of pumping the brakes more reliably.

Teleoperator and Telefactor shared control for a task

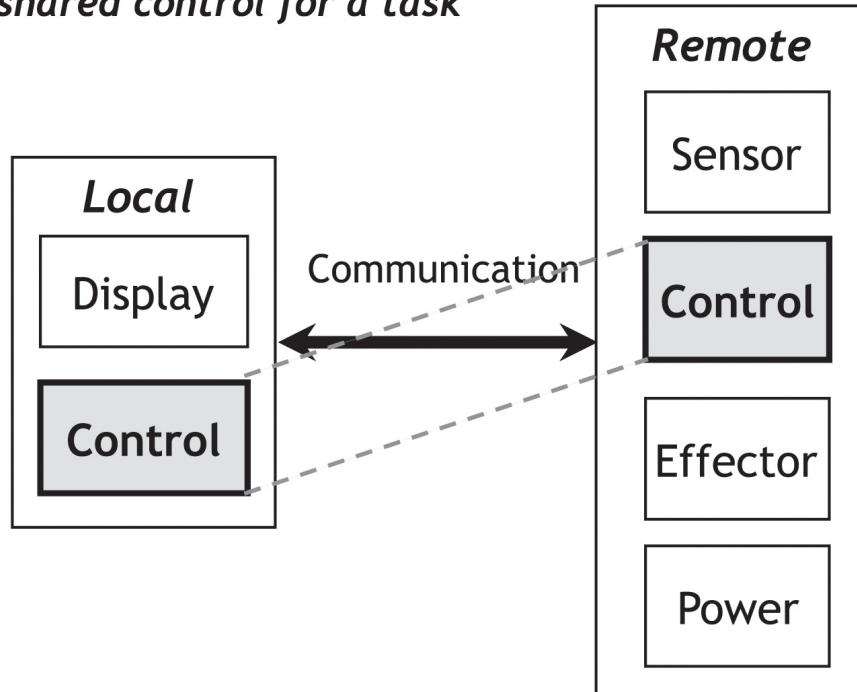


Figure 5.8 Shared control between a teleoperator and a telefactor.

5.4.6 Guarded Motion

GUARDED MOTION

An important form of control that has been used since 1980 is *guarded motion*. Guarded motion is defined in Pratt and Murphy¹⁷⁰ as a type of human-in-the-loop control, where the robot guards itself from unintended consequences of human directives. Examples include vision systems for trucks, which slow or stop the truck to avoid collisions, surgical robots that prevent excessive force of a scalpel or probe, UAVs that cannot fly out of boundaries, or mobile robots that compensate for unstable configurations that may cause the robot to fall. In most cases, guarded motion is making the robot's action safer. Guarded motion presumes that, compared to the robot, the human has inferior knowledge of the environment and the robot's pose and relation to its environment.¹⁷⁰ The robot, as the situated agent in the environment, can sense more completely or comprehend more quickly, what is happening

and react faster or more appropriately than the remote teleoperator. The autonomous capabilities of guarded motion can either be traded or shared, though guarded motion is often categorized as shared control because the robot is always monitoring for situations to guard against.

Following Pratt and Murphy,¹⁷⁰ guarded motion has five components:

- *Autonomy Intervention Criteria*, which defines when the system is allowed to modify the commands issued by the operator. The criteria may lead to *exceptions* where the system steps in. Emergency stops, boundaries on flight, or off-limit configurations are examples of exceptions. Alternatively, the system may constantly be adjusting the operator's commands *online*. For example, in robot wheelchairs, the system may smooth the course generated by a person with tremors or adjust the course to give more clearance from walls and obstacles.
- *Command Integration Method*, which describes how the system integrates its commands with the user's commands, either as traded control or shared control.
- *Monitored Condition*, which specifies the variables and environmental conditions the system monitors and accounts for. Emergency stops and no-fly boundaries are commonly associated with guarded motion, but it has been applied to other conditions including monitoring a robot for system health and overheating.
- *Interface Modality*, which presents information to the teleoperator. The teleoperator always wants an indication of what the robot is doing. If the robot suddenly does not respond to commands, the teleoperator needs to know whether it is a robot failure, it is stuck, or has it entered an unsafe or forbidden condition. In cars, a light will flash when antilock brakes are activated in case the driver does not hear or feel the distinctive rapid pumping action or understand that the noise and vibration is "normal" and not a defect.
- *Display Preprocessing*, which specifies the preprocessing performed on sensor data before it is presented to the teleoperator. Simplifying the data into readily comprehensible information is important because if something is going wrong, the teleoperator may not have time or can devote attention to go through raw data.

5.5 Human Factors

Given that a telesystem is a type of joint cognitive system, the human element is as important as hardware and software to a successful system. The human element of concern in teleoperation research is about the relationship between the teleoperator and the robot, though in telecommuting there is a social relationship between the robot and the person(s) working with or around the robot in the task environment. The relationship between teleoperator and telefactor is addressed in human factors and human-robot interaction literature, while the social interactions are addressed by human-robot interaction literature covered in chapter 18. The primary topics in human factors for teleoperation are the sources of cognitive fatigue, the impact of latency on the joint cognitive system, how the cognitive limitations translate into the necessary manpower to safely operate, and how to prevent the human out-of-the-loop control problem.

5.5.1 Cognitive Fatigue

Teleoperation is not an ideal solution for all situations. Many tasks are inherently repetitive and boring. For example, consider using a joystick to drive a radio controlled car; after a few hours, it tends to get harder and harder for humans to pay attention. If the human is manually controlling the robot, the task is made even harder by loss of *depth perception*; the human brain has to work harder to compensate for the lost information that would have been obtained from eye saccades, head movements, audio cues, and so forth. Likewise, the human brain has to work harder to translate human ways of doing things into robot ways of doing things; the effort to perform actions at a distance has been dubbed the *telekinesis problem*.¹⁹¹ As a result of the demands of a task, the loss of depth perception, and the telekinesis problem, most people quickly experience *cognitive fatigue*; their attention wanders and they may even experience headaches and other physical symptoms of stress.

KEYHOLE EFFECT

Cognitive limitations and fatigue can be introduced by the display and sensor components. Imagine trying to control a radio controlled car while looking through a small camera mounted in front. The task becomes much harder because of the limited field of view; essentially there is no peripheral vision. This is known as the *keyhole effect*,²²¹ where a teleoperator is trying to understand the environment by looking through a keyhole in a door or a hole in the wall. One common, though misguided, approach to the keyhole effect is to place more cameras on the robot. This can actually increase the cognitive load and introduce errors because the operator now has to choose between viewpoints and mentally fuse different viewpoints.

SIMULATOR SICKNESS

More advanced displays try either to create simulations of the larger environment or introduce camera effects similar to the slight distortions of peripheral vision²¹⁴ in order to reduce cognitive load. But even if the visual display is excellent, the teleoperator may get *simulator sickness* due to the discordance between the visual system indicating the operator is moving and the inner ear indicating the operator is stationary.²¹¹

Latency in the communications component can also introduce cognitive fatigue. It is tiring to have to wait to see how the robot has executed a command before giving another command. The slow pace is frustrating and also makes it hard to pay attention. Latency is discussed separately in the next subsection because it has other impacts on the success of the telesystem.

5.5.2 Latency

TELEOPERATION TIME HEURISTIC

One problem with teleoperation for space applications, telemedicine, or applications over long distances is that it can be inefficient or dangerous because of *large time delays*.¹⁹² A large time delay can result in the teleoperator giving a remote a command, unaware that it will place the remote in jeopardy or that an unanticipated event, such as a rock fall, might occur and destroy the robot before the teleoperator can see the event and command the robot to flee. A rule of thumb, or *heuristic*, is that the time it takes to do a task with traditional teleoperation grows linearly with the transmission delay. A teleoperation task which took 1 minute for a teleoperator to guide a remote to do on the Earth might

take 2.5 minutes to do on the Moon, and 140 minutes on Mars.⁶¹

Latency effects are less prevalent on terrestrial applications, but still occur. For example, telemedicine from a hospital in one country to a field hospital in another country would have to contend with network delays. Even in non-life-and-death situations, such as telecommuting, it can be frustrating to control devices over the Internet where the bandwidth may change unpredictably.

5.5.3 Human: Robot Ratio

A practical drawback to teleoperation is that it often requires more than one person to operate the robot due to the cognitive limitations of a single teleoperator. Studies of teleoperation of ground robots and aerial vehicles deployed for disasters show that it is more effective and safer to have one person drive and one person look through the sensors, even if there is a single camera for both navigation and mission sensing. Work by Burke and Murphy showed that responders were nine times more likely to find simulated victims in rubble using teleoperated robots if the responders worked cooperatively in teams of two.³⁴ This is similar to having a pilot and copilot in commercial aviation, though it may seem excessive to have more than one person control a robot with only a few effectors and surfaces. However, human factors suggest that this is why more people are needed; because a “dumb,” minimally capable robot may have only a few movements or actions it can perform, determining the right plan under limitations in a complex world can be very difficult.

SAFE HUMAN-ROBOT RATIO

The state of the practice for designers is to start with the baseline *safe human-robot ratio* and then determine if the autonomy and interface support a reduction in manpower. The baseline safe human-robot ratio is as follows:

$$N_h = N_v + N_p + 1$$

where N_h is the number of humans, N_v is the number of vehicles, and N_p is the number of payloads. This equation should be considered as the starting point for designing a human-robot system, but the impact of autonomy, interfaces, attentional demands, sparse environments, and so forth, on reducing cognitive limitations may permit a lower human-robot ratio.

The Predator unmanned aerial vehicle requires at least one teleoperator to fly the vehicle and another teleoperator to command the sensor payload to look at particular areas. Other UAVs have teams composed of as many as four teleoperators plus a fifth team member who specializes in takeoffs and landings. These teleoperators may have over a year of training before they can fly the vehicle. In the case of UAVs, teleoperation permits a dangerous, important task to be completed, but with a high cost in manpower.

The Predator is an example of how the type of human supervisory control may dynamically change over the timeline of a mission, which is loosely categorized as a sequence of initiation, implementation, and termination.¹⁴⁴ In computer science, this sequence would be similar to a program loop consisting of the loop entry conditions, the execution of the steady-state case within the loop, and the loop exit conditions. While the main purpose of the loop is the steady-state case, the entry and exit conditions are often trickier to design than the steady-state case and can lead to major programming errors. Likewise,

initiation and termination can be trickier in robotics. In the case of the Predator, autonomy was applied only to the implementation phase, ignoring the opportunities for onboard intelligence for takeoff and landing, which are also difficult and pose high cognitive workloads. In general, artificial intelligence might be available for autonomous operations or to provide assistance and reduce the workload of mission initiation and termination as well as implementation.

The Predator is also an example of two reasons why more than one operator may be needed to control a single robot. One reason is historically many people are needed to operate a new technology because humans are the ultimate fix to an initial poor design.²²⁰ A second reason is that the people are integers; consider that a specialist in takeoffs and landings may not be used 100% of the mission time, yet has to be available during the mission.

As noted in the 2012 Defense Science Board study on autonomy for unmanned systems,¹⁴⁴ design of military robots often focuses on the hardware, deferring issues of human control, human factors, and autonomy on the assumption that a person can be trained to overcome the limitations of the robot control system. However, people cannot be trained beyond their physiological limits. On the other hand, adding more autonomous capabilities can decrease manpower needs; however, increased automation may actually decrease safety because of the human-out-of-the-loop control problem described in the next subsection.

5.5.4 Human Out-of-the-Loop Control Problem

HUMAN OUT-OF-THE-LOOP (OOTL) CONTROL PROBLEM

Recall that a form of traded control is initiated by the remote robot when it encounters a problem or exception and that, in shared control, the human teleoperator is expected to notice and respond to problems or exceptions. In these cases, teleoperation is the back-up plan for an autonomous capability. However, it is not always possible for the human to react fast enough to serve as a reliable back-up plan. This is the *human out-of-the-loop (OOTL) control problem*, where the human suddenly has to shift modes from traded or shared control to manual control and to control the remote manually under an unmodeled or unexpected circumstance that the automation could not handle. The human out-of-the-loop control problem has been documented since the 1960s.^{99;118;125} The human factors community notes that there are often significant delays in a human assuming control of an autonomous process.⁷¹ In general, the more autonomous the process, the harder it is for the human to quickly and correctly react to a problem because the human does not maintain the necessary situation awareness for control of the process and must build it before reacting. Building situation awareness in order to solve a problem can be difficult because the construction process may require shifting perspectives. The egocentric, environment-based frame of reference held by the Mission Specialist looking through a camera at the world is quite different from the exocentric or possibly mixed exocentric/ egocentric cockpit types of displays needed by a Pilot to look at the state of the robot relative to the environment. The differences in these two views seem difficult to overcome in a few seconds.

An example of the human out-of-the-loop control problem occurred with the DarkStar UAV (shown in figure 5.9) and shows that human factors “failures” generally involve more than one violation of design principles. DarkStar was an expensive experimental unmanned aerial vehicle (UAV) developed in simulation to reduce costs. Advanced prototypes of these vehicles could fly autonomously, but take-

offs and landings were expected to be more difficult for the onboard computer to manage. As a safety precaution, human expert pilots were available to assume teleoperation control of the vehicle should it encounter problems during take-off. Unfortunately, DarkStar did encounter problems during takeoff because the simulation had oversimplified the model of the runway in a classic instance of the closed-world assumption that all attributes of the world could be explicitly enumerated. DarkStar spectacularly crashed, earning the unofficial nickname “Darkspot.”

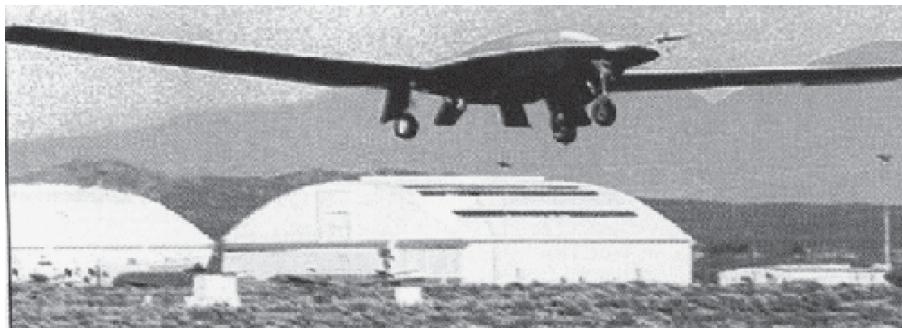


Figure 5.9 DarkStar unmanned aerial vehicle. (Photograph courtesy of DefenseLink Office of the Assistant Secretary of Defense-Public Affairs.)

In the DarkStar system, it was actually impossible for the experts to shift in time from out-of-the-loop to in-the-loop control for two reasons. First, the display prevented the pilots from adequately defining the problem and thus generating an appropriate response. The display had been developed for post-hoc engineering diagnostics, not for real-time flight control, so it was essentially a stream of numbers, the equivalent of building a telesystem without the local display component. The engineering display did show the pilot that something was going wrong with the takeoff, but the pilot did not have enough time to mentally fuse the raw data into a hypothesis of the problem and generate a correction for the problem. The design of the shared control backup plan had failed to consider the cognitive limitations of the human. Second, the contingency plan did not factor in the seven second delay introduced by using a satellite as the communications link. The latency meant that, even if the pilot had comprehended the cause of the flight deviation and generated a correct control response, the remote could not have gotten the new commands fast enough to prevent the crash.

The use of robots at the Fukushima Daiichi nuclear accident illustrated that teleoperation can be a deliberate choice for avoiding human out-of-the-loop control problems. A Honeywell T-Hawk® unmanned aerial system was used to conduct radiological and general assessment surveys starting four weeks after the accident. The T-Hawk flew approximately 40 missions up to 1,600 meters from the launch point into the extreme radiation zone. The T-Hawk had fully autonomous navigation capabilities that would allow it fly a preprogrammed route and return home if any problems were encountered. Yet the expert pilots turned off the autonomous navigation capabilities and manually flew the route. The concern was that, if the robot started to deviate from the expected path, the pilots would have no way of knowing whether the deviation was due to radiation damage or winds. The pilots preferred to stay directly engaged with the system rather than risk having the autonomy functions encounter an unmodeled condition and react unpredictably. In human factors terminology, the pilots were worried

about a human out-of-the-loop control situation arising. The way they prevented that situation from happening was to be in-the-loop.

5.6 Guidelines for Determining if a Telesystem Is Suitable for an Application

TASK CHARACTERISTICS

According to *The Concise Encyclopedia of Robotics*,⁶¹ teleoperation is best suited for applications where:

1. *The tasks are unstructured and not repetitive.* Otherwise, the designer should look at adding autonomy.
2. *The task workspace cannot be engineered to permit the use of industrial manipulators.* Workspaces, such as nuclear power plants, manufacturing facilities, and hospitals, can be engineered to support specific tasks and standard robotic equipment. It may not be possible to tailor other environments, such as other planets, outdoors, or disasters, to the safety of workers.
3. *Key portions of the task intermittently require dexterous manipulation, especially hand-eye coordination.* Most robots do not have sufficient sensing and planning (though that is changing) to do certain tasks. Having the teleoperator perform a task requiring dexterity is not a trivial design issue, and the display and control components need to support the teleoperator's cognitive work.
4. *Key portions of the task require object recognition, situational awareness, or other advanced perception.* As described in chapter 3, the symbol grounding problem remains a fundamental barrier in artificial intelligence. Thus, tasks which require this function may be beyond current capabilities.

TELEPRESENCE

VIRTUAL REALITY

5. *The needs of the display technology do not exceed the limitations of the communication link (bandwidth, time delays).* Display and communication issues have been a major limitation of telepresence. Telepresence aims for what is popularly called *virtual reality*, where operators have complete sensor feedback and feel as if they are the robot. The amount of data about the environment needed to effectively recreate the situation for the teleoperator typically exceeds the communications component's capabilities.
6. *The availability of trained personnel is not an issue.* In some situations, such as space exploration, the telefactor can be supported by dozens or hundreds of technicians and scientists. In others, such as disaster response, there are few experts available.

5.6.1 Examples of Telesystems

Teleoperation is a popular solution for controlling remotes because the capabilities of AI technology are nowhere near human levels of competence, especially in terms of perception and decision-making. One example of teleoperation is the exploration of underwater sites, such as the Titanic. Having a human control a robot is advantageous because a human can isolate an object of interest, even if it is

partially obscured by mud in murky water.²¹¹ Humans can also perform dexterous manipulation (e.g., screwing a nut on a bolt), which is very difficult to program a manipulator to do.

Another example is the Sojourner robot shown in [figure 2.7](#), which explored Mars from July 5 to September 27, 1997, until it ceased to reply to radio commands. Since there was little data on what Mars was like before Sojourner arrived, it was hard to develop sensors and algorithms which could detect important attributes or even design control algorithms to move the robot. It is important that any unusual rocks or rock formations (like the orange rock Dr. Schmitt found on the Moon during Apollo 17) be detected. Humans are particularly adept at perception, especially at seeing patterns and anomalies in pictures. AI perceptual abilities at that time fell far short of human abilities. Humans are also adept at problem-solving. Recall from earlier in the book, that the air bags for the Mars Pathfinder craft did not deflate properly and ground control sent commands to retract the petals and open them again. That type of problem solving is extremely difficult given the current capabilities of AI.

The use of ground robots during the 2011 Fukushima Daiichi nuclear accident provides an additional example of why teleoperation is useful. Teleoperated robots were used successfully to assess the damage to the reactor buildings, remove debris from roadways and access points, survey the extent and magnitude of the radiation, and pump water to cool the reactors.¹⁴⁷ The use of robots highlighted the cognitive limitations imposed by the sensors on the remote and on the display at the local. Two robots were needed to allow a single robot to perform a task (see [figure 5.10](#)). Outside the reactor buildings, small Bobcat® construction robots were adapted for teleoperation by QinetiQ to clear debris and bury radioactive materials in order to let crews and larger equipment have access to the buildings. A small, agile bomb squad robot was paired with a construction robot. The bomb squad robot could move around the Bobcat in order to let the teleoperator see if the front bucket for moving dirt and debris was touching the ground. Inside the reactor buildings, iRobot Packbots® were used in pairs to get through debris and to perform manipulator tasks such as opening doors. One robot performed the task but the second robot served as a visual assistant.



Figure 5.10 A teleoperated Bobcat and Talon robots working together to enable the Bobcat to clear debris at Fukushima. (Photograph courtesy of QinetiQ.)

5.7 Summary

This chapter addresses teleoperation and telesystems, which historically were the first attempts at designing intelligent robots. A telesystem is a human-machine system where the human, or teleoperator, at a local display uses a telecommunication link to control a remote robot that is out of sight. The remote robot, or telefactor, provides sensing, mobility, effectors, and power but may also have onboard control as well.

Teleoperation arose as a temporary solution, or hack, for tasks that required robots, but robots could not be adequately programmed to act correctly on their own. This problem gave rise to a debate questioning if teleoperation is a “temporary evil” on the path to autonomy or if it is a different style of AI. Now teleoperation is recognized as a different style of AI, one that assists humans in obtaining *remote presence* into a distal environment. Teleoperation is no longer considered a “temporary evil” on the path to true autonomy but rather one facet of human supervisory control. Understanding teleoperation is critical because it is the likely mode of control when artificial intelligence for a *taskable agent* fails.

Teleoperation, as a facet of human supervisory control, raises the question *What is human supervisory control?* Human supervisory control refers to a spectrum of ways in which humans and machines can interact to accomplish tasks from manual control of a remote robot to full delegation of the entire task to the robot. The spectrum has produced a variety of terms often grouped under the label of *semi-autonomy*. This, in turn, raises the questions *What is semi-autonomy?* and *How are the four modes of semi-autonomy different?* Semi-autonomy refers to human-machine control modes where the human cannot directly observe the robot in the task environment, and portions of the task are directed by the robot, not the human. In *traded control*, the task is split into parts that either the human or the robot is good at. When the task reaches the point at which the expertise of one agent is exceeded, the dominant control shifts or trades to the other agent. In *shared control*, both the human and robot are working simultaneously on aspects of the task.

Guarded motion is perhaps the most prevalent application of shared control. In guarded motion, the robot agent actually overrules or modifies the human agent’s control commands. Guarded motion is an excellent example of how robots are physically situated agents and thus may comprehend better, and certainly faster, than a human who is not located in the environment. Two major misconceptions about the human supervisory control spectrum are that 1) it represents hierarchy of intelligence, when it is simply a labelling convention, and 2) each task has a single supervisory control mode when most implementations show that the control mode dynamically changes with the phases of the mission.

Since teleoperation is a different style of AI, it becomes essential to understand what task domains telesystems are useful for. There are six guidelines for identifying a task domain where using a telesystem is likely to be successful. Otherwise, the solution may be to engineer the environment to accommodate robots or to rethink the mission. Applications, such as Mars planetary exploration, present task environments that cannot be engineered to allow unintelligent robots. Terrestrial applications, such as using ground robots at the Fukushima Daiichi nuclear accident to enter the plant, survey, move debris, and insert sensors, show that teleoperation is needed when the tasks are unstructured, new, or require dexterous manipulation.

Teleoperation methods typically are cognitively fatiguing, require high communication bandwidths

and short communication latency, and require one or more teleoperators per remote as directed by the *safe human:robot ratio* formula. Telepresence techniques attempt to create a more natural interface for the human to control the robot and interpret what it is doing and seeing but at a high communication cost. Reduced cognitive fatigue and the general robustness of the telesystem depend on the availability of *proprioception*, *exteroception*, and, the often overlooked, *exproprioception*. Teleoperation is the default back-up plan for a taskable agent, but it may not be cognitively possible for a human to take over and recover from an anomaly, which is the *human out-of-the-loop* problem.

Finally, this chapter should reinforce the notion that the key to creating artificially intelligent robots is more than just adding sophisticated algorithms. It involves careful consideration of the role of the human either “in-the-loop,” “on-the-loop,” or “out-of-the-loop.” AI robotics also involves understanding the ecology of the task environment and how the robot’s capabilities must fit into that ecology; this ecology is the heart of reactive robotics, which is the focus of the next five chapters in part II.

5.8 Exercises

Exercise 5.1

List three problems with teleoperation.

Exercise 5.2

Describe the components and the responsibilities of the local and the remote members of a telesystem.

Exercise 5.3

Consider exoskeletons, such as the power lifter seen in the movie *Aliens* and robots such as the AT-AT Walker in movies like *Star Wars* and *Pacific Rim*, where the operator is inside the robot. Would these be considered teleoperated? Why or why not?

Exercise 5.4

List the characteristics of applications that are well suited for teleoperation.

Exercise 5.5

Give at least two examples of potentially good applications for teleoperation not covered in the chapter. In thinking about applications, consider whether the robot is being used to replace a person, allowing a person to project themselves into a distant workspace, assisting a person, or amusing a person.

Exercise 5.6

Compare and contrast *taskable agency* and *remote presence*.

Exercise 5.7

Define guarded motion and explain its relationship to traded and shared control.

Exercise 5.8

List and describe the five components of guarded motion.

Exercise 5.9

What is the human out-of-the-loop control problem?

Exercise 5.10

Why should teleoperation always be included in a robot system? What happens if there is a strong possibility of human

out-of-the-loop control errors?

Exercise 5.11

In 1994, an autonomous car (with a person in the driver's seat), developed at Carnegie Mellon University, drove a total of 2897 miles, from Pittsburgh to San Diego, primarily along interstate highways in an event called "No Hands Across America." However, an additional 48 miles were driven by a human when the car needed to exit the interstate to get gas or the passengers needed food or lodging. Is it reasonable to classify this trip as an example of as autonomous control? Fully autonomous control? Was the trip, or portions of the trip, shared control? Was the trip, or portions of the trip, traded control?

Exercise 5.12

Why does not adding more cameras necessarily reduce the cognitive limitations from the keyhole effect?

Exercise 5.13

Consider proposals to launch rovers to the moon and allow private citizens to teleoperate them, in effect, creating telesystem tourism. Consider that one motivation for creating such a system is that people want to drive and look directly at the flags posted on the moon from the Apollo missions and see the astronauts footsteps. However, this scenario poses the risk that someone will drive over the footprints, hit the flag, or otherwise damage or destroy the site. Is this a real possibility? Is there anything inherent in telesystems that could be applied to prevent this?

[World Wide Web]

Exercise 5.14

Search the World Wide Web for companies that provide telecommuting robots. Compare and contrast the robots in terms of how natural it would be for you, as the teleoperator, to operate the robot and interact with the remote workspace and how natural it would be for the people in the remote workspace to work with the robot. Does the physical shape, height, mobile base, and so forth present any problems in either mobility or in interacting with people? Which robots do you think are better in each of the four categories of human supervisory control?

[Programming]

Exercise 5.15

(This requires access to a robot with an on-board video camera and a teleoperation interface.) Teleoperate the robot through a slalom course of obstacles while keeping the robot in view as if controlling a remotely controlled car. Now, looking only at the output of the video camera, repeat the obstacle course. Repeat the comparison several times, and keep track of the time it takes to complete the course and the number of collisions with obstacles. Which viewpoint leads to faster completion of the course? Fewer collisions? Why?

[Science Fiction]

Exercise 5.16

Read the short story "Stranger in Paradise," by Isaac Asimov and enumerate the problems with telepresence illustrated in this story.

[Science Fiction]

Exercise 5.17

Watch the 1971 movie, *The Andromeda Strain*, based on the novel by Michael Crichton. The movie has several nerve-wracking scenes as the scientists try to telemanipulate an unknown, deadly organism as fast as possible from one container to another without dropping the sample. Do you think this type of telemanipulation could be accomplished faster and more reliably with today's robots? Compare the telemanipulation tasks in *The Andromeda Strain* to the DARPA Robotics Challenge.

5.9 End Notes

Real teleoperated robot in a movie.

In the 1994 movie *Stargate*, the military uses a teleoperated vehicle to be the first to go through the stargate and test the environmental conditions on the other side. In reality, the movie used the NASA Jet Propulsion Laboratory's Hazbot.

II

Reactive Functionality

Contents:

- [Chapter 6: Behaviors](#)
- [Chapter 7: Perception and Behaviors](#)
- [Chapter 8: Behavioral Coordination](#)
- [Chapter 9: Locomotion](#)
- [Chapter 10: Sensors and Sensing](#)
- [Chapter 11: Range Sensing](#)

6

Behaviors

Chapter Objectives:

- Describe the three levels in a computational theory.
- Give the mathematical definition of *behavior*.
- Explain in one or two sentences each of the three categories of behavior (*reflexive, reactive, conscious*).
- Be able to classify a behavior either as a *reflex, taxis, or fixed-action pattern*.
- If given a graphical representation of a behavior, identify the behavioral, perceptual, and motor schemas and translate them into *S-R schema notation*.
- Express a set of behaviors in S-R schema notation.

6.1 Overview

Chapter 4 introduced the canonical operational architecture with three layers, representing different styles of intelligence. The most basic layer was the Reactive Layer, composed of animal-like behaviors. The analogy of reactive robot intelligence and animal intelligence raises two questions that will be addressed in this chapter. The first, *Is the reactive layer really like ethology?* This question can also be rephrased as: how can a robot be like an animal, and is there any real, concrete value in thinking of a robot as an animal? The second, more blunt, question is *If so, how do you get that crunchy-chewy animal documentary stuff into a computational format?* Analogies can be interesting from a theoretical perspective, but programming a robot requires concrete mechanisms. What would be helpful are specific mechanisms for duplicating animal intelligence with computational constructs.

The chapter will answer these questions by first providing the historical motivation for artificial intelligence researchers to explore animal behaviors. Next, the chapter will introduce Marr's Computational Theory, which was designed to provide a bridge between biological and silicon-based intelligence and jumpstart the process of transferring biological principles into programming constructs.

Marr's computational theory has three levels loosely corresponding to the operational (big picture),

systems (needed transformations), and technical (actual implementation) architectures. With the framework in place for thinking computationally about animal behaviors, the chapter next presents the types of animal behaviors, emphasizing the Level 1 computational theory aspects.

Animal behaviors can be modeled as highly modular, generic, independent processes that are triggered by perceptual or internal events or computational states. Behaviors can be expressed using schema theory, which converts inputs, outputs, and transformations into stimuli, responses, and behaviors. Schemas become tangible programming modules with S-R notation, where schemas are identical to objects in object-oriented programming.

This chapter is the first of three chapters on behaviors. Reflexive behaviors, covered in the next chapter, do not require world models or long term memory, making them extremely computationally efficient. However, the price of high modularity and efficiency is algorithmic non-determinism; the overall behavior of the robot emerges rather than is linearly programmed. Chapter 8 will cover the different coordination algorithms by which the individual, independent behaviors produce the overall behavior.

6.2 Motivation for Exploring Animal Behaviors

Progress in robotics in the 1970s was slow. The most influential robot was the Stanford Cart, developed by Hans Moravec, which used stereo vision to see and avoid obstacles protruding from the ground. In the late 1970s and early 80s, Michael Arbib began to investigate models of animal intelligence from the point of view of the biological and cognitive sciences in the hope of gaining insight into what was missing in robotics. While many roboticists had been fascinated by animals, and many, including Moravec, had used some artifact of animal behavior for motivation, no one approached the field with the seriousness and dedication of Arbib.

At nearly the same time, a slender volume by Valentino Braitenberg, called *Vehicles: Experiments in Synthetic Psychology*,²⁷ appeared. It was a series of gedanken or conceptual thought experiments, speculating as to how machine intelligence could evolve. Braitenberg started with Vehicle 1, which had a simple thermal sensor-motor actuator pair. The vehicle could move faster in warm areas and slower in cold areas. The next more complex vehicle had two thermal sensor-motor pairs, one on each side of the vehicle, mimicking a mutation. As a result of the differential drive effect whereby the heat on one side would cause that motor to rotate faster than the motor on the cooler side, Vehicle 2 could turn around to go back to cold areas. Throughout Braitenberg's book, each vehicle added more complexity. This layering was intuitive and seemed to mimic the principles of evolution in primates. *Vehicles* became a cult classic among roboticists, especially in Europe.

Soon a new generation of AI researchers answered the siren's call to investigate the lessons that could be learned from biological intelligence. They began exploring the biological sciences in search of new organizing principles and methods of producing intelligence. As will be seen in the next chapter, these efforts would lead to the Reactive Paradigm. This chapter attempts to set the stage for understanding the Reactive Paradigm by recapping influential studies and discoveries and attempting to cast them in light of how they can contribute to robotic intelligence.

Why should roboticists explore biology, ethology, cognitive psychology, and other biological sciences? There is a tendency for people to argue against considering biological intelligence when

designing robots using the analogy that airplanes do not flap their wings. The counter-argument is that almost everything else about a plane's aerodynamics duplicates a bird's wing. Almost all the movable surfaces on the wing of a plane perform the same functions as parts of a bird's wing. The advances in aeronautics came as the Wright Brothers and others captured aerodynamic principles from observing the flight of birds. Once the principles of flight were established, mechanical systems could be designed which adhered to these principles and performed the same functions but not necessarily in the same way as biological systems. The "planes do not flap their wings" argument turns out to be even less convincing for computer systems. Animals make use of innate capabilities; robots rely on compiled programs.

AI roboticists often turn to the biological sciences for a variety of reasons. One is that animals and humans provide existence proofs of different aspects of intelligence. It often helps a researcher to know that an animal can do a particular task, even if it is not known how the animal does it, because the researcher at least knows it is possible. For example, the issue of how to combine information from multiple sensors (sensor fusion) has been an open question for years. At one point, papers were being published discouraging roboticists from researching sensor fusion, on the grounds that sensor fusion was a phenomenon that sounded reasonable but had no basis in fact. Additional research showed that animals (including humans) do perform sensor fusion, although with surprisingly different mechanisms than most researchers had considered.

A second reason for exploring biological science is that animals live in an *open world*, and roboticists wanted to overcome the *closed-world assumption* that presented so many problems with Shakey. Many "simple" animals, such as insects, fish, and frogs, exhibit intelligent behavior, yet have little or no brain. Therefore, they must be doing something that avoids the *frame problem*.

6.3 Agency and Marr's Computational Theory

COMPUTATIONAL THEORY

Even though it seems reasonable to explore biological and cognitive sciences for insights into intelligence, how can we compare such different systems: carbon and silicon "life" forms? One powerful means of conceptualizing the different systems is to think of an abstract intelligent system as an agent. In object-oriented programming terms, the "agent" is the superclass, and the classes of "person," "animal," and "robot" agents are derived from it. Of course, just referring to animals, robots, and intelligent software packages as "agents" does not make the commonalities and correspondences between intelligences any clearer. One helpful way of seeing correspondences is to decide on the level at which these entities have something in common. The set of levels of commonality leads to, what is often called a *computational theory*,¹²⁰ as defined by David Marr. Marr was a neurophysiologist who tried to recast biological vision processes into new techniques for computer vision. The levels in a computational theory can be greatly simplified as:

Level 1: Existence proof of what can/should be done. For example, suppose a roboticist is interested in building a robot to search for survivors trapped in a building after an earthquake. The roboticist might consider animals that seek out humans. As anyone who has been camping knows, mosquitoes are very good at finding people. Mosquitoes provide an existence proof that it is possible for a

computationally simple agent to find a human being using heat and carbon dioxide. At Level 1, agents can share a commonality of purpose or functionality.

Level 2: Decomposition of “what” into inputs, outputs, and transformations. This level can be thought of as creating a flow chart of “black boxes” to actually accomplish the “what” from Level 1. Each box represents a transformation of an input into an output. Returning to the example of a mosquito, the roboticist might realize from biology that the mosquito finds humans by homing in on the heat of a human (or any warm blooded animal). If the mosquito senses a hot area, it flies toward it. The roboticist can model this process as: `input=thermal image, output=steering command`. The “black box” is a placeholder for the mechanism that the mosquito uses to transform the input into the output. A mosquito might be taking the centroid of the thermal image (the centroid weighted by the heat in each area of the image) and steering to that area. If the hot patch moves, the thermal image will change with the next sensory update, and a new steering command will be generated. This mechanism might not be exactly how the mosquito actually generates steering commands, but it suggests how a robot could duplicate the functionality. Also notice that by focusing on the process rather than on the implementation, a roboticist does not have to consider that mosquitoes fly because a search and rescue robot might have wheels instead of wings. At Level 2, agents can exhibit common processes.

Level 3: How to implement the process. This level of the computational theory focuses on describing how each transformation, or black box, is implemented. For example, in a mosquito, the steering commands might be implemented with a special type of neural network, while in a robot, the implementation might involve an algorithm which computes the angle between the centroid of heat and where the robot is currently pointing. Likewise, a researcher interested in thermal sensing might examine the mosquito to see how it is able to detect temperature differences in such a small insect; electro-mechanical thermal sensors weigh close to a pound! At Level 3, agents may have little or no commonality in the actual physical implementation of the functionality specified in Level 1.

[Figure 6.1](#) illustrates the general idea of a computational theory. The top, most abstract, level might correspond to a phenomenon that people can do, such as recognizing a house in a picture or a house from a cartoon of lines. The second level refines the description of the phenomenon into a series of transformations that would describe what sensor data enters the brain, and what regions of the brain are activated and produce outputs leading to the ultimate output of recognizing a house. The regions of the brain correspond to modules or algorithms in a computer program. The lowest level describes the activity of each region of the brain in terms of the specific types of neurons and how they are connected. This level of activity in a robot might be implemented on a dedicated computer chip.

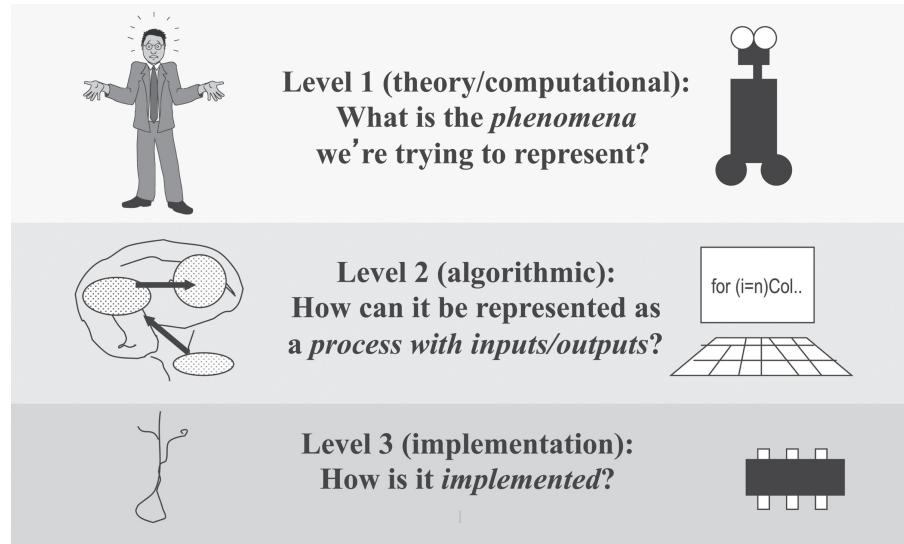


Figure 6.1 Marr's Computational Theory.

It should be clear that Levels 1 and 2 are abstract enough to apply to any agent. It is only at Level 3 that the differences between a robotic agent and a biological agent really emerge. Some roboticists actively attempt to emulate biology by reproducing the physiology and neural mechanisms. (Most roboticists are familiar with biology and ethology but do not try to make exact duplicates of nature.) Robert Full's work in analyzing how geckos adhere to walls as they climb and how cockroach legs work has made great breakthroughs in robot locomotion.^{15,7} Figure 6.2 shows work at Case Western Reserve's Bio-Bot Laboratory, under the direction of Roger Quinn, reproducing a cockroach on a neural level.

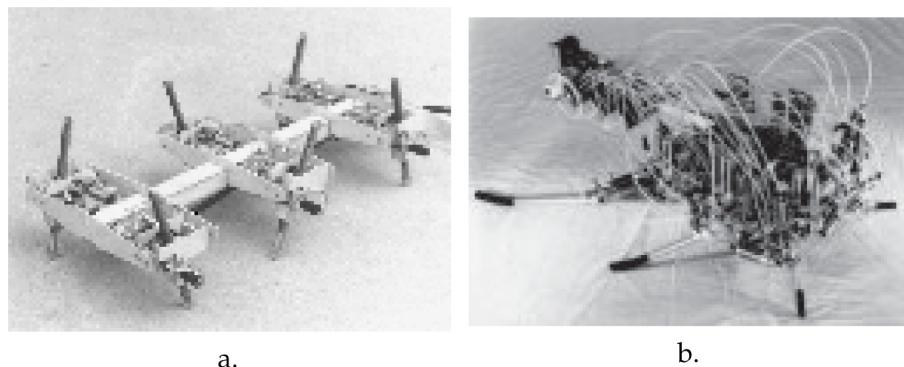


Figure 6.2 Robots built at the Bio-Bot Laboratory at Case Western Reserve University imitate cockroaches at Level 3: a.) Robot I, an earlier version, and b.) Robot III. (Photographs courtesy of Roger Quinn.)

In general, it may not be possible, or even desirable, to duplicate the way a biological agent does something. Most roboticists do not strive to replicate animal intelligence precisely, even though many

build creatures that resemble animals, as seen by the insect-like Genghis in [figure 6.3](#) or humanoid robots. But by focusing on Level 2 of a computational theory of intelligence, roboticists can gain insights into how to organize intelligence.

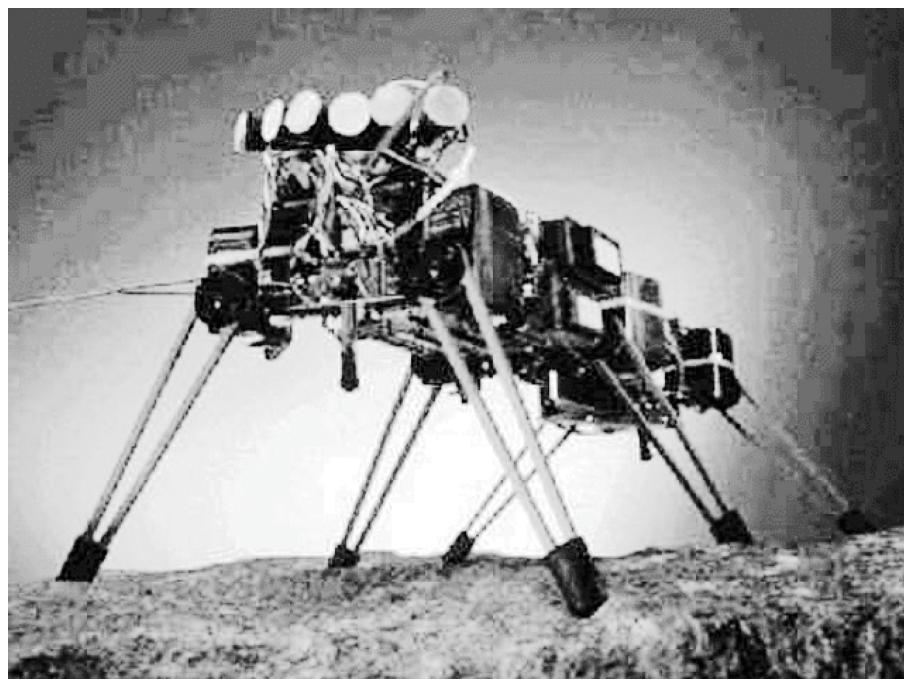


Figure 6.3 Genghis, a legged robot built by Colin Angle, IS Robotics, which imitates an insect at Levels 1 and 2. (Photograph courtesy of the National Aeronautics and Space Administration.)

6.4 Example of Computational Theory: *Rana Computatrix*

Rana computatrix is the earliest example of a computational theory being applied to AI robotics. It demonstrates the value of a computational theory. It is a concrete example of how one researcher translated the biological organization into computational objects. Having a concrete example is helpful in understanding the computational process because different researchers may interpret the biological organization of a phenomenon differently and thus propose different theories. *Rana computatrix* is also interesting because Arbib used a particular computation structure, called schemas, to mathematically express the relationships between those objects. Schemas will be the basic programming structure for the Reactive Layer in intelligent robots and will formally be presented later in the chapter.

RANA COMPUTATRIX

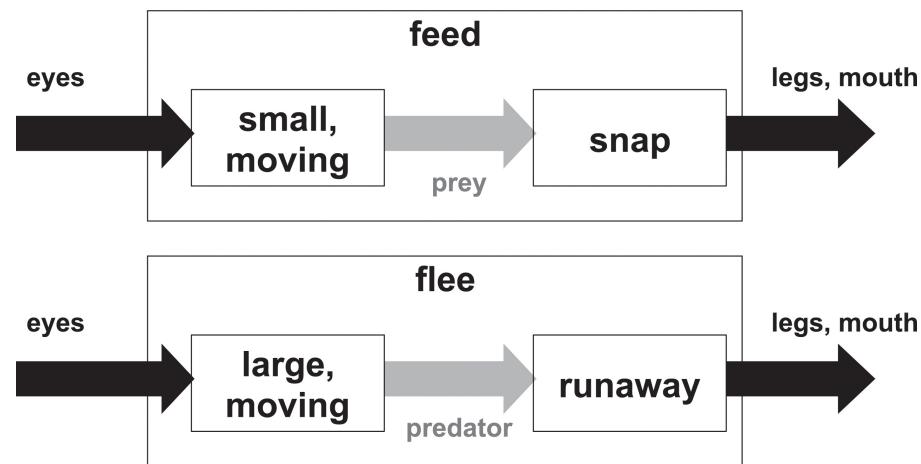
In the 1980s, Arbib and colleagues constructed computer models of visually guided behaviors in frogs and toads in order to better understand intelligence. They called their model *rana computatrix* (*Rana* is the genus of frogs).

Starting with Level 1 of the computational theory approach, toads and frogs have two visually directed behaviors: feed and flee. Level 2 requires expressing the feed and flee behaviors as inputs,

outputs, and transformations. From biology studies, frogs can be characterized as being able to detect only two visual inputs: small moving blobs on their visual field or large moving blobs. The small moving blobs are inputs to the feeding behavior, which produce the output. The output is that the frog orients itself towards the blob (which is called a taxis behavior) and then snaps at it. The small moving blob could be the eraser of a pencil being waved in front of the frog, but because it fits the stimulus of small moving blob, the frog will snap at it as if it were an edible fly. If the object associated with the visual blob turns out not to be a fly, the frog can spit it out. Large moving blobs are inputs to the fleeing behavior, causing the frog to hop away.

PERCEPT

[Figure 6.4](#) shows a graphical representation of this Level 2 conceptualization, in which the frog perceives a small moving blob on its visual field, thereby instantiating an instance of the feed behavior which causes the frog to turn toward that stimulus and snap at it. Note that the stimulus provides a location in the visual field. The stimulus output is called a *percept*; we apply the label “fly” or “prey” to it, but there is no symbol-grounding processing going on. There is merely a stimulus that is coupled to a response. When a frog perceives a large moving blob, the flee behavior is instantiated and the frog hops away. That stimulus can be labeled “predator,” but the label is arbitrary, like choosing a variable name in a program. In robotics, a perceptual variable is generally referred to as a percept.



[Figure 6.4](#) Schema theory of a frog snapping at a fly.

Arbib’s group also explored Level 3 on the computational theory.⁹ They implemented the taxis behavior as a vector field: *rana computatrix* would literally feel an attractive force along the direction of the fly. The relative direction and intensity of the stimulus was represented as a vector. The direction indicated where *rana* had to turn and the magnitude indicated the strength that should be applied to snapping. A weaker stimulus of a small moving blob might indicate a fly at a further distance, and thus the frog would need to snap harder. Level 3 is shown in [figure 6.5](#), where the percept (the location of the centroid of the blob and intensity of the stimulus) and output (a vector) are more precisely defined, though still not expressed in a true software engineering format. The point of the figure is that it is easy

to imagine that there is a function call for detecting small and moving in an image, corresponding to a SENSE primitive, and another function converting the direction and intensity into a vector, corresponding to an ACT primitive.

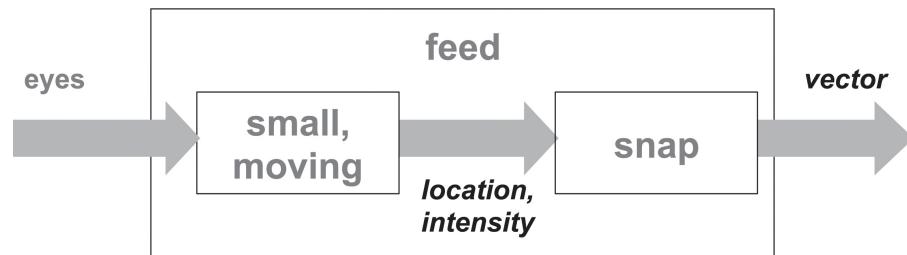


Figure 6.5 Graphical illustration of the feed behavior emphasizing the computational theory Level 3 specificity needed for implementing in a computer.

What is especially interesting is that the computation model can help biologists. The model explained Ingle's observations as to what occasionally happens when a toad sees two flies at once (see figure 6.6).⁴⁰ Toads have eyes that face different directions. It is possible for a toad to simultaneously see a fly with the left eye and a different fly with the right eye. In this situation, each fly triggers a separate instance of the feeding behavior. Each behavior produces the vector that the toad needs to turn to in order to snap at that fly, without being aware that the other behavior exists. According to the vector field implementation of the schema model, the toad now receives two vector inputs, instead of one. What to do? Since the *rana computatrix* implementation uses a vector methodology, the two vectors are summed, resulting in the generation of a third vector in-between the original two! The model predicted that the toad would snap at neither fly, but, instead, snap in-between, which was then observed to happen with real toads. The unexpected interaction of the two independent instances probably is not a significant disadvantage for a toad because if there are two flies in such close proximity, eventually one of them will come back into range.

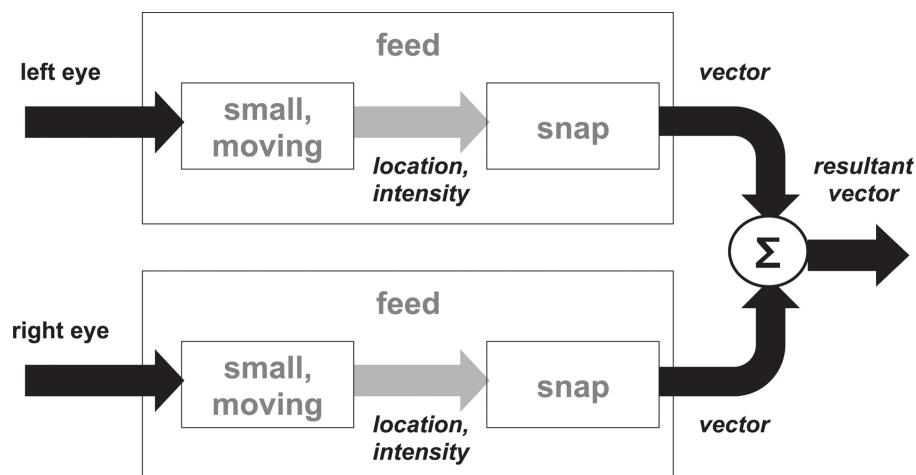


Figure 6.6 Schema theory representation of a toad snapping at a fly when presented with two equidistant flies.

EMERGENT BEHAVIOR

This example illustrates three important lessons for robotics. First, it validates the idea of a computational theory, where functionality in a computer and an animal can be equivalent and used to inspire discoveries. To recap, the equivalence between a computer and an animal is as follows: the concept of behaviors is Level 1 of the computational theory, schema theory expresses Level 2, and the vector field implementation of the motor action is captured at Level 3. Second, the example shows the property of *emergent behavior*, where the agent appears to do something fairly complex, but the observed behavior is really just the result of interaction between simple modules. It also illustrates one reason why it can be hard to simulate emergent behavior; it is often difficult for a designer to imagine the exceptional cases or “corner cases,” such as what happens if the toad simultaneously sees two flies. Third, the example also shows how behaviors correspond to object-oriented programming principles.

6.5 Animal Behaviors

BEHAVIOR

Rana computatrix illustrates how a computational theory spans biological and artificial intelligence. Biological intelligence is the prime source of inspiration for the Reactive Layer in the canonical operational architecture and for behavior-based robots. Therefore it is helpful to introduce definitions from ethology that will be useful later on. A *behavior* is a mapping of sensory inputs to a pattern of motor actions which then are used to achieve a task.

REFLEXIVE BEHAVIOR

STIMULUS-RESPONSE

REACTIVE BEHAVIOR

CONSCIOUS BEHAVIOR

Behaviors can be divided into three broad categories.¹¹ *Reflexive behaviors* are *stimulus-response (S-R)* reactions; for example, when your knee is tapped, it jerks upward. Essentially, reflexive behaviors are “hardwired”; they are neural circuits that insure the stimulus is directly connected to the response in order to produce the fastest response time. *Reactive behaviors* are learned and then consolidated to where they can be executed without conscious thought. Any behavior that involves what is referred to in sports as “muscle memory” is usually a reactive behavior (e.g., riding a bike or skiing). Reactive behaviors can also be changed by conscious thought; a bicyclist riding over a very narrow bridge might “pay attention” to balance, position of hands, position of feet on the cranks, and where to look to the point of unintentionally degrading performance. *Conscious behaviors* are deliberative (e.g., assembling a robot kit, stringing together previously developed behaviors, etc.).

Understanding the differences between the three categories of behaviors is worthy of note for several reasons. First, the Reactive Layer makes extensive use of reflexive behaviors, to the point that some architectures only call a robot behavior a behavior if it is a stimulus-response reaction. Second, the categories can help a designer determine what type of behavior to use, leading to insights about the appropriate implementation. Third, the use of the word “reactive” in ethology is at odds with the way

the word is used in robotics. In ethology, reactive behavior means learned behaviors or a skill; in robotics, it connotes a reflexive behavior. If a roboticists is unaware of these differences, it may be hard to read either the ethological or AI literature without being confused.

6.5.1 Reflexive Behaviors

Reflexive behaviors are particularly interesting, since they imply no need for cognition: if you sense it, you do it. For a robot, this means a hardwired response, eliminating computation and guaranteed to be fast. Indeed, many kit or hobby robots work off of reflexes, represented by circuits.

Reflexive behaviors can be further divided into three categories:¹¹

REFLEXES

1. *reflexes*: where the response lasts only as long as the stimulus, and the response is proportional to the intensity of the stimulus.

TAXES

2. *taxes*: where the response is to move to a particular orientation. Baby turtles exhibit *tropotaxis*; they are hatched at night, hidden from shore birds who normally eat them, and move to the brightest light. Until recently the brightest light would be the ocean reflecting the moon, but the intrusion of man has changed that. Owners of beach front property in Florida now have to turn off their outdoor lights during hatching season to avoid having their lights become a competing source for tropotaxis. Ants exhibit a particular taxis known as *chemotaxis*; they follow trails of pheromones.

FIXED-ACTION PATTERNS

3. *fixed-action patterns*: where the response continues for a longer duration than the stimulus. This type of behavior is helpful for turning around and fleeing predators. It is important to keep in mind that a taxis can result in any pattern of movement relative to a stimulus, not just moving towards.

The above categories are not mutually exclusive. For example, an animal going over rocks or through a forest with trees blocking its view might persist (fixed-action patterns) in orienting itself to the last sensed location of a food source (taxis) before it loses sight of it.

IDIOTHETIC

ALLOTHETIC

The tight coupling of action and perception can often be quantified by mathematical expressions. An example of this is orienting in angelfish. In order to swim upright, an angelfish uses an internal (*idiopathic*) sense of gravity combined with its vision sense (*allothetic*) to see the external percept of the horizon line of the water. If the fish is put into a tank with prisms that make the horizon line appear at an angle, the angelfish will swim cockeyed. On closer inspection, the angle that the angelfish swims at is the vector sum of the vector parallel to gravity and the vector perpendicular to the perceived horizon line! The ability to quantify animal behavior suggests that computer programs can be written to do the same.

6.6 Schema Theory

Schema theory provides a helpful way of translating the insights from animal behaviors into an object-oriented programming format.⁸ As seen with *rana computatrix*, the SENSE and ACT components of the feed and flee behavior are reusable modules. Indeed, the SENSE and ACT modules are templates that can be instantiated with different conditions (e.g., Predator, Prey) to produce different actions. Psychologists have used schema theory since the early 1900s to formally model behaviors and their components.

6.6.1 Schemas as Objects

SCHEMA

SCHEMA CLASS

A schema is used in cognitive science and ethology to refer to a particular organized way of perceiving cognitively and responding to a complex situation or set of stimuli. A *schema* consists of two parts: the knowledge of how to act and/or perceive (knowledge, data structures, models) and the computational process (the algorithm or procedure) it uses to accomplish the activity. The idea of a schema maps nicely onto a class in object-oriented programming (OOP). A *schema class* in C++ or Java contains both data (knowledge, models, releasers) and methods (algorithms for perceiving and acting), as shown below.

Behavior::Schema	
Data	
Methods	perceptual_schema() motor_schema()

SCHEMA INSTANTIATION (SI)

A schema is a generic template for doing some activity, like riding a bicycle. It is a template because a person can ride different bicycles without starting the learning process all over. Since a schema is parameterized like a class, parameters (type of bicycle, height of the bicycle seat, position of the handlebars) can be supplied to the object at the time of instantiation (when an object is created from the class). As with object-oriented programming, the creation of a specific schema is an instantiation, which is specifically called a *schema instantiation (SI)*.

The schema instantiation is the object which is constructed with whatever parameters are needed to tailor it to the situation. For example, there could be a move_to_food schema, where the agent always navigates in a straight line to the food. Notice that “always heads in a straight line” is an activity template, and as a template it is a reusable algorithm for motion control. However, “always heads in a straight line” is just a method. Until the move_to_food schema is instantiated, there is no specific goal to head for (e.g., the candy bar on the table). The same schema could be instantiated for moving to a sandwich. The schema can be instantiated with parameters, such as how hungry the agent is or how fearful that someone else will take the candy bar first. These parameters could lead to the agent performing the same template of action, but faster.

Figure 6.7 illustrates how the basic behavior, motor, and perceptual schema can be expressed in unified model language.

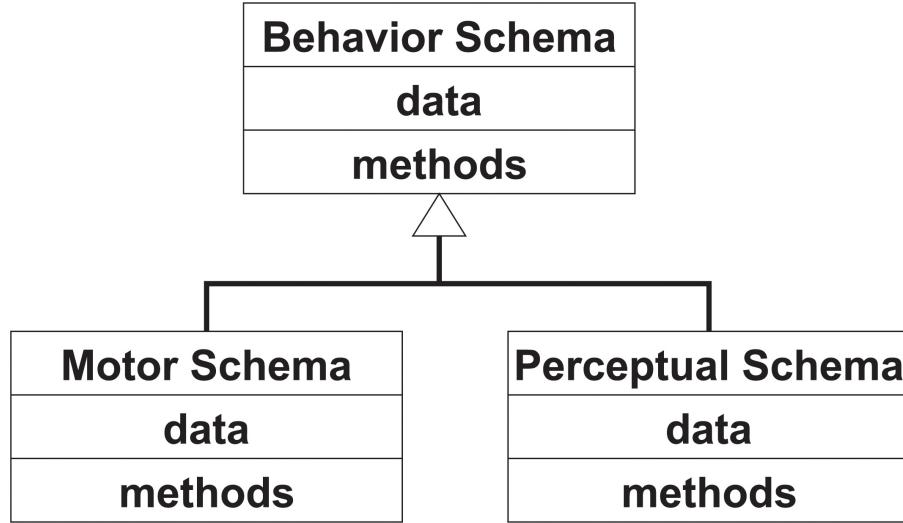


Figure 6.7 Unified model language representation of behavior, motor, and perceptual schemas.

6.6.2 Behaviors and Schema Theory

MOTOR SCHEMA

PERCEPTUAL SCHEMA

In the Arbibian application of schema theory within a computational theory of intelligence, a behavior is a schema which is composed of a motor schema and a perceptual schema (see [figure 6.8](#)). The *motor schema* represents the template for the physical activity, and the *perceptual schema* embodies the stimulus and any fixed-action delays.

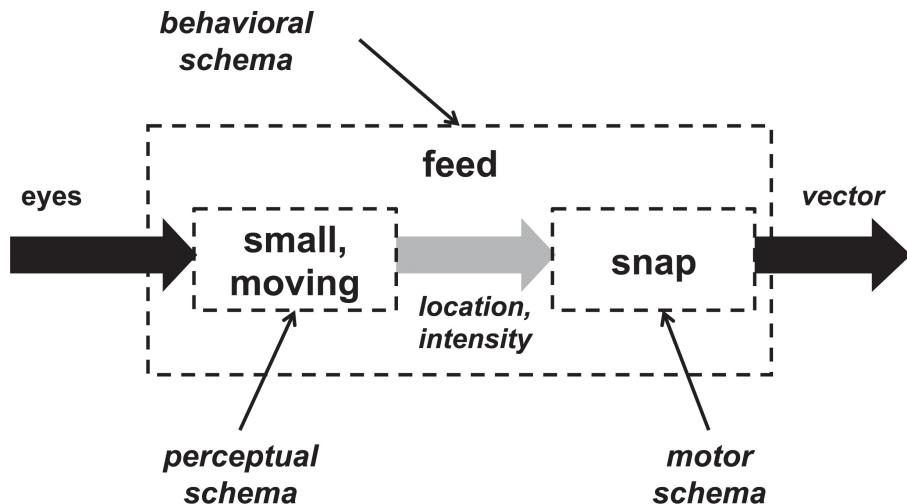


Figure 6.8 Feed behavior decomposed into perceptual and motor schemas.

Essentially, the perceptual and motor schema concepts fit concepts in ethology and cognitive psychology as follows:

- A behavior takes sensory inputs and produces motor actions as an output.
- The transformation of sensory inputs into motor action outputs can be divided into two subprocesses: a perceptual schema and a motor schema.
- A behavior can be represented as a schema, which is essentially an object-oriented programming construct.
- A schema is reusable.

In OOP terms, the motor schema and perceptual schema classes are derived from the schema class. A primitive behavior has just one motor and one perceptual schema.

As objects, schemas can encapsulate two types of knowledge:

1. Schema-specific knowledge, or local data in OOP parlance. The schemas described so far have not used any local data, but it is possible.
2. Procedural knowledge, or methods in OOP parlance. Each of the schemas has a function that performs the necessary computation.

Consistent with OOP, a schema instantiation is specific to a situation and equivalent to an instance in OOP. Likewise, a schema can be composed of other schemas, for example, a behavioral schema is composed of a perceptual and motor schema.

Notice that the schema concept necessitates that there will be libraries of behavioral schemas, perceptual schemas, and motor schemas. From a systems architecture viewpoint, these modules can be expressed as schemas and put into relevant subsystems, such as Perception and Motor, each of which has a library of schemas to instantiate as needed. In implementation terms (or a technical architecture perspective), the schemas are objects.

6.6.3 S-R: Schema Notation

MATHEMATICAL DEFINITION OF BEHAVIOR

The *mathematical definition of behavior* is a mapping of sensory inputs to a pattern of motor actions output which are then used to achieve a task:

$$\{B : S \rightarrow R\}$$

where B is represents the behavior, which takes the input S ("S" for sensing or stimulus) and produces the output R ("R" for reaction or response, which is the same as an action). The definition of a behavior in schema theory is written as:

$$B[S] = R \tag{6.1}$$

where B, S, R are functions. S is a function that take sensor data as input and returns a percept as output. R is a function that takes a percept as an input and returns an action as an output. [Equation 6.1](#)

expresses the SENSE-ACT coupling of robot primitives. It connotes a self-contained loop that is independent of other behaviors; it is complete. A graphical representation of the behavior is shown in figure 6.9.

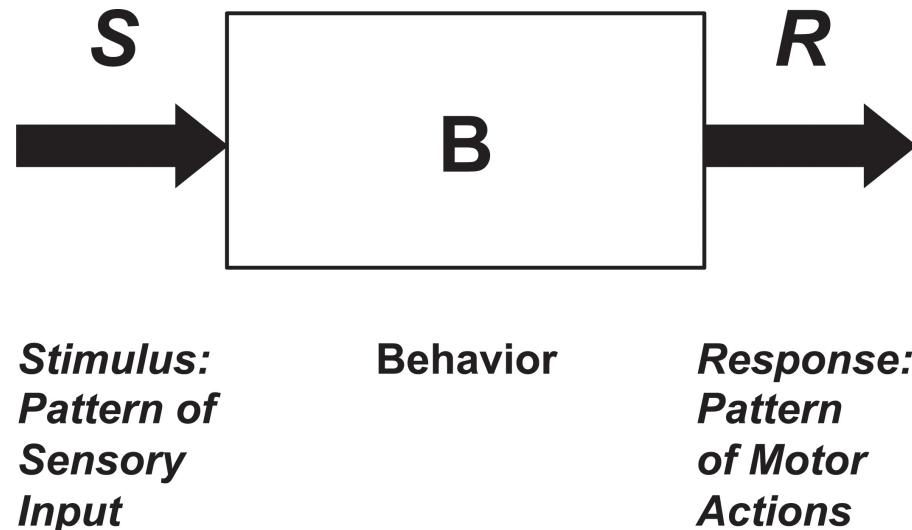


Figure 6.9 A graphical representation of a behavior.

The feeding and fleeing behaviors in *rana computatrix* are expressed in S-R schema notation as:

$$\begin{aligned} \text{Feed[prey]} &= \text{Snap} \\ \text{Flee[predator]} &= \text{Runaway} \end{aligned}$$

Note that the *feed* and *flee* are the names of functions (behavioral schemas) for the behavior. *prey* and *predator* are the names of functions (perceptual schemas) that the behavior calls. Each behavior activates a function (motor schema), either *snap* or *runaway*. As discussed earlier in this chapter, names such as *feed*, *flee*, *prey*, *predator*, *snap*, and *runaway* are arbitrary labels; the names exist because programs require labels for functions and variables. There is no explicit model of a *predator* or *prey*, these are simply a convenient abstraction of “large moving blob” and “small moving blob.” A different programmer might declare the *prey* percept with the name *fly* as a matter of preference. Also note that the behavior function uses the [] brackets instead of (). That is deliberate because the overall behavior of an agent can be the result of multiple concurrent behaviors, which are represented as a matrix of behavioral schemas.

$$B \begin{bmatrix} \text{prey} \\ \text{predator} \end{bmatrix} = \begin{bmatrix} \beta_{\text{feed}} \\ \beta_{\text{flee}} \end{bmatrix}$$

The matrix of behavioral schemas has two corresponding matrices representing the associated stimulus, or perceptual schema *S*, and the response, or motor schema *R*:

$$S = \begin{bmatrix} S_{prey} \\ S_{predator} \end{bmatrix}$$

$$R = \begin{bmatrix} R_{snap} \\ R_{runaway} \end{bmatrix}$$

The subscripts provide an internal label of what the schema does. S_{prey} is a perceptual schema that takes the sensory input and finds the *prey*. However, these internal labels are still vague from a programming standpoint. Therefore, the actual function calls can be substituted for the elements in the matrices.

Consider

$$S = \begin{bmatrix} prey_dir = blob_analysis(vision, SMALL, MOVING) \\ predator_dir = blob_analysis(vision, LARGE, MOVING) \end{bmatrix}$$

This matrix can be interpreted as: The agent has two perceptual schemas, one for each behavior. The perceptual schema associated with the *feed* behavior, β_{feed} , is the function *blob_analysis*. *blob_analysis* takes *vision* as the specific input and *small, moving* as parameters and produces a percept of the blob's relative direction in the agent's visual field, which is called *prey_dir*.

β_{flee} also uses the function *blob_analysis* that takes *vision* as the specific input, but the instance called by β_{flee} takes *large, moving* as parameters and produces a percept of the blob's relative direction in the agent's visual field, which is called *predator_dir*.

The good programming practice of having one general purpose object *blob_analysis* means that, technically, there is really only one perceptual schema with two instantiations. The instantiations of *blob_analysis* are independent and not tied to each other. The animal can be only looking for food, only looking for predators, or both, but the individual behaviors are not aware of anything outside of the scope of their instantiation.

Similarly for the motor schema:

$$R = \begin{bmatrix} snap(turn = prey_dir, hop = NULL) \\ runaway(turn = (predator_dir + 180), hop = FORWARD) \end{bmatrix}$$

The behavioral response, or action, matrix can be interpreted as: *Rana computatrix* has two motor schemas, one for each behavior. The motor schema associated with the *feed* behavior, β_{feed} , is function *snap*, which takes the percept *prey_dir* from the associated perceptual schema to determine where to snap. β_{feed} has another parameter, *hop*, because the action is a direction and a movement along that direction. In feeding, *rana computatrix* should just *turn* towards the stimulus and *snap*. However, in *runaway*, it should both *turn* 180 degrees away from the predator stimulus and *hop*. Naming functions is arbitrary but the general convention is that the overall behavioral description, for example, *feed* or *flees*, is associated with β , while the motor schema name reflects the specific action, for example, *snap* or *runaway*.

6.7 Summary

This chapter has addressed two questions the reader may have about biologically inspired intelligence. The first question posed was: *Is the reactive layer really like ethology?* The answer is “yes,” both from a design perspective and a perspective of how to program it with object-oriented programming. The reactive layer relies on behaviors, which are the fundamental element of biological intelligence. A *behavior* is defined as a mapping of sensory inputs to a pattern of motor actions which then are used to achieve a task. The second question posed by the chapter was: *If so, how do you get that crunchy-chewy animal documentary stuff into a computational format?* The answer is to use Marr’s computational theory as a general framework for transferring biological insights into actual code and then exploit schema theory as a concrete, object-oriented way of representing and thinking about behaviors.

The important attributes of schema theory for behaviors are:

- Schema theory is used to represent behaviors in both animals and computers and is sufficient to describe intelligence at the first two levels of a computational theory. Note that the choice of vector fields to implement feeding and fleeing behaviors was at the third tier of a computational theory and that another computational mechanism could have been chosen.
- A behavioral schema is composed of at least one motor schema and at least one perceptual schema, plus local, behavior-specific knowledge about how to coordinate multiple component schemas. Although this will be discussed in later chapters, it is worth noting for completeness that one of the motor or perceptual schemas in a behavior can be NULL. A NULL schema allows an agent to observe without acting (NULL motor schema) or can act based on an internal drive, such as hunger or the need to reproduce (NULL perceptual schema).
- More than one behavior schema can be instantiated at a time, but each schema acts independently. As a result, the overall behavior of the agent emerges as the interaction of these independent behaviors. The types of interactions of independent behaviors and how they are coordinated is deferred to chapter 8.

Although the chapter answered the two questions posed in the introduction, the content of the chapter may have raised more questions than it answered. It introduced schema theory and its expressive power for translating ethological insights into computational objects, but the chapter did not delve into content, notably: *What goes into the schemas, especially the perceptual schema?* This topic is covered in the next chapter, chapter 7. The chapter also deferred any discussion of non-determinism and emergent behavior. A real concern is whether some sort of coordination is necessary to get predictable complex emergent behavior. The answer is “yes,” coordination is necessary and chapter 8 presents the different types.

6.8 Exercises

Exercise 6.1

What are two reasons why roboticists turn to the biological sciences?

Exercise 6.2

Describe the three levels in a Computational Theory.

Exercise 6.3

Explain, in one or two sentences, each of the following terms: reflexes, taxes, fixed-action patterns, schema.

Exercise 6.4

Represent a generic schema, behavioral schema, perceptual schema, and motor schema with an Object-Oriented Design class diagram.

Exercise 6.5

Where would behavioral schemas, perceptual schemas, and motor schemas be stored in the typical subsystems in a systems architecture?

Exercise 6.6

Lego® Mindstorms® kits contain sensors and actuators which are coupled together in reflexive behaviors. Use Mindstorms to build robots which exhibit the following behaviors:

- a. Reflexive avoid: Turn left when they touch something (use touch sensor and two motors).
- b. Phototaxis: Follow a black line (use the IR sensor to detect the difference between light and dark).
- c. Fixed-action pattern avoid: Back up and turn right when robot encounters a “negative obstacle” (a cliff).

Exercise 6.7

Determine the perceptual and motor schema for the homing and avoid behaviors for a small security-guard robot and express the schemas using S-R notation. The security-guard robot has an acoustic array, a bump sensor, and omnidirectional wheels. If the robot hears a noise, it homes in on it. If the robot is blocked (bumps into something), it turns in a random direction and moves forward.

Exercise 6.8

Repeat the above exercise for the case where the security-guard robot has a bump sensor and radar. In order to avoid an obstacle, if the robot is blocked (bumps into something), it turns in the direction detected by the radar to be the most open, and it moves forward.

[Advanced Reading]

Exercise 6.9

Read the first four chapters in Braitenberg’s *Vehicles*.²⁷ Write a 2- to 5-page paper:

- a. List and describe the principles of behaviors for robotics in this chapter.
- b. Discuss how *Vehicles* is consistent with the biological foundations of reactivity. Be specific, citing which vehicle illustrates what principle or attribute discussed in the book.
- c. Discuss any flaws in the reasoning or inconsistencies between *Vehicles* and the biological foundations of reactivity or computer science.

[Advanced Reading]

Exercise 6.10

Read “Sensorimotor transformations in the worlds of frogs and robots,” by Arbib and Liaw.⁹

- a. List the principles of schema theory that stem from this paper.
- b. Summarize the main contributions of the paper.

6.9 End Notes

For the roboticist’s bookshelf.

Valentino Braitenberg’s *Vehicles: Experiments in Synthetic Psychology* ²⁷ is the cult classic of AI roboticists

everywhere. It does not require any hardware or programming experience, just a couple hours of time and an average imagination to experience this radical departure from the mainstream robotics of the 1970s.

About David Marr.

David Marr's idea of a computational theory was an offshoot of his work bridging the gap between vision from a neurophysiological perspective (his) and computer vision. As is discussed in his book, *Vision*,¹²⁰ Marr had come from England to work on computer vision in the MIT AI lab. The book represented his three years there, and he finished it while on his deathbed dying from leukemia. His preface to the book is heartbreaking.

More about those baby turtles and phototaxis.

To further emphasize how powerful taxes can be and how people have a tendency to miss simple sources of stimulus, it is instructive to consider how it was discovered that baby turtles follow the light to the ocean when they hatched. The story goes that a volunteer left a flashlight on the sand while setting up an experiment intended to show that the baby turtles used magnetic fields to orient themselves. The magnetic field theory was abandoned after the volunteers noticed the baby turtles heading for the flashlight!

7

Perception and Behaviors

Chapter Objectives:

- Explain in one or two sentences each of the following terms: *reflexes, taxes, fixed-action patterns, schema, affordance*.
- Explain the difference between the SENSE, PLAN, ACT cycle, *action- perception cycle*, and *active perception*.
- Be able to write pseudocode of an animal's behaviors in terms of *innate releasing mechanisms*, identifying the *releasers* for the behavior.
- Given a description of perception, classify it as *direct perception* or *recognition*.
- Given a description of an animal's sensing abilities, its task, and environment, identify an *affordance* for each behavior.
- Given a description of an animal's behaviors in terms of its stimuli, strengths, and responses, express the behaviors using *schema theory* and *S-R notation*.

7.1 Overview

Recall *rana computatrix* from the previous chapter, which was used to motivate behaviors in the Reactive Layer and introduce schema theory. The overall behavior for *rana computatrix* could be decomposed into two behaviors, feeding and fleeing, each with a unique percept and action. While the previous chapter provided a formal framework for translating biological insights into object-oriented schemas, it skirted the question: *What goes into the schemas, especially the perceptual schema?*

The perceptual schema is especially intriguing because the action produced by the motor schema is visible to an observer of the robot, but the perception which drives the action is opaque. This opaqueness makes the content of a perceptual schema less obvious. *Rana computatrix* illustrates how percepts can be based on exploiting attributes of the environment. For example, prey and predators can be reliably sensed with the stimuli “small and moving” and “large and moving.” Percepts derived directly from attributes or events in the world are called affordances and will be explored in more detail in this chapter.

Perceiving affordances is not the only perception that influences behavior. The agent may have internal drives, such as hunger. Perception for a behavior may be the result of a combination of perceptual schemas, which leads to more questions about designing perception. The strength of the perception may vary, leading to different strengths of responses. These strengths act as gains on the behavior.

In *rana computatrix*, the two feeding and fleeing behaviors appeared always to be “on” though they did not always produce an action because a toad can just sit there; it does not have to be either running away or snapping at a fly. Ethologists discovered that indeed many behaviors are “off” until a stimulus occurs which turns them on, or releases them. Then the behavior executes based on the perceptual schema-motor schema pattern. This process is generally called an innate releasing mechanism. This means there are two aspects of perception that a designer has to keep in mind, one that triggers behaviors and one that is part of the internal workings of the behaviors.

The presence of behaviors which can be independently released on and off raises a second question: *Don’t you need some sort of coordination to get complex emergent behavior?* The previous chapter and chapter 4 have stressed emergent behavior and non-determinism but have not provided any concrete examples. Thus it is time to explore how emergence happens.

This chapter attempts to address these questions about perceptual schemas and coordinations so that the reader will have a firm basis for deeper dives into perceptual algorithms, locomotion, and the major programming styles for coordinating behaviors. It will first delve into the works of cognitive psychologists Ulrich Neisser,¹⁶⁰ on action-oriented perception, and J.J. Gibson,⁸² on an ecological theory of perception, which provide a foundation for thinking about robotic perceptual schemas. Gibson refuted the necessity of global world models, a direct contradiction to the way perception was handled in the hierarchical paradigm. Gibson’s use of *affordances*, also called *direct perception*, is an important key to the success of the Reactive Paradigm. Later work by Neisser attempts to define when global models and object recognition are appropriate and when an affordance is more elegant. The chapter next covers the work of Lorenz and Tinbergen in describing how concurrent simple animal behaviors interact to produce complex behaviors through Innate Releasing Mechanisms (IRMs).

7.2 Action-Perception Cycle

Ulrich Neisser, who created the term “cognitive psychology” in his book, *Cognition and Reality*, argued that perception cannot be separated from action.¹⁶⁰ He posited that action and perception formed a cycle as seen in [figure 7.1](#).

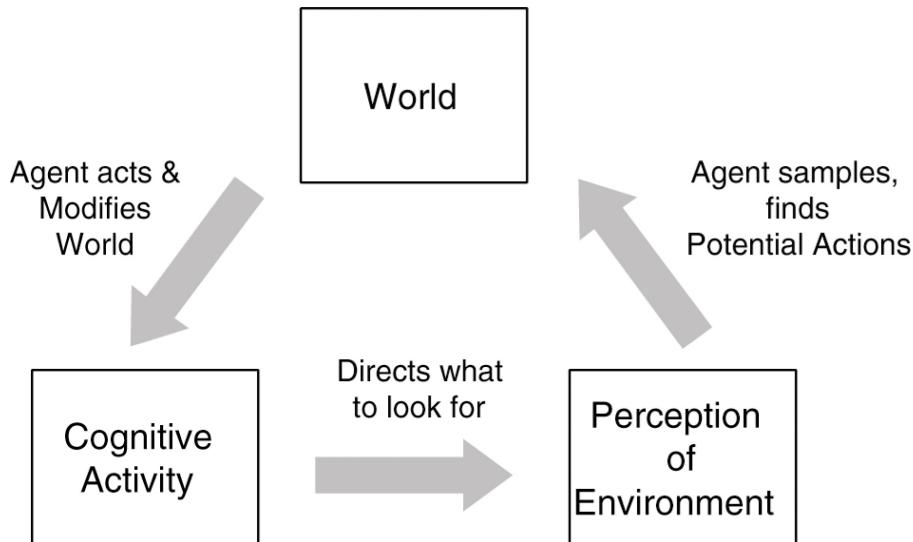


Figure 7.1 Action-Perception Cycle.

ACTION-PERCEPTION CYCLE

The *action-perception cycle* illustrates how perception is fundamental to any intelligent agent. A simple interpretation of the cycle is: When an agent acts, it interacts with its *actual world* because it is situated in that environment; that is, the agent is an integral part of the environment. So, as the agent acts, it changes things or how it perceives things (e.g., move to a new viewpoint, trigger a rock slide, etc.). Therefore the agent's perception of the world *modifies* the *cognitive map*, or state of comprehension, of the agent's goals.

An interesting point is that the updates to the cognitive map are handled by *anticipatory schema*. This schema filters the sensory input based on its goals. If the agent is hungry, the agent might not notice anything that is not relevant to food or perhaps to immediate danger. If the agent is hungry enough, it may miss signs of danger. Thus, the cognitive map may be reactive with little or no memory or object recognition or it may be more deliberative. The cognitive map is not necessarily a global world model but it may be the set of the local world models for behaviors.

In order to obtain or improve its perception, the agent *directs a perceptual exploration* to fill the gaps on the next update(s). The agent may move forward in its environment or take other perceptual actions, such as turning its head to look for a specific face in the crowd or physically probing the environment or an object. This is a type of selective attention or focus-of-attention, where the agent perceptually *samples* the actual world. If the agent acts to externally gather more perception before continuing with its primary action, then that action is sometimes referred to as *active perception*. Part of the sampling process is to determine the potential for action afforded by the world.

There are two important notes about the action-perception cycle. First, the action-perception cycle is not the same as the hierarchical paradigm of SENSE, PLAN, ACT. The action-perception cycle is not about sensing the world and comprehending everything, then planning goals, and finally acting on those goals. Instead there is a background process of constantly perceiving the world and managing the frame problem and introducing additional actions in order to acquire more information. Indeed, the

action-perception cycle contradicts the formal SENSE, PLAN, ACT sequence because perception, either for reaction or deliberation, is running concurrently with acting, suggesting a hybrid paradigm.

Second, the action-perception cycle explains how agents handle the perceptual complexity of the world, or frame problem. The agent's goals and knowledge instantiates anticipatory schema to filter and manage the perception. Presumably there are numerous schemata that are active for each of the agent's different behaviors and goals, including a default schema for looking for danger.

7.3 Gibson: Ecological Approach

The action-perception cycle relied on perception to perceive the potentials for action afforded by the world. The cognitive psychologist J.J. Gibson concentrated on how an agent perceives these potentials for action, which he termed affordances.

Gibson referred to his work as an “ecological approach” because he believed that perception evolved to support actions and that it is silly to try to discuss perception independently of an agent’s environment and its survival behaviors. For example, a certain species of bee prefers one special type of poppy. But, for a long time, the scientists could not determine how the bees recognized that type of poppy because, as color goes, it was indistinguishable from another type of poppy that grows in the same area. Smell? Magnetism? Neither. Then scientists looked at the poppy under ultraviolet and infrared light. In the non-visible bandwidths that particular type of poppy stood out from other poppy species. And indeed, the scientists were able to find retinal components sensitive to those bandwidths of light. The bee and poppy had co-evolved; the poppy’s color evolved to a unique bandwidth while, at the same time, the bee’s retina was becoming specialized at detecting that color. With the bee’s retina “tuned” to the poppy’s color, the bee did not have to do any reasoning about whether there was a poppy in view or if it was the right species of poppy. If that color was present, the poppy was there and thus the potential for feeding was afforded.

Fishermen have exploited affordances since the beginning of time. A fishing lure attempts to emphasize those aspects of a fish’s desired food, thereby presenting the strongest feeding stimulus possible. If the fish is hungry, the stimulus of the lure will trigger feeding. As seen in [figure 7.2](#), to a human fishing lures often look almost nothing like the bait they imitate, yet they exaggerate the affordances of the prey: a minnow is shiny as it catches the light; a perch may have a distinctive gill color or a distinctive movement, and so on.

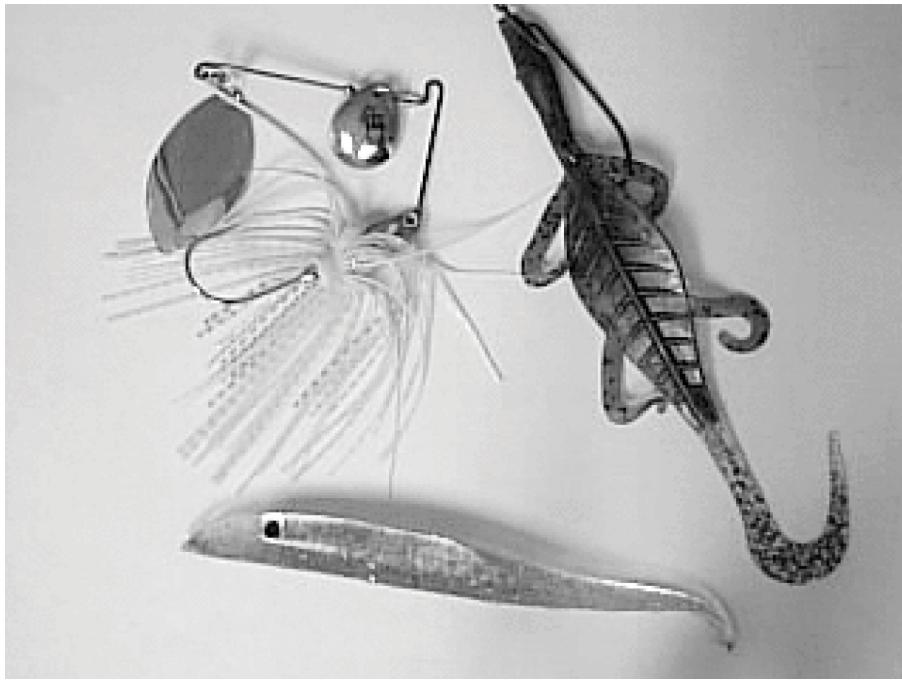


Figure 7.2 A collection of artificial bait, possibly the first example of humans exploiting affordances. Notice that the lures exaggerate one or more attributes of what a fish might eat.

AFFORDANCES

The central tenet of Gibson's approach is that "...the world is its own best representation." Gibson's work is especially interesting because it complements the role of perception in IRM and is consistent with the action-perception cycle. Gibson postulated (and proved) the existence of affordances. *Affordances are perceptible potentialities of the environment for an action.*

What makes Gibson so interesting to roboticists is that an affordance is directly perceivable. *Direct perception* means that the sensing process does not require memory, inference, or interpretation. Therefore it requires minimal computation time and storage, which usually translates to very rapid execution times (near instantaneous) with a computer or robot.

But can an agent actually perceive anything meaningful without some memory, inference, or interpretation? The previous example of *rana computatrix* models how animals can survive, and thrive, without memory, inference, or interpretation. Does this work for humans? Consider that you are walking down the hall when somebody throws something at you. You would most likely duck. You also would probably duck without recognizing the object, although later you may determine that it was only a foam ball. The response happens too fast for any reasoning. You do not say to yourself, "Oh look, something is moving towards me. It must be a ball. Balls are usually hard. I should duck."

7.3.1 Optic Flow

OPTIC FLOW

TIME TO CONTACT

The scenario above leads to one of the most common visual affordance in animals: time to contact. Ducking to avoid an object faster than your brain could explicitly analyze the situation relies on a phenomena called *optic flow*. Optic flow is a neural mechanism for determining motion. Animals can determine time-to-contact quite easily using it. You probably are somewhat familiar with optic flow from driving in a car. When driving or riding in a car, objects in front seem to be in clear focus, but objects on the side of the road are a little blurry from the speed. The point in space that the car is moving towards is the focus of expansion. From that point outward, there is a blurring effect. The more blurring on the sides, the faster the car is going. (Science fiction movies use this blurring effect to simulate faster-than-light travel.) That pattern of blurring is known as a flow field (because it can be represented by vectors, like a gravitational or magnetic field). It is straightforward, neurally, to extract the time to contact, represented in the cognitive literature by τ .

Gannets and pole vaulters both use optic flow to make last-minute, precise movements as reflexes. Gannets are large birds which dive from high altitudes after fish. Because the birds dive from hundreds of feet up in the air, they have to use their wings as control surfaces to direct their dive at the targeted fish. But they are plummeting so fast that, if they hit the water with their wings open, their hollow bones will shatter. Gannets fold their wings just before hitting the water. Optic flow allows the time to contact, τ , to be a stimulus: when the time-to-contact dwindles below a threshold, fold those wings! Pole vaulters also make minute adjustments as to where they plant their pole as they approach the hurdle. This is quite challenging given that the vaulter is running at top speed. It appears that pole vaulters use optic flow rather than reason about where the best place is for planting the pole. (Pole vaulting is not the only instance where humans use optic flow, just one that has been well-documented.)

In most applications, a fast computer program can extract an affordance. Recently, breakthroughs have supported computing optic flow in real time. DJI, a manufacturer of popular small quadcopter UAVs, now offers UAVs with obstacle avoidance using only the onboard camera.

7.3.2 Nonvisual Affordances

Affordances are not limited to vision. A common affordance is knowing when a container is almost filled to the top. Think about filling a jug or the fuel tank of a car. Without being able to see the cavity, a person knows when the tank is almost filled by the change in sound. That change in sound is directly perceivable; the person does not need to know anything about the size or shape of the volume being filled or what the liquid is.

One particularly fascinating application of affordances to robotics, which also serves to illustrate what an affordance is, is the research of Louise Stark and Kevin Bowyer.¹⁹⁹ A seemingly unsurmountable problem in computer vision has been the ability to have a computer recognize an object from a picture. Literally, the computer should say, “That’s a chair,” if the picture is of a chair.

STRUCTURAL MODELS

The traditional way of approaching the problem has been to use *structural models*. A structural model attempts to describe an object in terms of physical components: “A chair has four legs, a seat, and a back.” However, not all chairs fit the same structural model. A typing chair has only one leg, with supports at the bottom. Hanging basket chairs do not have legs at all. A bench does not have a

back. So clearly the structural approach has problems: instead of one structural representation, the computer has to have access to many different models. Structural models also lack flexibility. If the robot is presented with a new kind of chair (say someone has designed a chair to look like your toilet or an upside down trash can), the robot would not be able to recognize it unless someone explicitly constructed another structural model.

Under Gibsonian perception, a chair should be a chair because it affords sitting. And that affordance of sittability should be something that can be extracted from an image:

- Without memory (the agent does not need to memorize all the chairs in the world).
- Without inference (the robot reasons: “If it has four legs, a seat, and a back, then it is a chair. We are in an area which should have lots of chairs, so this makes it more likely it is a chair”).
- Without an interpretation of the image (the robot reasons: “There is an arm rest, and a cushion, …”). A computer should just be able to look at a picture and say if something in that picture is sittable or not.

Stark and Bowyer applied Gibsonian perception to chair recognition in a program called Generic Recognition Using Form and Function (GRUFF) (see [figure 7.3](#)). GRUFF defined sittability as a reasonably level and continuous surface which is at least the size of a person’s butt and at about the height of the knees. (Everything else, like seat backs, just serves to specify the kind of chair.) Stark and Bowyer wrote a computer program which accepted CAD/CAM drawings from students who tried to come up with nonintuitive things that could serve as chairs (like toilets, hanging basket chairs, trash cans). The computer program was able to correctly identify sittable surfaces that even the students missed.

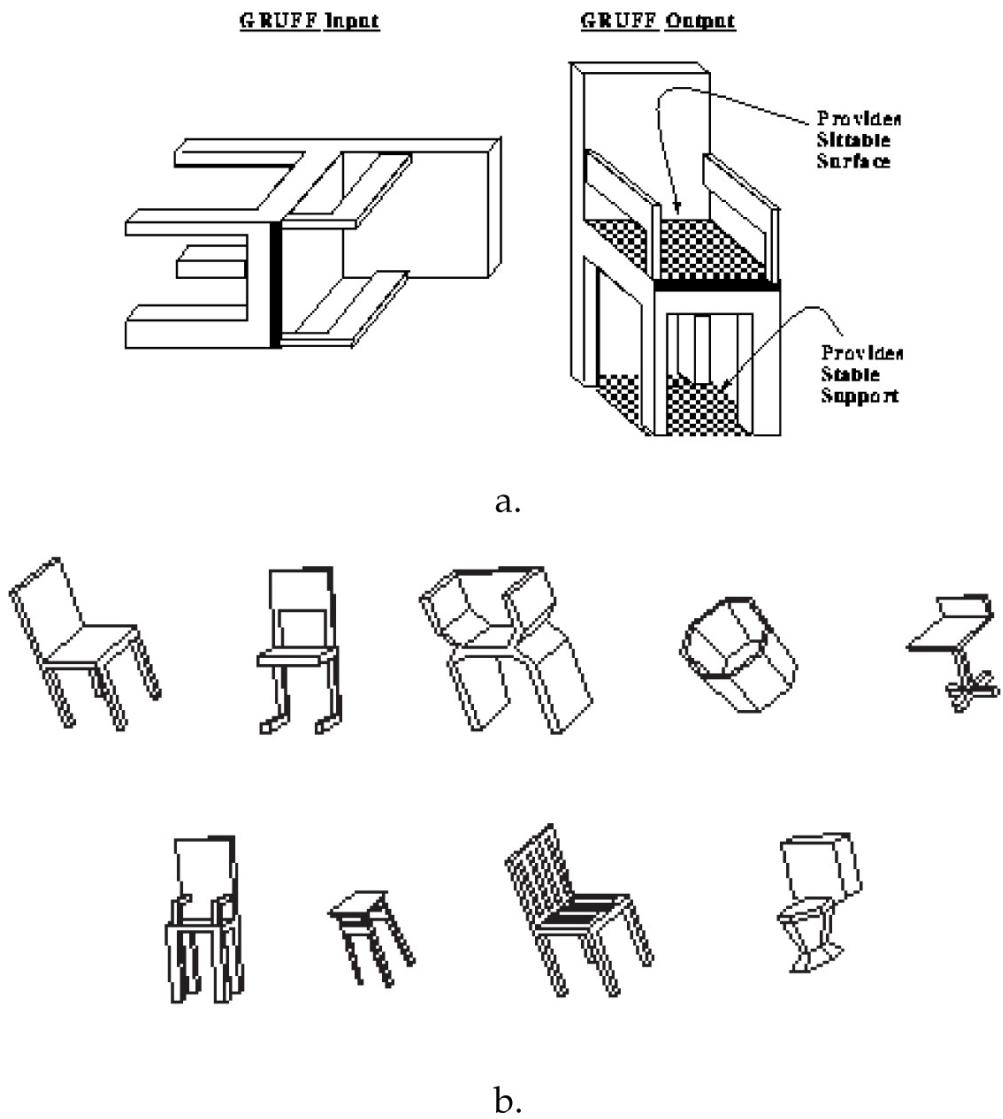


Figure 7.3 The GRUFF system: a.) input, and b.) different types of chairs recognized by GRUFF. (Figures courtesy of Louise Stark.)

It should be noted that Stark and Bowyer are hesitant to make claims about what this says about Gibsonian perception. The computer vision algorithm can be accused of some inference and interpretation (“That is the seat. That is the right height”). But on the other hand, that level of inference and interpretation is significantly different than that involved in trying to determine the structure of the legs, and so forth. And the relationship between seat size and height could be represented in a special neural net that could be *released* whenever the robot or animal got tired and wanted to sit down. The robot would start noticing that it could sit on a ledge or a big rock if a chair or bench was not around.

7.4 Two Perceptual Systems

At this point, the idea of affordances should seem reasonable. A chair is a chair because it affords sitability. But what happens when someone sits in *your* chair? It would appear that humans have some mechanism for recognizing specific instances of objects. Recognition definitely involves memory (“My car is a blue Ford Explorer, and I parked it in slot 56 this morning”). Other tasks, like the kind of sleuthing performed by Sherlock Holmes, may require inference and interpretation. (Imagine trying to duplicate Sherlock Holmes in a computer. It is quite different from mimicking a hungry baby arctic tern.)

So while affordances certainly are a powerful way of describing perception in agents, it is clearly not the only way agents perceive. Neisser postulated that there are two perceptual systems in the brain (and he cites neurophysiological data).¹⁵⁹

DIRECT PERCEPTION

1. *direct perception.* This is the “Gibsonian,” or ecological, track of the brain, and consists of structures low in the brain which evolved earlier on and account for affordances. The direction perception track of the brain may correspond to local world models.

RECOGNITION

2. *recognition.* This is the more recent perceptual track in the brain, which ties in with problem solving and other cognitive activities, and accounts for the use of internal models to distinguish “your coffee cup” from “my coffee cup.” This is where top-down, model-based perception occurs. The recognition track of the brain may correspond to a global world model.

On a more practical note, Neisser’s dichotomy suggests that the first decision in designing a behavior is to determine whether a behavior can be accomplished with an affordance or if it requires recognition. If it can be accomplished with an affordance, then there may be a simple and straightforward way to program it into a robot; otherwise, we will most certainly have to employ a more sophisticated (and slower) perceptual algorithm.

7.5 Innate Releasing Mechanisms

The action-perception cycle, and especially affordances, describes how an agent can perceive the world in a computationally tractable way. But neither describe how the agent translates perception into action: *When does the agent actually act on the potentiality?* and *What if there are multiple potentialities?* The most basic biological model of how behaviors convert perception into action is via a process called *innate release mechanisms (IRM)*.

KONRAD LORENZ NIKO TINBERGEN

Konrad Lorenz and Niko Tinbergen were the founding fathers of ethology. Each man independently became fascinated not only with individual behaviors of animals, but also with the ways in which animals acquired behaviors and selected or coordinated sets of behaviors. Their studies provide some insight into four different ways an animal might acquire and organize behaviors. Lorenz’s and Tinbergen’s works also help with a computational theory Level 2 understanding of how to make a

process acquiring and coordinating behaviors.

The four ways to acquire a behavior are:

INNATE

1. to be born with a behavior (*innate*). An example is the feeding behavior in baby arctic terns. Arctic terns, as the name implies, live in the Arctic where the terrain is largely shades of black and white. However, the Arctic tern has a bright reddish beak. When babies are hatched and are hungry, they peck at the beak of their parents. The pecking triggers a regurgitation reflex in the parent, who literally coughs up food for the babies to eat. It turns out that the babies do not recognize their parents *per se*. Instead, the babies are born with a behavior that says: If hungry, peck at the largest red blob you see. Notice that the only red blobs in their field of vision should be the beaks of adult Arctic terns. The largest blob should be the nearest parent (the closer objects are, the bigger they appear). This is a simple, effective, and computationally inexpensive strategy.

SEQUENCE OF INNATE BEHAVIORS

INTERNAL STATE

2. to be born with a *sequence of innate behaviors*. The animal is born with a sequence of behaviors. An example is the mating cycle of digger wasps. A female digger wasp mates and then builds a nest. Once it sees the nest, the female lays eggs. The sequence is logical, but the important point is the role of stimulus in triggering the next step. The nest is not built until the female mates; mating changes the female's *internal state*. The eggs are not laid until the nest is built; the nest is a visual stimulus releasing the next step. Notice that the wasp does not have to "know" or understand the sequence. Each step is triggered by the combination of its internal state and the environment. This is very similar to Finite State Machines in computer science programming and will be discussed later in chapter 19.

INNATE WITH MEMORY

3. to be born with behaviors that need some memory for initialization (*innate with memory*). An animal can be born with innate behaviors that need customizing based on the situation the animal is born into. An example of this is bees. Bees are born in hives. The location of a hive is something that is not innate; a baby bee has to learn what its hive looks like and how to navigate to and from it. It is believed that the curious behavior exhibited by baby bees (which is innate) allows them to learn this critical information. A new bee will fly out of the hive for a short distance and then turn around and come back. This flight will be repeated, with the bee going a bit farther along the straight line each time. After a time, the bee will repeat the behavior but at an angle from the opening to the hive. Eventually, the bee will have circumnavigated the hive. Why? Well, the conjecture is that the bee is learning what the hive looks like from all possible approach angles. Furthermore, the bee can associate a view of the hive with a motor command ("fly left and down") to get to the opening. The behavior of zooming around the hive is innate; what is learned about the appearance of the hive and where the opening is requires memory.

LEARN

4. to *learn* a set of behaviors. Behaviors are not necessarily innate. In mammals, and especially primates, babies must spend a great deal of time learning. An example of learned behaviors is

hunting in lions. Lion cubs are not born with any hunting behaviors. If they are not taught by their mothers over a period of years, they show no ability to fend for themselves. At first it might seem strange that something as fundamental as hunting for food would be learned, not innate. However, consider the complexity of hunting for food. Hunting is composed of many subbehaviors, such as searching for food, stalking, chasing, and so on. Hunting may also require teamwork with other members of the pride. It requires great sensitivity to the type of the animal being hunted and to the terrain. Imagine trying to write a program to cover all the possibilities of learning how to hunt! While the learned behaviors are very complex, they can still be represented by innate releasing mechanisms. It is just that the releasers and actions are learned; the animal creates the program itself.

Note that four categories suggest that a roboticist has a spectrum of choices as to how a robot can acquire one or more behaviors: from being preprogrammed with behaviors (innate) to somehow learning them (learned). It also suggests that behaviors can be innate but require memory. The lesson here is that, while S-R types of behaviors are simple to preprogram or hardwire, robot designers certainly should not exclude the use of memory. But as will be seen in chapter 8, this exclusion is a common constraint placed on many robot systems. This is especially true in a popular style of hobby robot building called BEAM robotics (biology, electronics, aesthetics, and mechanics), espoused by Mark Tilden. Numerous BEAM robot websites guide adherents through construction of circuits which duplicate memoryless innate reflexes and taxes.

MOTIVATION

An important lesson that can be extracted from Lorenz's and Tinbergen's works are that the *internal state* and/or *motivation* of an agent may play a role in releasing a behavior. Being hungry is a stimulus, equivalent to the pain introduced by a sharp object in the robot's environment. Another way of looking at it is that motivation serves as a stimulus for behavior. Motivations can stem from homeostasis conditions (like being hungry) or an affect (emotions). Emotions, and how they can be modeled as impacting agents, are covered in chapter 18.

One of the most exciting insights is that behaviors can be sequenced to create complex behaviors. Something as complicated as mating and building a nest can be decomposed into primitives or certainly more simple behaviors. The ability to decompose behaviors into primitives has an appeal to the software engineering side of robotics.

7.5.1 Definition of Innate Releasing Mechanisms

INNATE RELEASING MECHANISM

RELEASER

Lorenz and Tinbergen attempted to clarify their studies about how behaviors are coordinated and controlled by giving the process a special name, *innate releasing mechanisms* (IRM). An IRM presupposes that there is a specific stimulus (either internal or external) which releases, or triggers, the stereotypical pattern of action. The IRM activates the behavior. A *releaser* is a latch or a Boolean variable that has to be set. One way to think of IRMs is as a process of behaviors (see [figure 7.4](#)). In a computational theory of intelligence using IRMs, the basic black boxes of the process are behaviors.

Recall that behaviors take sensory input and produce motor actions. But IRMs go further and specify when a behavior gets turned on and off. The releaser acts as a control signal to activate a behavior. If a behavior is not released, it does not respond to sensory inputs and does not produce motor outputs. For example, if a baby arctic tern is not hungry, it does not peck at anything red, even if there is a red beak nearby.

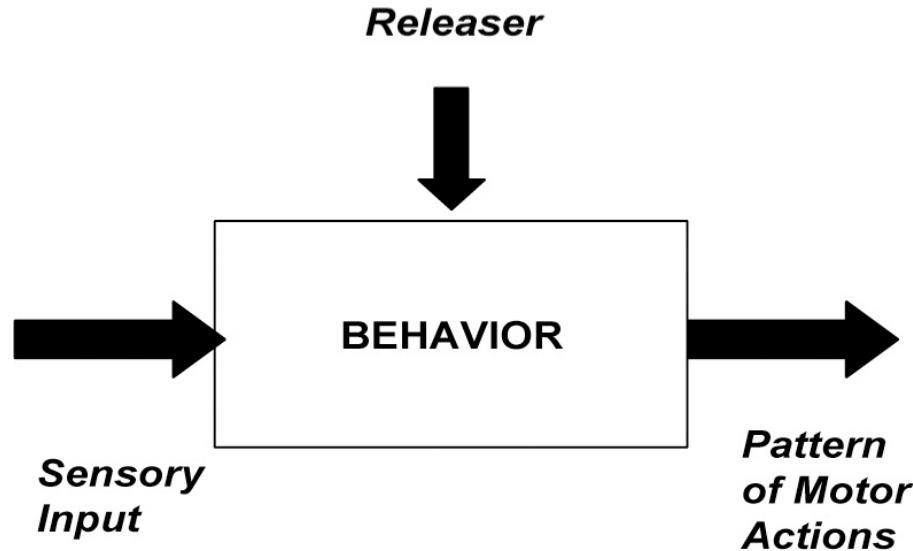


Figure 7.4 Innate Releasing Mechanism as a process for activating behaviors.

Another way to think of IRMs is as a simple computer program shown below. Imagine the agent running a C program with a continuous while loop. Each execution through the loop causes the agent to move for one second. Then the loop would repeat.

```

enum          Releaser={PRESENT, NOT_PRESENT};
Releaser      predator;
while (TRUE)
{
    predator = sensePredators();
    if (predator == PRESENT)
        flee();
}
  
```

In this example, the agent does only two things: sense the world and then flee if it senses a predator. Only one behavior is possible: `flee`. `flee` is released by the presence of a predator. A predator is of type `Releaser` and has only two possible values: it is either present or it is not. If the agent does not sense the releaser for the behavior, the agent does nothing. There is no “default” behavior.

This example also shows filtering of perception. In the above example, the agent looks only for predators with a dedicated detection function, `sensePredators()`. The dedicated predator detection function could be a specialized sense (e.g., retina is sensitive to the frequency of motions associated

with predator movement) or a group of neurons which act as the equivalent of a computer algorithm.

COMPOUND RELEASERS

Another important point about IRMs is that the releaser can be a *compound of releasers*. Furthermore, the releaser can be a combination of either external stimuli (from the environment) or internal state (motivation). If the releaser in the compound is not satisfied, the behavior is not triggered. The pseudocode below shows a compound releaser.

```
enum           Releaser={PRESENT, NOT_PRESENT};  
Releaser       food, hungry;  
while (TRUE)  
{  
    food = senseFood();  
    hungry = checkState();  
    if (food == PRESENT && hungry==PRESENT)  
        feed();  
}
```

The example below shows what happens in a sequence of behaviors, where the agent eats, then nurses its young, then sleeps, and repeats the sequence.

```
enum           Releaser={PRESENT, NOT_PRESENT};  
Releaser       food, hungry, nursed, child;  
while (TRUE) {  
    food = sense();  
    hungry = checkStateHunger();  
    child = checkStateChild();  
    if (hungry==PRESENT)  
        searchForFood(); //sets food = PRESENT when done  
    if (hungry==PRESENT && food==PRESENT)  
        feed(); // sets hungry = NOT_PRESENT when done  
    if (hungry== NOT_PRESENT && child==PRESENT)  
        nurse(); // set nursed = PRESENT when done  
    if (nursed ==PRESENT)  
        sleep();  
}
```

IMPLICIT CHAINING

The behaviors are *implicitly chained* together by their releasers. Once the initial releaser is encountered, the first behavior occurs. It executes for one second (one “movement” interval), and then control passes to the next statement. If the behavior is not finished, the releasers remain unchanged, and no other behavior is triggered. The program then loops to the top and the original behavior executes again. When the original behavior has been completed, the internal state of the animal may have changed or the state of the environment may have been changed as a result of the action. When the motivation and environment match the stimulus for the releaser, the second behavior is triggered, and so on.

The example also reinforces the nature of behaviors. If the agent sleeps and wakes up, but is not

hungry, what will it do? According to the releasers created above, the agent will just sit there until it gets hungry.

In the previous example, the agent's behaviors allowed it to feed and enable the survival of its young, but the set of behaviors did not include fleeing or fighting predators. Fleeing from predators could be added to the program as follows:

```
enum Releaser {PRESENT, NOT_PRESENT};
Releaser food, hungry, nursed, child, predator;
while (TRUE) {
    predator = sensePredator();
    if (predator==PRESENT)
        flee();
    food = senseFood();
    hungry = checkStateHunger();
    child = checkStateChild();
    if (hungry==PRESENT)
        searchForFood();
    if (hungry==PRESENT && food==PRESENT)
        feed();
    if (hungry== NOT_PRESENT && child==PRESENT)
        nurse();
    if (nursed ==PRESENT)
        sleep();
}
```

Notice that this arrangement allowed the agent to flee the predator regardless of where it was in the sequence of feeding, nursing, and sleeping because predator is checked for first. But fleeing is temporary, because it does not change the agent's internal state (except possibly to make it more hungry which would show up on the next iteration). The code may cause the agent to flee for one second, then feed for one second.

One way around this intermittent flee/feed sequence is to *inhibit*, or turn off, any other behavior until fleeing is completed. Inhibition can be accomplished with an *if-else* statement:

```
while (TRUE) {
    predator = sensePredator();
    if (predator==PRESENT)
        flee();
    else {
        food = senseFood();
        hungry = checkStateHunger();
        ...
    }
}
```

The addition of the *if-else* statement prevents other, less important behaviors from executing. It does not solve the problem caused by the predator releaser disappearing as the agent runs away. If the agent turns and the predator is out of view (say behind the agent), the value of predator will go to

NOT_PRESENT in the next update. The agent will go back to foraging, feeding, nursing, or sleeping. Fleeing should be a fixed-pattern action behavior which persists for some period of time, T. The fixed-pattern action effect can be accomplished with:

```
#define T LONG_TIME
while (TRUE) {
    predator = sensePredator();
    if (predator==PRESENT)
        for(time = T; time > 0; time--)
            flee();
    else {
        food = senseFood();
        ...
    }
}
```

The C code examples were implemented as an implicit sequence, where the order of execution depended on the value of the releasers. Here is an implementation of the same behaviors with an explicit sequence:

```
Releaser      food, hungry, nursed, child, predator;
while (TRUE) {

    predator = sensePredator();
    if (predator==PRESENT)
        flee();
    food = senseFood();
    hungry = checkStateHunger();
    child = checkStateChild();
    if (hungry==PRESENT)
        searchForFood();
    feed();
    nurse();
    sleep();
}
```

The explicit sequence at first may be more appealing to a coder because it is less cluttered, and the compound releasers are hidden. But this implementation is not equivalent to the original implicit sequence. It assumes that instead of the loop executing every second and the behaviors acting incrementally, each behavior takes control and runs to completion. Note that the agent cannot react to a predator until it has finished the sequence of behaviors. Calls to the fleeing behavior could be inserted between each behavior, or fleeing could be processed on an interrupt basis. But every “fix” makes the program less general in purpose and harder to modify and maintain.

The main point here is: *Simple behaviors operating independently can lead to, what an outside observer would view as, a complex sequence of actions.*

7.5.2 Concurrent Behaviors

An important point stemming from the examples with the IRMs is that behaviors can, and often do, execute concurrently and independently. What appears to be a fixed sequence may be the result of a normal series of events. However, some behaviors may violate or ignore the implicit sequence when the environment presents conflicting stimuli. In the case of the parent agent, fleeing a predator was mutually exclusive of the feeding, nursing, and sleeping behaviors.

Interesting things can happen if two (or more) behaviors are released that usually are not executed at the same time. It appears that these strange interactions fall into the following categories:

EQUILIBRIUM

- *Equilibrium (the behaviors seem to balance each other out):* Consider feeding versus fleeing in a squirrel when the food is just close enough to a person on a park bench. A squirrel often appears to be visibly undecided as to whether to go for the food or to stay away.

DOMINANCE

- *Dominance of one (winner take all):* you are hungry and sleepy. You do one or the other, not both simultaneously.

CANCELLATION

- *Cancellation (the behaviors cancel each other out):* Male sticklebacks (fish), when their territories overlap, get caught between the need to defend their territory and the need to attack the other fish. So the males make another nest! Apparently the stimuli cancel out, leaving only the stimulus normally associated with nest building.

Unfortunately, it does not appear to be well understood when these different mechanisms for conflicting behaviors are employed. Clearly, there is no one method. But it does require a roboticist who works with behaviors to pay close attention to how the behaviors will interact. The different approaches to handling conflicting behaviors will give rise to the differences in architectures in the Reactive and Hybrid Paradigms, discussed in later chapters.

7.6 Two Functions of Perception

RELEASE

GUIDE

Perception in behavior serves two functions. First, as we saw with IRMs, it *releases* a behavior. However, releasing a behavior is not necessarily the same as the second function: perceiving the information needed to accomplish the behavior. For example, consider an animal in a forest fire. The fire activates the fleeing behavior. But the fleeing behavior needs to extract information about open spaces to avoid obstacles in order to *guide* the behavior. A frightened deer might bolt right past a hunter without apparently noticing him.

ACTION-ORIENTED PERCEPTION

In both roles as a releaser and as a guide for behavior, perception filters the incoming stimulus for

the task at hand. This is often referred to as *action-oriented perception* by roboticists, when they wish to distinguish their perceptual approach from the more hierarchical global models style of perception. Often times, animals have evolved specialized detectors which simplify perception for their behaviors. Some frogs, that sit in water all day with just half their eyes poking above the water line, have a split retina: the lower half is good for seeing in water, the upper half in air.

7.7 Example: Cockroach Hiding

Cockroaches exhibit a reflexive hiding behavior that is quite robust in the open world. These insects can scurry away and find hard-to-reach niches in seconds and yet have little in the way of brains. Their hiding behavior serves as a good case study of how to translate behavioral insights into robot programming by decomposing the overall behavior into more primitive behaviors, identifying the releasers and the implicit sequence of activity with innate releasing mechanisms, the role of internal state and perception, and identifying how to express with S-R notation. The example concludes with a discussion of architectural considerations and how they fit into a systems or a technical architecture.

7.7.1 Decomposition

The hiding behavior, B can be described as follows:

- When the light goes on in a room with a cockroach in it, the cockroach turns and runs toward the dark.
- · When the roach gets to a “wall,” or any other obstacle, such as a chair leg, it follows the “wall.” This is also a taxis where the cockroach orients itself to the “wall.”
- When the cockroach finds a “hiding place,” it goes in and faces outward. This is another taxis where the cockroach orients to the wall. Cockroaches are thigmotrophic (they like to be squeezed), so a hiding place is one where there is pressure or contact on all sides.
- The cockroach then waits, even if the lights are turned back off earlier for some length of time, then comes out. This is a fixed pattern action, as the cockroach waits to resume its activities even if the original stimulus to hide (the lights go on) is removed.

Designers decompose the hiding behavior into three behaviors: flee (β_{flee}), follow-wall ($\beta_{follow-wall}$), and hide (β_{cower}). The decomposition is arbitrary because follow-wall can be considered part of fleeing. In this case, the partitioning is driven by the differences in perception. Detecting light is different from detecting obstacles, which, in turn, seems different from detecting a hiding place.

The hiding behavior also illustrates the problematic assignment of arbitrary names for the component behaviors. The overall behavior is hiding but the last activity of cowering in the crevice would linguistically be referred to as hiding. Rather than have B be named “hiding” and one of the three component behaviors be “hide,” “cower” was chosen instead to attempt to avoid confusion. The behavior names are essentially function names, and thus, the behaviors can be named anything.

In schema notation, the hiding behavior is expressed as:

$$B = \begin{bmatrix} \beta_{flee} \\ \beta_{follow-wall} \\ \beta_{cower} \end{bmatrix}$$

7.7.2 Identifying Releasers

According to innate release mechanism theory, each of the three component behaviors has a releaser. The releaser for flee is “lights on.” The releaser for follow-wall is “blocked” because whenever the cockroach encounters an obstacle, it turns and begins to follow the obstacle on one side. The releaser for “hide” occurs when the cockroach follows a wall and finds itself in a tight spot or “surrounded.”

However, these three releasers do not produce the exact desired overall behavior. What is missing is the stimulus for the fixed-action pattern. Once the lights go on, the cockroach runs until it finds a hiding spot or gives up. Thus, all three behaviors can persist, not just the “hide” behavior. Also, the releasers are such that if the cockroach finds itself blocked, it must begin to follow the obstacle, or if there is a tight space, it must stop there and hide. These actions might always be appropriate for the cockroach to perform, but we want to keep the scope of the behaviors within the specific conditions and not obligate the cockroach to act in unforeseen situations.

One way to capture with IRMs that the overall hiding behavior persists is with the use of internal state. In this case, the cockroach may have an internal state of being “SCARED” which sets an internal timer accessed by the three behaviors. The lights-on stimulus releases the flee behavior but also triggers the SCARED internal state. SCARED is a better releaser for flee because it simplifies how the cockroach can continue to run away for some period of time even if the lights get turned off. Now the releaser for follow-wall is blocked && SCARED and for hide is surrounded && SCARED. When the time for being SCARED is up, the component behavior exits.

Now that we have the releasers, SCARED, blocked, and surrounded, we can move to determining the perceptual and motor schemas and their output. The action associated with the flee behavior was to turn and run. From a computational perspective, these instructions are vague. How many degrees did it turn? How did it measure that it turned enough? If the cockroach has a perceptual schema for sensing the brightest region, then it would be possible to use that to turn 180_circ from the region. But what if the perceptual schema is not reliable, such as a room with even lighting? Instead, for this case, the design is to have the cockroach turn a random number of degrees, and then run, which we will label the turn-and-move motor schema. In a robot, the degree of turn can be controlled with encoders or can be just a timed turn, depending on the sophistication of the robot. Note that the turn-and-move motor schema might be useful in other situations, so we can generalize it (if we want) by letting the degree of turn be an input argument:

turn – and – move(degree = RANDOM)

The perceptual schema within flee is NULL in the above formulation as the turn-and-move(RANDOM) does not require sensing, and the motor schema executes moving until the SCARED timer is up. However, the perceptual schema can also be implemented as a function random-direction that returns a RANDOM number within the turning radius for that robot. This approach can help build libraries of schemas that can be used at a future date for another behavior or robot. The motor schema can be written with the perceptual schema as an argument as:

turn – and – move(random – direction(TURNING – RADIUS))

The action associated with the follow-wall behavior is to turn either to the left or right once the robot is blocked and then have the robot follow the “wall.” If the robot is covered in bump sensors, then it can detect that it is blocked, turn, and then move to keep contact using the bump sensors similar to the way real roaches use their antennae. We will call this the maintain-contact motor schema. In robots with coarse proximity sensing, such as bump sensors or slow sensor update cycles, using contact proximity sensing results in a see-saw or drunken sailor motion, where the robot gets too close to the wall, veers away, loses contact with the wall, then turns back to the wall. The motor schema for follow-wall can be expressed as a function call, with *proximity(bump)* the perceptual schema:

maintain – contact(proximity(bump))

The action for cowering can be captured by an orient-even-pressure motor schema and a perceptual schema that senses the pressure on the surface of the robot. Alternatively, a tactile or contact-sensor array or IR proximity sensors could be used. In this case, the bump sensor is assumed:

orient – even – pressure(pressure – pattern(bump))

If the robot cockroach does not feel enough pressure, the niche is not big enough. We would want the roach to reject the niche and keep searching. The “feel enough” pressure could be expressed as threshold parameter.

orient – even – pressure(pressure – pattern(bump), threshold)

It is important to note that the strength of the stimulus may vary and may lead to emergent behavior. In the examples so far, stimulus has been discrete, either present or not present, and the agent reacts in a binary (either on or off) way. Returning to the arctic terns, consider the attraction to red. The bigger the red area, the more effort the baby tern may put into pecking at it. The tern may execute a sharper turn toward it or peck with more force, which translates into a higher gain for aspects of those actions. This is an example of a stimulus as a continuous function, and stronger stimuli lead to stronger responses.

It is easy to imagine different functions as responses to stimuli. A linear relationship is common. For example, the effort the tern puts into pecking can be proportional to the percentage of the visual field subtended by red. The strength of the stimulus S is written as

$$S = (p, \lambda)$$

where p is the percept and λ is the strength of p .

In S-R notation, the stimulus for the cockroach hide behavior can be written as:

$$S = \begin{bmatrix} SCARED1.0 \\ (BLOCKED, SCARED)1.0 \\ (SURROUNDED, SCARED)percent - surrounded \end{bmatrix}$$

Notice that the `flee` and `follow-wall` behaviors are weighted with full strength, but the amount of effort that will go into the `cower` behavior will be a function of how surrounded the robot agent feels.

ACTIGRAMS

The *actigrams* for the behaviors are shown in [figure 7.5](#). The internal state is shown as a global variable. An actigram is a graphical convention from software engineering where a transformation, or computational action, is represented as a box. Inputs and outputs are shown entering and exiting the box on the X axis, and control or other parameters are shown on the Y axis. Ethologists use a similar graphical convention called an *ethnogram*; we will use the word actigram in this book.

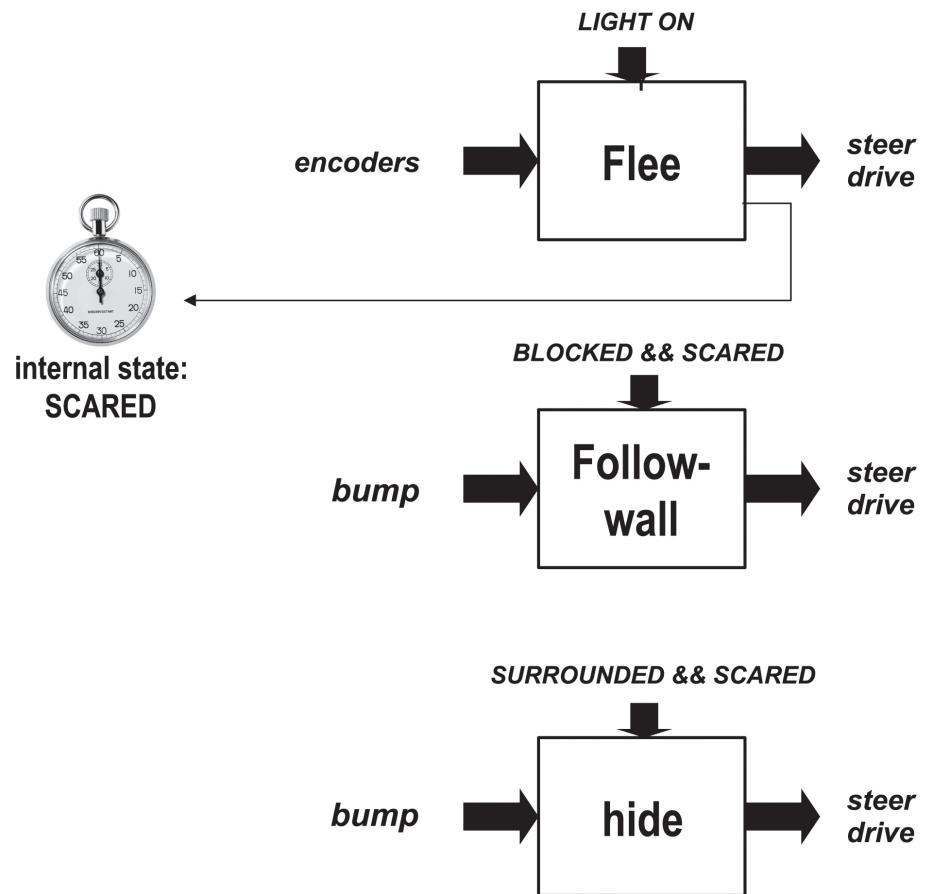


Figure 7.5 The hiding behavior expressed as actigrams and internal state for the set of component behaviors.

7.7.3 Implicit versus Explicit Sequencing

Expressing the overall hiding behavior in S-R notation (below) helps us to visualize the component behaviors as independent and implicitly sequenced.

$$B_{\text{hide}} \begin{bmatrix} \text{flee} \\ \text{follow-wall} \\ \text{cower} \end{bmatrix}$$

$$S = \begin{bmatrix} \text{SCARED}1.0 \\ (\text{BLOCKED}, \text{SCARED})1.0 \\ (\text{SURROUNDED}, \text{SCARED})\text{percent-surrounded} \end{bmatrix}$$

$$R = \begin{bmatrix} \text{turn-and-move}(\text{random-direction}(\text{TURNING-RADIUS})) \\ \text{maintain-contact}(\text{proximity}(\text{bump})) \\ \text{orient-even-pressure}(\text{pressure-pattern}(\text{bump}), \text{percent-surrounded}) \end{bmatrix}$$

The implicit sequence is that turning on a light triggers an internal state, which then releases the turn-and-move behavior. If the internal state is still active and the robot encounters an obstacle, the maintain-contact behavior is released. If the SCARED robot encounters a niche where it is partially surrounded, the orient-even-pressure behavior is released. However, this sequence is implicit, and the robot cockroach could start next to a wall and go directly into maintain-contact().

All three behaviors can be released in any order or simultaneously, so the sequence is implicit, not explicit. The result of simultaneous interaction of the three component behaviors probably would be to drive the robot into a niche. Trying to get away (fleeing) can push the robot further into a niche, and with the desire to maintain contact, drive the robot even deeper. On the other hand, the interaction can be explicitly modulated by coordination mechanisms, such as inhibition, where the orient-even-pressure behavior can turn off the other two behaviors. More mechanisms for coordination will be discussed in the next chapter. However, the rule with respect to coordination is that *less is more*. Attempts to rigidly impose a finite state machine on large sets of behaviors can produce unexpected results because of failures to identify all possible states and transitions.

7.7.4 Perception

The cockroach hiding behavior is an example of the two roles perception plays and how sensors are shared. Perception is used for releasers, in this case, perception of light, which leads to the internal state releasers of SCARED, and contact through bump sensors, which leads to triggering a *follow-wall* or *cower* behavior. However, the bump sensors also serve to guide the execution of the *follow-wall* and *cower* behaviors. *follow-wall* uses the front bump sensors as the perception for its maintain-contact motor schema. *cower* uses the whole set of bump sensors as the perception for its orient-even-pressure motor schema. Both behaviors are independent and have no awareness of what the other behaviors are doing with the sensor input.

The perception of SCARED is interesting. In this case, it is an internal state that is based on the cockroach's perception of the world. The presence of light turned it on. It is easy to imagine that the duration of the internal timer would be related to the strength of the stimulus, that a brighter light or light plus the noise of footsteps would increase the time. The timer could be refreshed, and as more noises or other affordances of predators persisted, the SCARED timer would reset.

The strength of the internal state is similar to gains in a control function. In the cockroach example,

the internal state of SCARED was a timer. But the timer could be reset or increased by other influences. Influences that we normally associate with an emotion, for example, fear or stress or attachment, are called affective. Affective computing and how emotions can be incorporated into a robot at the reactive, deliberative, and interactive layers are discussed in chapter 18.

HOMEOSTASIS

The internal state is also regulated by, and helps regulate, *homeostasis*. The principle of homeostasis is that the agent acts to keep itself internally in balance. The example from Arkin¹¹ is hunger, the hungrier the agent gets, the more priority is given to behaviors which directly impact getting food. An agent may have the same behaviors instantiated, but the gains on those behaviors dynamically change to adapt to the urge. A hungry mountain lion might attack prey that it normally considers too big to risk an encounter; the gain on feeding behaviors has increased. As a result, the response to small percepts is larger than normal, and the duration of fighting with the prey is now longer. The pattern of action is the same, but the gains driven by the need to get hunger back in balance have changed the intensity of the pattern of action.

7.7.5 Architectural Considerations

The cockroach example is intended to explain concepts in biology and illustrate how to think of them computationally. The example was not a principled design process using an ecological decomposition, but rather a way of adding sensors. Design of behavior-based robots will be discussed in detail in chapter 19.

From a systems architecture perspective, the question is: Where do these schemas and mechanisms belong? Returning to the five most common subsystems discussed in chapter 4 (Planning, Cartographer, Navigation, Motor Schema, and Perception), only two are associated with behavior-based robotics. The Perception subsystem is clearly the library for perceptual schemas and the Motor Schema subsystem, the library for motor schemas.

From a technical architecture perspective, biology gives some hints on what goes into those libraries. It answers the question posed in the overview: *What goes into the schemas, especially the perceptual schema?* Perceptual schemas are the knowledge representations for extracting, affordances, releasers, and local internal state measures, such as timers. The internal state can be factored out and serve as a global variable. Even though a behavior schema consists of a perceptual and a motor schema, the motor schema is often synonymous with the behavior, given that it produces the observable action of the behavior.

Biology also provides insights into coordination. In terms of the question from the overview: *Don't you need some sort of coordination to get complex emergent behavior?* The answer is that explicit coordination is not necessary. Behavioral mechanisms, such as internal state, innate releasing mechanisms, and gains can lead to resilient behaviors without explicit coordination. More implicit and explicit coordination methods are discussed in the next chapter.

7.8 Summary

A behavior is the fundamental element of biological intelligence and will serve as the fundamental

component of intelligence in most robot systems. A *behavior* is defined as a mapping of sensory inputs (SENSE) to a pattern of motor actions (ACT) which then are used to achieve a task. Schema theory is a good local procedural knowledge representation, both for acting out a behavior and for acting to get better perception, which is called *active perception*. S-R notation is a good mechanism for expressing schemas computationally.

Implicit chaining eliminates planning (PLAN). Innate Releasing Mechanisms are one model of how behaviors can emerge without the computational constructs of planning, a global world model, or memory. IRMs model intelligence at Level 2 of computational theory, describing the process but not the implementation. In IRM, releasers activate a behavior. A releaser may be either an internal state (motivation) and/or an environmental stimulus. Unfortunately, IRMs do not make the interactions between concurrent, or potentially concurrent, behaviors easy to identify or diagram, and a designer's judgment has to come into play.

Perception in behaviors serves two roles, either as a releaser for a behavior or as the percept which guides the behavior. The same percept can be used both as a releaser and a guide; for example, a fish can respond to a lure and follow it before striking. Perception can have an associated intensity, or stimulus strength, which also influences the motor schema; for example, a fish might strike harder at a lure or pursue it longer if it presents a stronger affordance of prey. In addition to the way in which perception is used, there appear to be two pathways for processing perception. The direct perception pathway uses *affordances*: perceivable potentialities for action inherent in the environment. Affordances are particularly attractive to roboticists because they can be extracted without inference, memory, or intermediate representations. A second important aspect of affordances is that their presence in the environment does not obligate the agent to act. For example, the presence of a large enough blob of red afforded the baby tern with the possibility of feeding, which it acts upon if it is hungry. The presence of a sittable surface does not mean a robot will sit on it; indeed, the robot would filter the perception of the sittable surface until there was a need (releaser) to sit. The recognition pathway makes use of memory and global representations to identify and label specific things in the world.

Important principles for AI robotics which can be extracted from natural intelligence are:

- Agents should decompose complex actions into independent behaviors (or objects), which tightly couple sensing and acting. Behaviors are inherently parallel and distributed.
- In order to simplify control and coordination of behaviors, an agent should rely on straightforward, boolean activation mechanisms (e.g., innate releasing mechanisms) versus rules or finite state machines wherever possible. However, finite state machines are sometimes used, with the state machine as part of the procedural knowledge or method in the behavioral schema. Implicit chaining eliminates planning and maintaining memory of past states through innate releasing mechanisms, and implicit chaining can accomplish some, but not all, of behavioral control. We prefer to do as much as possible with reflexive behaviors, which use only information from the present.
- In order to simplify sensing, perception should filter sensing and consider only what is relevant to the behavior (*action-oriented perception*).
- Direct perception (*affordances*) reduces the computational complexity of sensing and permits actions to occur without memory, inference, or interpretation.

- Behaviors are independent, but the output from one a) may be combined with another to produce a resultant output, or b) may serve to inhibit another (competing-cooperating). Inhibition is a key component of the Subsumption Architecture which will be discussed in chapter 8.

It is also important to remember that natural intelligence does not map perfectly onto the needs and realities of programming robots. One major advantage that animal intelligence has over robotic intelligence is evolution. Animals evolve in a way that leads to survival of the species. But robots are expensive and only a small number are built at any given time, which is counter to evolution. Therefore, individual robots must “survive,” not species. This puts tremendous pressure on robot designers to get a design right the first time. The lack of evolutionary pressures over long periods of time make robots extremely vulnerable to design errors introduced by a poor understanding of the robot’s ecology. Chapter 19 will provide a case study of a robot that was programmed, using the affordance of white, to follow white lines in a path-following competition. The robot was distracted off course by the white shoes of a judge. Fortunately that design flaw was compensated for when the robot got back on course by reacting to a row of white dandelions in seed.

7.9 Exercises

Exercise 7.1

What is the difference between the SENSE, PLAN, ACT sequence and action-oriented perception?

Exercise 7.2

Explain in one or two sentences each of the following terms:

- reflexes
- taxes
- fixed-action patterns
- schema
- affordance.

Exercise 7.3

Many mammals exhibit a camouflage metabehavior. The animal freezes when it sees motion (an affordance for a predator) in an attempt to become invisible. The behavior persists until the predator is very close, and then the animal flees. (This explains why squirrels freeze in front of cars, and then suddenly dash away, apparently flinging themselves under the wheels of a car.) Draw an actigram of the behaviors involved in the camouflage behavior in terms of innate releasing mechanisms, identifying the releasers for each behavior.

Exercise 7.4

Consider a mosquito hunting for a warm-blooded mammal and a good place to bite. Identify the affordances for a warm-blooded mammal and the associated behavior. Represent this with schema theory (perceptual and motor schemas) as specific functions.

Exercise 7.5

Identify the releaser(s) for each behavior below:

- A lobster detects the smell of food and uses the casting behavior (where it turns to one side and if it does not acquire the scent, turns to the other side) to follow the scent.
- When the air temperature rises above 52 degrees F, a male cicada begins chirping as part of its mating behavior.

Exercise 7.6

An unmanned blimp is being used to follow the tradewinds over the ocean while maintaining a constant altitude and sending reports every hour. The balloon does this with two behaviors. The *stay-at-altitude* behavior senses the distance to the water surface (*distance-to-water*) and then adjusts the altitude (*adjust-altitude*). The stay-at-altitude behavior is always on. The *send-update* behavior senses the direction of the tracking tower (*sense-tower*) and turns the antenna to face the tower (*turn-antenna*). The send-update behavior is activated by an internal timer called *report-timer*, which runs every hour.

- a. What is the releaser for the send-update behavior?
- b. Express the overall behavior using S-R notation. Give B, S, and R separately. Use only the italicized labels from the problem description.

Exercise 7.7

What is the difference between direct perception and recognition?

Exercise 7.8

Consider a robot with two vision sensors mounted on each side (like the eyes of a frog). The robot searches for red coke cans among stacks of white newpapers. When it sees a red can, it moves to the can, grasps the can, and then begins searching for a blue recycling bin. When it sees the bin, the robot moves to the bin and deposits the can. If it sees a white bundle, it avoids it. The robot repeats the cycle ad infinitum.

- a. What are the behaviors?
- b. What behavior(s) is/are always active?

Exercise 7.9

Consider a recent autonomous boat competition. One task was to have an autonomous boat navigate out of the harbor, staying the middle of the channel and avoiding any obstacles. The robot was assumed to be placed in the center of the channel facing the correct direction at the start of the channel. The channel is marked with green buoys on one side and red buoys on the other side. The buoys are spaced every 10 meters and the channel is 7 meters wide. A blue buoy marks the end of the course. Suppose your robot entry is a boat 1 meter wide. It has one vision sensor mounted in the front and facing forward with a 180-degree field of view to 40 feet. The vision sensor can extract the center and distance to red, green, and blue. The vision sensor does not pan, tilt, or zoom.

Break this down into behaviors and schemas and releasers. Describe your concept and the type of behavior you are using. Provide sketches. Do not discuss implementation, just describe the behaviors and schemas and express using S-R notation.

[Advanced Reading]

Exercise 7.10

Look up research on the use of motivation and emotion as one means of controlling behaviors. Describe how these relate to internal releasers.

7.10 End Notes

For the roboticist's bookshelf.

J.J. Gibson's *The Ecological Approach to Visual Perception* ⁸² remains the classic tome on perception and cognition. Ulrich Neisser's *Cognitive and Reality* ¹⁶⁰ is also a classic. It is viewed as the first book to solidly establish cognitive psychology. Interestingly, Neisser had viewed his work as antithetical to Gibson's but later became heavily influenced by him. Howard Gardner's *The Mind's New Science* ⁷⁹ gives a nice readable overview of cognitive psychology. He conveys a bit of the controversy Gibson's work caused.

J.J. and E.J. Gibson.

While J.J. Gibson is very well-known, his wife Jackie (E.J. Gibson) is also a prominent cognitive psychologist. They met when he began teaching at Smith College, where she was a student. She raised a family, finished a PhD, and publishes well-respected studies on learning. At least two of the Gibson's students followed their husband-wife teaming: Herb Pick was a student of J.J. Gibson, while his wife, Anne Pick, was a student of E.J. Gibson. The Picks are at the University of Minnesota, and Herb Pick has been active in the mobile robotics community.

8

Behavioral Coordination

Chapter Objectives:

- Be able to write the formula for the *coordination function* and define each component.
- List and describe the three methods of behavioral coordination: *cooperating*, *competing*, and *sequencing*.
- Given a description of a set of behaviors, construct a *potential field* by combining primitive potential fields, calculate the vector output from each behavior, and then combine vectors numerically and graphically to compute the resultant action.
- List three limitations of potential fields for behavioral coordination and possible solutions.
- Given a description of a set of behaviors and their inputs and outputs, design a *subsumption* style coordination scheme using *suppression* and *inhibition* appropriately.
- Represent a sequence of behaviors with a *finite state automata*, showing both the complete transition table and graphical representation. Represent the same sequence with a *script*.

8.1 Overview

The previous chapter introduced innate releasing mechanisms and described how they created implicit chaining because multiple affordances do not (normally) occur in the world. If multiple affordances collide, the instantiated behaviors can create an equilibrium where the animal is frozen in place, exert dominance, where one behavior is the winner, or result in cancellation, where behaviors cancel each other out and the animal reverts to another behavior.¹¹ None of these responses are particularly desirable outcomes of multiple behaviors, especially as the results maybe somewhat random.

Once the designer has a few well-designed and tested individual behaviors, questions such as: *What happens when several behaviors are released (instantiated) at the same time?* and *What about sequences of behaviors?* begin to arise. These questions lead to the study of behavior coordination.

SERIES OF BEHAVIORS

Behavioral coordination is concerned with how independent behaviors interact correctly but, more importantly, with *series of behaviors*, that is, how each behavior operates as part of a recognizable

sequence. One of the first popular series of behaviors that many roboticists looked at was having a robot pick up and dispose of an empty soft drink can. This sequence involved six behaviors: search for a can, move toward the can when it is found, pick up the can, search for the recycle bin, move toward the recycle bin, and drop the can into the bin.^{50;18;77} It is counter-intuitive to think of these behaviors as being concurrent or merged. (There is certainly the possibility of concurrency, for example, avoiding obstacles while moving to the soda can or recycle bin.) Therefore, new techniques had to be introduced for controlling sequences of behaviors. Most of these techniques are conceptually the equivalent of constructing a macro or “abstract” behavior, where the schema structure is used recursively to simplify programming the control program.

This chapter formally introduces behavioral coordination. It begins by presenting the mathematical expression of the coordination function C . The chapter is then divided into three sections, one for each of the major types of coordination functions and their most popular algorithm: potential fields, subsumption, and finite state automata. The chapter concludes with a discussion of AI and behavioral coordination followed by a summary.

8.2 Coordination Function

COORDINATION FUNCTION

The overall *response* of robot ρ is a function of the *behaviors* B , the *gain* of each behavior G , and the *coordination function* C . The coordination function C takes the output of multiple behaviors and produces one response or output. This is shown graphically in figure 8.1 and expressed in S-R notation as:

$$\rho = C(G \times B(S)) \quad (8.1)$$

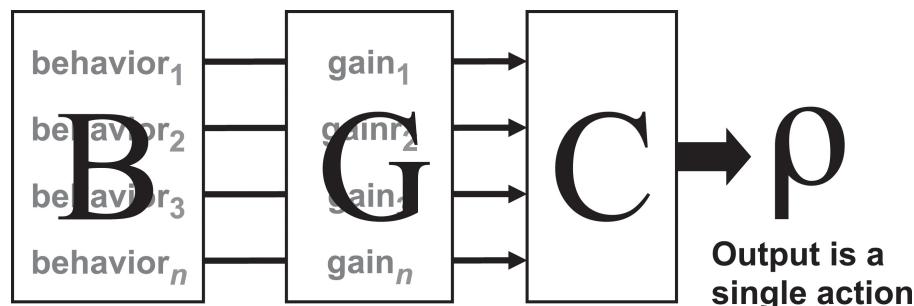


Figure 8.1 A graphical illustration of the coordination function.

COOPERATING METHODS

COMPETING METHODS

SEQUENCING METHODS

There are three categories of coordination functions. Figure 8.2 shows a taxonomy of possible choices of coordination functions. A set of behaviors can be either *concurrent*, that is, all are acting at the same time, or they can be acting as a *sequence*. If the behaviors are concurrent, there are two broad

categories of algorithms for coordinating the set of behaviors. One set is *cooperating methods*, where the coordination function blends the output of the individual behaviors into a single output. The most common cooperating methods are *vector summation*, where the vectors result from *potential fields* representations of the motor schema, and *fuzzy logic*. The other category of concurrent algorithms is *competing methods*, where the individual behaviors undergo arbitration. The most common methods of competition are *subsumption* and *voting*. If the behaviors are to form a complex sequence beyond what can be visualized with IRMs, methods, such as *finite state automata* and *scripts*, serve as *sequencing methods*.

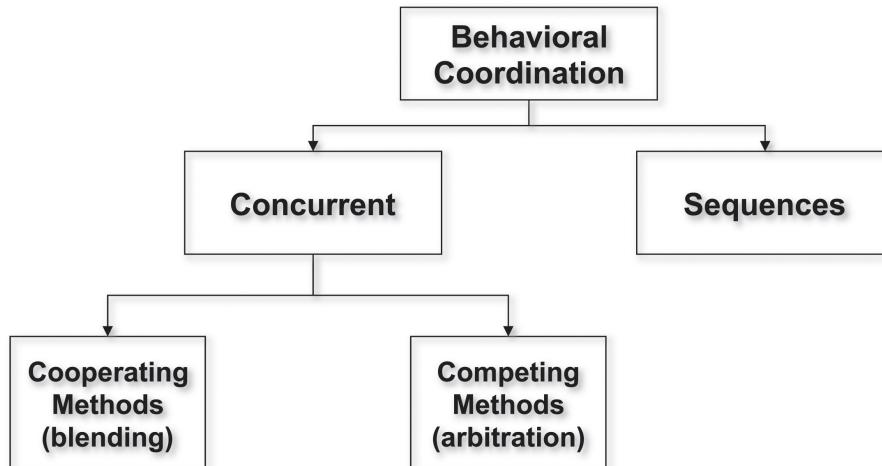


Figure 8.2 A taxonomy of algorithms that can serve as the coordination function.

ABSTRACT BEHAVIOR

There are two important practical notes about coordination functions. First, AI-oriented researchers rarely use rule-based systems to coordinate functions because of the many disadvantages of rules for open worlds and the problems with rule-based systems in general. Second, technical architectures often mix and match coordination functions. A group of behaviors that form a sequence or macro can be grouped into an *abstract behavior* that may use a finite state automata to explicitly coordinate behaviors while any other behaviors, such as avoiding obstacles, might use potential fields.

8.3 Cooperating Methods: Potential Fields

POTENTIAL FIELDS METHODOLOGY

The most common method for cooperating behaviors is to represent the output as a vector. Often the vector is produced by the motor schema using a potential field representation of possible actions, typically referred to as a *potential fields methodology*, or pfields for short. Potential field styles of behaviors always use vectors to represent the action output of behaviors and vector summation to combine vectors from different behaviors to produce an emergent behavior. Fuzzy logic is the other major method and beyond the scope of this chapter.

8.3.1 Visualizing Potential Fields

VECTORS

The first tenet of a potential fields architecture is that the motor action of a behavior must be represented as a potential field. A potential field is an array, or field, of vectors. A *vector* is a mathematical object which consists of a magnitude and a direction. Vectors are often used to represent a force. They are typically drawn as an arrow, where the length of the arrow represents the magnitude of the force and the angle of the arrow represents the direction. Vectors are usually represented with a boldface capital letter, for example, \mathbf{V} . A vector can also be written as a tuple (m, d) , where m stands for magnitude and d for direction. By convention the magnitude is a real number between 0.0 and 1 because the magnitude is normalized to be on the scale of what the robot can actually do. The magnitude can be any real number.

The array represents a region of space. In most robotic applications, the space is in two dimensions, representing a bird's-eye view of the world, just like a map. The map is divided into squares, creating a (x, y) grid. Each element of the array represents a square of space. Perceivable objects in the world exert a force field on the surrounding space. The force field is analogous to a magnetic or gravitation field. The robot can be thought of as a particle that has entered the field generated by an object or environment. The vector in each element represents the force, both the direction to turn and the magnitude or velocity to head in that direction, a robot would feel if it were at that particular spot. Potential fields are continuous because it does not matter how small the element is; at each point in space, there is an associated vector.

[Figure 8.3](#) shows how an obstacle would exert a field on the robot and make it run away. If the robot is close to the obstacle, say within five meters, it is inside the potential field and will feel a force that makes it want to face directly away from the obstacle (if it has not already) and move away. If the robot is not within range of the obstacle, it just sits there because there is no force acting on it. Notice that the field represents what the robot should do (the motor schema) based on whether the robot perceives an obstacle (the perceptual schema). The field is not concerned with how the robot came to be so close to the obstacle; the robot feels the same force if it happens to move within range or if it was just sitting there and someone put a hand next to the robot.

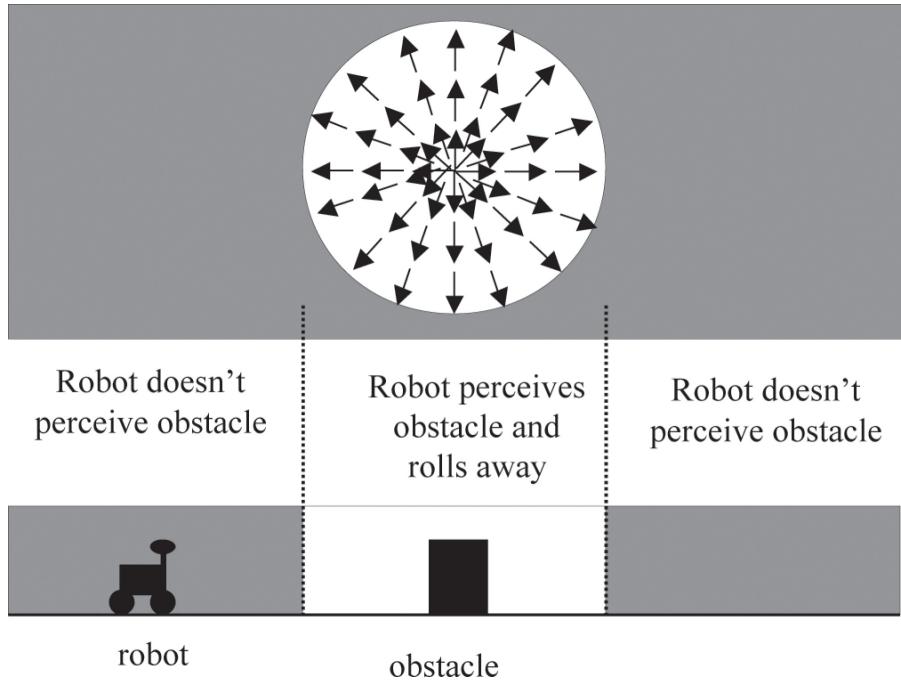


Figure 8.3 Example of an obstacle exerting a repulsive potential field over the radius of one meter.

One way of thinking about potential fields is to imagine a force field acting on the robot. Another way is to think of them as a potential energy surface in three dimensions (gravity is often represented this way) with the robot as a marble. In that case, the vector indicates the direction the robot would “roll” on the surface. Hills in the surface cause the robot to roll away or around (vectors would be pointing away from the “peak” of the hill), and valleys would cause the robot to roll downward (vectors pointing toward the bottom).

FIVE PRIMITIVE FIELDS

UNIFORM FIELD

There are five basic potential fields, or primitives, which can be combined to build more complex fields: *uniform*, *perpendicular*, *attractive*, *repulsive*, and *tangential*. [Figure 8.4](#) shows a uniform field. In a *uniform field*, the robot would feel the same force no matter where it was. No matter where it got set down and at what orientation, it would feel a need to align itself with the direction of the arrow and to move in that direction at a velocity proportional to the length of the arrow. A uniform field is often used to capture the behavior, “go in direction n° .”

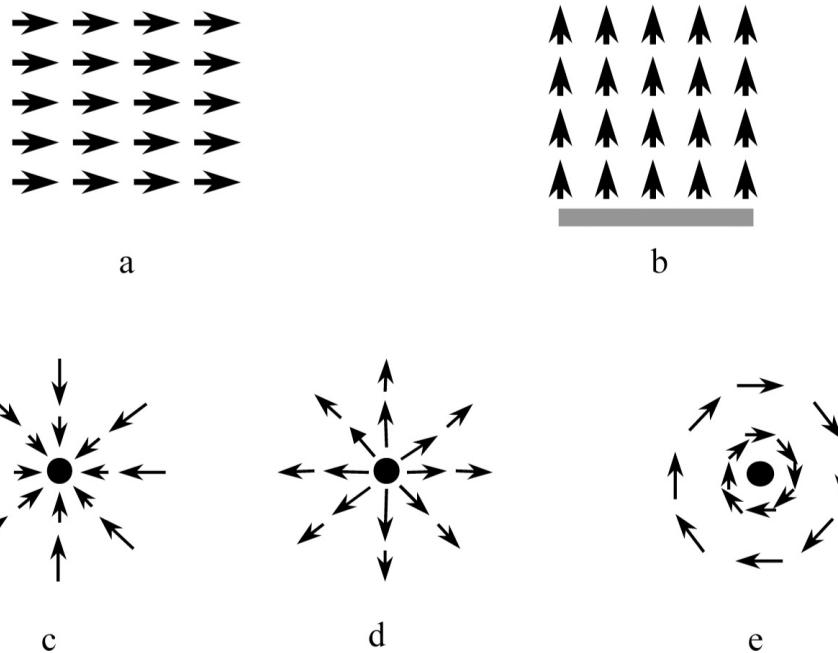


Figure 8.4 Five primitive potential fields: a.) uniform, b.) perpendicular, c.) attraction, d.) repulsion, and e.) tangential.

PERPENDICULAR FIELD

[Figure 8.4b](#) shows a *perpendicular field*, where the robot is oriented perpendicular to some object or wall or border. The field shown is directed away from the gray wall, but a perpendicular field can be pointed toward an object as well.

ATTRACTIVE FIELD

REPULSIVE FIELD

[Figure 8.4c](#) illustrates an *attractive field*. The circle at the center of the field represents an object that is exerting an attraction on the robot. Wherever the robot is, the robot “feels” a force relative to the object. Attractive fields are useful for representing a taxis or tropism, where the agent is literally attracted to light or food or a goal. The opposite of an attractive field is a *repulsive field*, shown in [figure 8.4d](#). Repulsive fields are commonly associated with obstacles or things the agent should avoid. The closer the robot is to the object, the stronger the repulsive force pushes the robot 180° away from it.

TANGENTIAL FIELD

The final primitive field is the *tangential field* illustrated in [figure 8.4e](#). The field is tangent to the object (think of a tangent vector as being perpendicular to radial lines extending outward from the object). Tangential fields can “spin” either clockwise or counterclockwise; [figure 8.4](#) shows a clockwise spin. These fields are useful for directing a robot to go around an obstacle or having a robot investigate something.

8.3.2 Magnitude Profiles

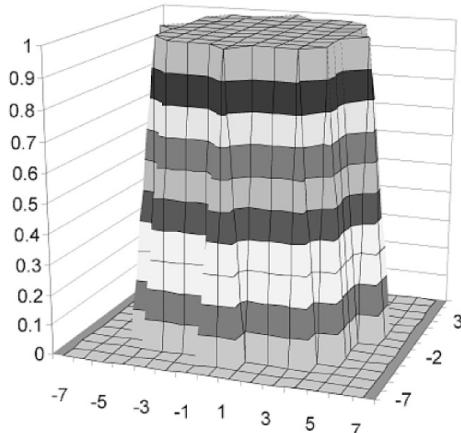
MAGNITUDE PROFILE

Notice that in [figure 8.4](#), the length of the arrows gets shorter as they get closer to the object. The way the magnitude of vectors in the field change is called the *magnitude profile*. (The term “magnitude profile” is used here because the term “velocity profile” is used by control engineers to describe how a robot’s motors accelerate and decelerate to produce a particular movement without jerking.)

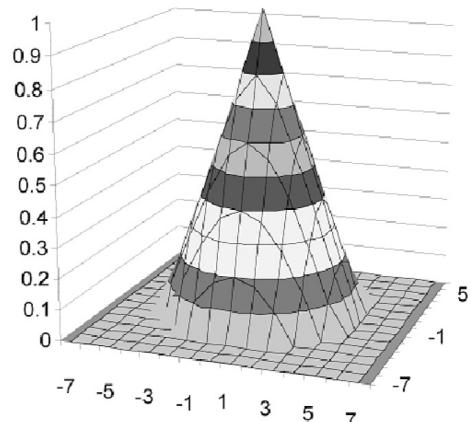
Consider the repulsive field in [figure 8.3](#), mathematically the field can be represented with polar coordinates, and the center of the field being the origin (0,0):

$$(8.2) \quad \begin{aligned} V_{direction} &= -\phi \\ V_{magnitude} &= c \end{aligned}$$

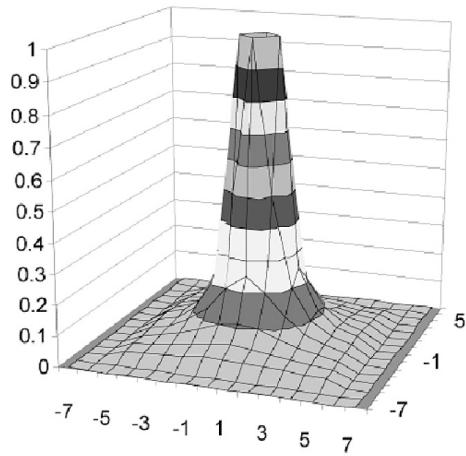
In that case, the magnitude was a constant value, c : the length of the arrows was the same. This can be visualized with a plot of the magnitude shown in [figure 8.5a](#).

Constant Magnitude

a.

Linear Drop off

b.

Exponential drop off

c.

Figure 8.5 Plots of magnitude profiles for a field of radius 5 units: a.) constant magnitude, b.) linear drop off with a slope of -1, and c.) exponential drop off.

This profile says that the robot will run away (the direction it will run is $-\phi$) at the same velocity, no matter how close it is to the object, as long as it is in the range of the obstacle. As soon as the robot gets out of range of the obstacle, the velocity drops to 0.0, stopping the robot. The field is essentially binary: the robot is either running away at a constant speed or stopped. In practice there is a problem with a constant magnitude. It leads to jerky motion on the perimeter of the range of the field. This is illustrated when a robot is heading in a particular direction and then encounters an obstacle. It runs away, leaving the field almost immediately. It turns back to its original path, encounters the field again, and so on.

REFLEXIVITY

LINEAR DROP OFF

Magnitude profiles solve the problem of constant magnitude. They also make it possible for a robot designer to represent reflexivity (that a response should be proportional to the strength of a stimulus) and to create interesting responses. Now consider the profile in [figure 8.4c](#). It shows how an observer would see a robot behave in that field: if the robot is far away from the object, it will turn and move quickly towards it and then slow up to keep from overshooting and hitting the object. Mathematically, this is called a *linear drop off*, since the rate at which the magnitude of the vectors drops off can be plotted as a straight line. The formula for a straight line is $y = mx + b$, where x is the distance and y is magnitude. b biases where the line starts, and m is the slope ($(m = \frac{\Delta y}{\Delta x})$). Any value of m and b is acceptable, positive or negative. If it is not specified, $m = 1$ or -1 (a 45° slope up or down) and $b = 0$ in linear functions.

EXPONENTIAL DROP OFF

The linear profile in [Figure 8.5b](#) matches the behavior desired by the designer: to have the robot react more, the closer it is. But the linear profile shares the problem of the constant magnitude profile in the sharp transition to 0.0 velocity. Therefore, another profile might be used to capture the need for a strong reaction but with more of a taper. One such profile is an *exponential drop off* function, where the drop off in magnitude is proportional to the square of the distance: for every unit of distance away from the object, the force on the robot decreases by half. The exponential profile is shown in [figure 8.5c](#).

As can be seen from the previous examples, almost any magnitude profile is acceptable. The motivation for using magnitude profiles is to fine-tune the behavior. It is important to note that the robot computes only the vectors acting on it at its current location. The figures display the entire field for all possible locations of the robot. The question then arises as to why the figures show an entire field over space. First, it aids in visualizing what the robot will do overall, not just at one particular time step. Second, since fields are continuous representations, it simplifies confirming that the field is correct and makes any abrupt transitions readily apparent.

8.3.3 Potential Fields and Perception

In the previous examples, the force of the potential field at any given point was a function of both the relative distance between the robot and an object and the magnitude profile. The strength of a potential field can be a function of the stimulus, regardless of distance. Recall from chapter 7 the example of the feeding behavior of baby arctic terns, where the feeding behavior is guided by the stimulus “red.” This behavior can be modeled by an attractive field. The bigger and redder an object in the baby’s field of view, the stronger the attraction, suggesting that a magnitude profile using an increasing exponential function will be more appropriate. Another important point that has already been mentioned is that potential fields are egocentric because robot perception is egocentric.

8.3.4 Programming a Single Potential Field

Potential fields are actually easy to program, especially since the fields are egocentric to the robot. The visualization of the entire field may appear to indicate that the robot and the objects are in a fixed,

absolute coordinate system, but they are not. The robot computes the effect of the potential field, usually as a straight line, at every update, with no memory of where it was previously or where it has moved from. This should become clear through the following examples.

A primitive potential field is usually represented by a single function. The vector impacting the robot is computed at each update. Consider the case of a robot with a single range sensor facing forward. The designer has decided that a repulsive field with a linear drop off is appropriate. The formula is:

$$V_{direction} = -180^\circ \quad (8.3)$$

$$V_{magnitude} = \begin{cases} \frac{(D-d)}{D} & \text{for } d \leq D \\ 0 & \text{for } d > D \end{cases}$$

where D is the maximum range of the field's effect, or the maximum distance at which the robot can detect the obstacle, and d is the distance of the robot to the obstacle. (D is not always the detection range. It can be the range at which the robot should respond to a stimulus. For example, many sonars can detect obstacles 20 feet away, producing an almost infinitesimal change in emergent behavior because the obstacles are so far away but requiring the runtime overhead of a function call to compute the negligible force. In practice, a roboticist might set a D of two meters.) Notice that the formula produces a result where $0.0 \leq V_{magnitude} \leq 1.0$.

Below is a C code fragment that captures the repulsive field.

```
typedef struct {
    double magnitude;
    double direction;
} vector;

vector repulsive(double d, double D)
{
    vector outputVector;
    if (d <= D) {
        outputVector.direction = -180; //turn around!
        outputVector.magnitude = (D-d)/D; //linear dropoff
    }
    else {
        outputVector.direction=0.0
        outputVector.magnitude=0.0
    }
    return outputVector;
}
```

At this point, it is easy to illustrate how a potential field can be used by a behavior, RUNAWAY, for the robot with a single sensor. The RUNAWAY behavior will use the `repulsive()` function as the motor schema, and a function `readSonar()` as the perceptual schema. The output of the behavior is a vector. `runaway` is called by the robot on every update cycle.

```

vector runaway( ) {
    double reading;
    vector Voutput;
    reading=readSonar(); //perceptual schema
    Voutput=repulsive (reading, MAX_DISTANCE); //motor schema
    return Voutput;
}
while (robot==ON)
{
    Vrunaway=runaway(reading); // motor schema
    turn(Vrunaway.direction);
    forward(Vrunaway.magnitude*MAX_VELOCITY);
}

```

8.3.5 Combination of Fields and Behaviors

VECTOR SUMMATION

As stated earlier in the chapter, the first attribute of a true potential fields methodology is that it requires all behaviors to be implemented as potential fields. The second attribute is that it combines behaviors by *vector summation*. A robot will generally have forces from multiple behaviors acting on it concurrently. This section provides two examples of how multiple behaviors arise and how they are implemented and combined using vector summation.

The first example is simple navigation, where a robot is heading for a goal (specified as “10.3 m in direction Θ ”) and encounters an obstacle. The motor schema of the move2goal behavior is represented with an attractive potential field, which uses the shaft encoders on the robot to tell if it has reached the goal position. The runaway behavior is a repulsive field and uses a range sensor to detect if something is in front of the robot. [Figure 8.6](#) shows a bird’s-eye view of an area that represents the potential field. The move2goal behavior in [figure 8.7b](#) exerts an attractive field over the entire space; wherever the robot is, it will feel a force from the goal. The runaway behavior in [figure 8.7a](#) exerts a repulsive field in a radius around the obstacle (technically the repulsive field extends over all of the space as does the move2goal, but the magnitude of the repulsion is 0.0 beyond the radius). The combined field is shown in [figure 8.7c](#).

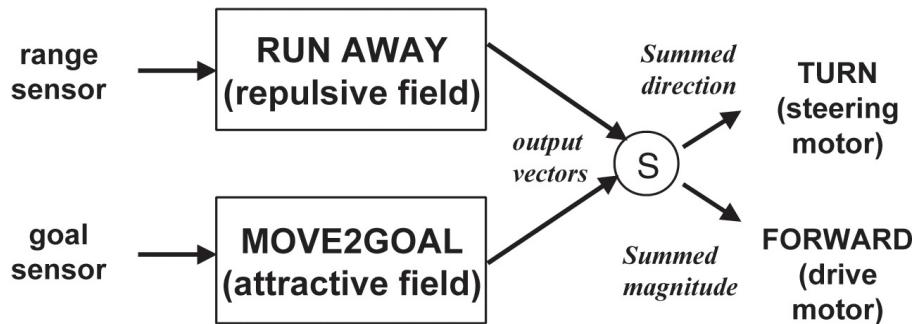
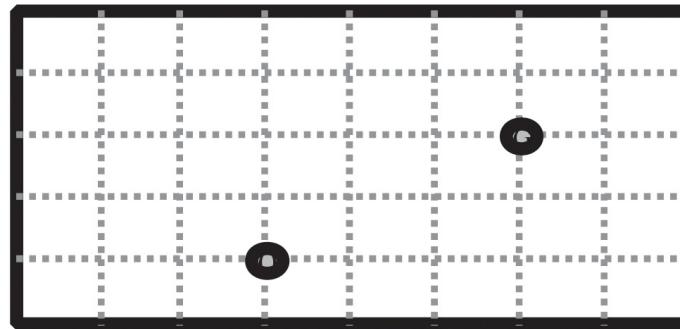


Figure 8.6 A bird's-eye view of a world with a goal and obstacle and the two active behaviors for the robot who will inhabit this world.

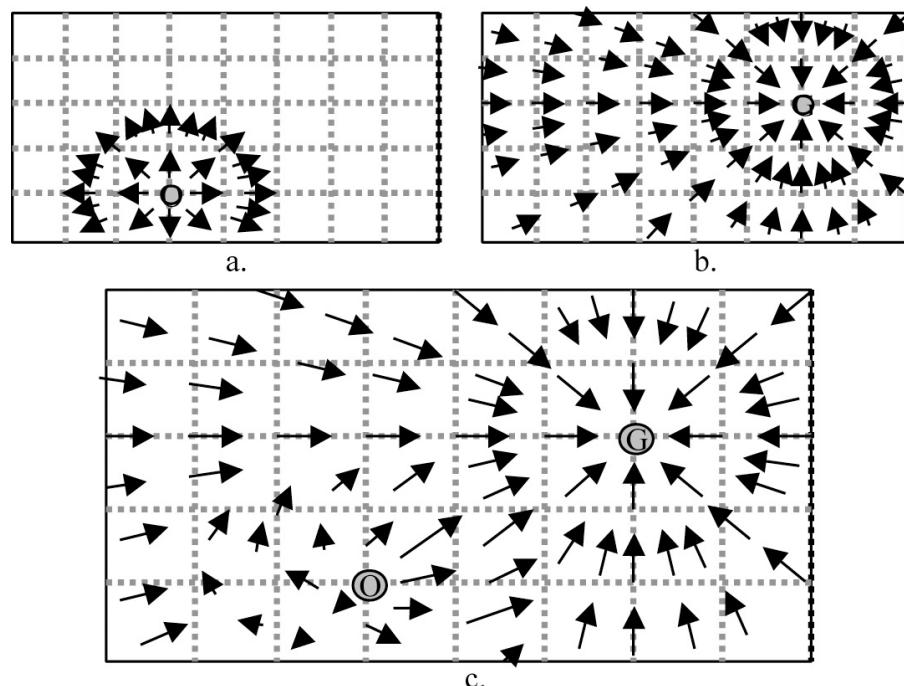


Figure 8.7 Potential fields from the world in [figure 8.6](#): a.) repulsive from the obstacle, b.) attractive from the goal,

and c.) combined.

Figure 8.8 illustrates the emergent behavior of the robot in the field if it starts in the lower right corner. At time t_0 , the robot senses the world. It can perceive only the goal and cannot perceive the obstacle, so the only vector it feels is attraction (runaway returns a vector with magnitude of 0.0). It moves on a straight line for the goal. At t_2 , the robot updates its sensors and now perceives both the goal and the obstacle. Both behaviors contribute a vector; the vectors are summed and the robot now moves off course. At t_3 , the robot has almost moved beyond the obstacle and the goal is exerting the stronger force. At t_4 , the robot resumes course and reaches the goal.

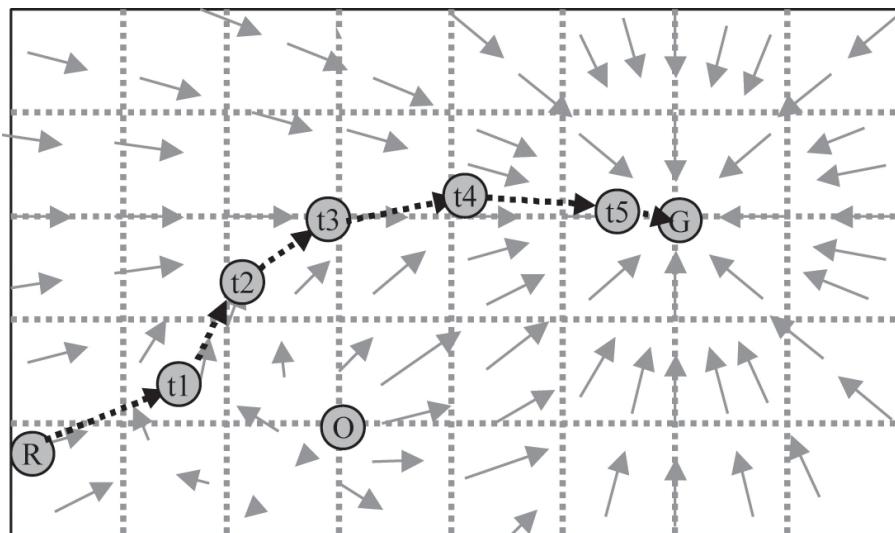


Figure 8.8 Path taken by the robot.

The example illustrates other points about potential fields methods: the impact of update rates, local minima, and holonomicity. Notice that the distance (length of the arrows) between updates is different due to the changes in the magnitude of the output vector, which controls the robot velocity. If the robot has a “shorter” vector, it travels slower and, therefore, covers less distance per unit of time. It can also “overshoot” as seen between t_3 and t_4 , where the robot actually goes farther without turning and then has to turn back to go to the goal. As a result, the path is jagged with sharp lines. The resulting path will be smoother if the robot has a faster update interval. Another aspect of the update rate is that the robot can overshoot the goal, especially if it is using shaft encoders (assume the goal is 10.3 meters from where the robot started). Sometimes designers use attractive fields with a magnitude that drops off as the robot approaches, slowing it down so the that the robot can tell when it has reached the goal. (Programmers usually put a tolerance around the goal location, for example, instead of 10.3 m, the goal is 10.3 m \pm 0.5 m.)

Potential fields treat the robot as if it was holonomic, that is, a particle that could change velocity and direction instantaneously. This is not true for real robots. Research robots, such as Kheperas (shown in figure 8.9), can turn in any direction in place but there is a measurable amount of error in the

turn due to the contact between the wheels and the surface. Many robots have Ackerman, or automobile, steering, and anyone who has tried to parallel park an automobile knows that a car can go only in certain directions—the car cannot go sideways.

LOCAL MINIMA PROBLEM

A third problem is that the fields may sum to 0.0. Returning to [figure 8.7](#), draw a line between the Goal and the Obstacle. Along that line behind the Obstacle, the vectors have only a head (direction of the arrow) and no body (length of the arrow). This means that the magnitude is 0.0 and that if the robot reaches that spot, it will stop and not move again. This is called the *local minima problem* because the potential field has a minima, or valley, that traps the robot. Solutions to the local minima problem will be presented at the end of the chapter.

8.3.6 Example Using One Behavior per Sensor

As another example of how powerful this idea of vector summation is, it is useful to consider how a RUNAWAY behavior for obstacle avoidance is commonly implemented on real robots. [Figure 8.9](#) shows a layout of the IR range sensors on a Khepera robot. Since the sensors are permanently mounted on the platform, their angle α_i relative to the front is known. If a sensor receives a range reading, something is in front of that sensor. Under a repulsive field for RUNAWAY, the output vector will be 180° opposite to angle α_i . The IR sensor is not capable of determining that the obstacle may be a little off the sensor axis, so a reading is treated as if an obstacle were straight in front of that sensor and perpendicular to it.

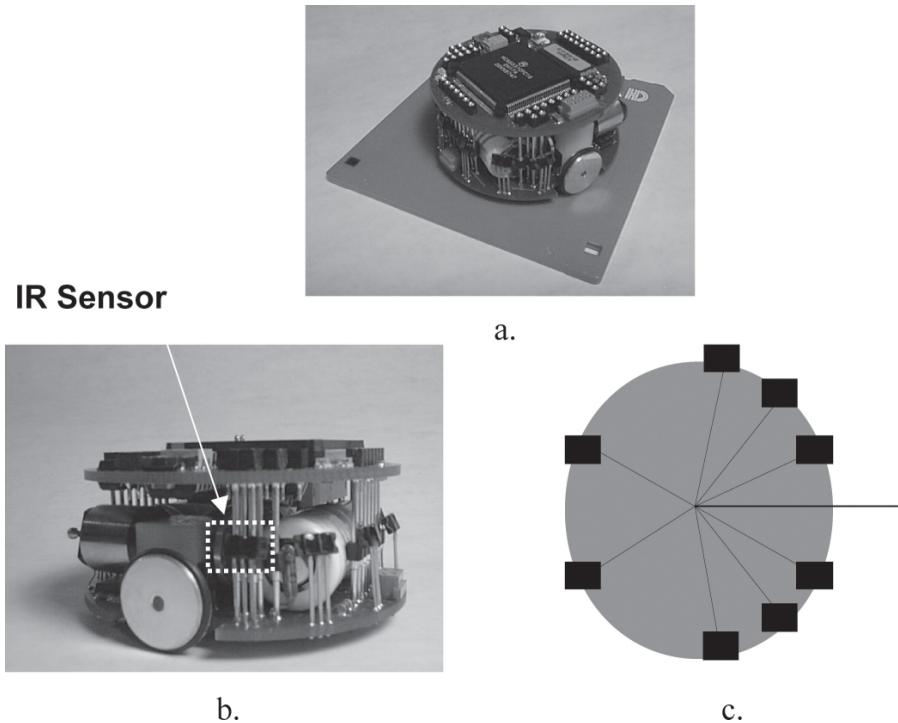


Figure 8.9 Khepera miniature robot: a.) a Khepera on a floppy disk, b.) IR sensors (small black squares) along the

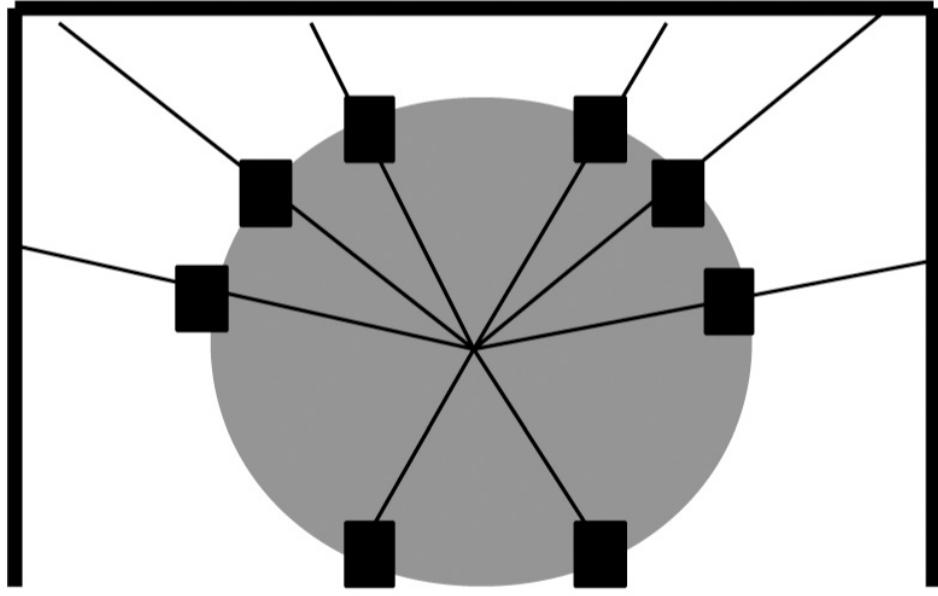
“waistline,” and c.) orientation of the IR sensors.

If this sensor were the only sensor on the robot, the RUNAWAY behavior would be very straightforward. But what if, as in the case of a Khepera, the robot has multiple range sensors? Bigger obstacles would be detected by multiple sensors at the same time. The common solution is to have a RUNAWAY behavior for each sensor. This is called multiple instantiations of the same behavior. Below is a code fragment showing multiple instantiations; all that had to be done is add a `for` loop to poll each sensor. This takes advantage of two properties of vector addition: it is associative ($a + b + c + d$ can be performed as $((a + b) + c) + d$), and it is commutative (it does not matter in what order the vectors are summed).

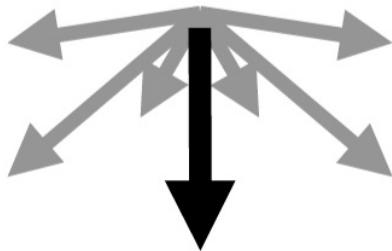
```
while (robot==ON) {  
    vectorOutput.mag=vectorOutput.dir=0.0; //initialize to 0  
    for (i=0; i<=numberIR; i++) {  
        vectorTemp=RUNAWAY(i); // accept a sensor number  
        vectorOutput = VectorSum(vectorOutput,vectorTemp);  
    }  
    turn(vectorOutput.direction);  
    forward(vectorOutput.magnitude*MAX-VELOCITY);  
}
```

BOX CANYON

As seen in [figure 8.10](#), the robot is able to get out of the cave-like trap, called a *box canyon*, without building a model of the wall. Each instance contributes a vector, some of which have a X or Y component that cancels out.



a.



b.

Figure 8.10 Khepera in a box canyon a.) range readings and b.) vectors from each instance of RUNAWAY (gray) and the summed output vector.

From an ethological perspective, the above program is elegant because it is equivalent to behavioral instantiations in animals. Recall from chapter 7 the model of *rana computatrix* and its real-life toad counterpart where each eye sees and responds to a fly independently of the other eye. In this case, the program is treating the robot as if it had eight independent eyes!

FUNCTIONAL COHESION

DATA COUPLING

From a robotics standpoint, the example illustrates two important points. First, the direct coupling of sensing to action works. Second, behavioral programming is consistent with good software engineering

practices. The runaway function exhibits *functional cohesion*, where the function does one thing well, and every statement in the function has something directly to do with the function's purpose.¹⁸⁶ Functional cohesion is desirable because it means the function is unlikely to introduce side effects into the main program or be dependent on another function. The overall organization shows *data coupling*, where each function call takes a simple argument.¹⁸⁶ Data coupling is good because it means all the functions are independent; for example, the program can easily be changed to accommodate more IRs sensors.

PROCEDURAL COHESION

The alternative to multiple instantiation is to have the perceptual schema for runaway process all eight range readings. One approach is to sum all eight vectors internally. (As an exercise, show that the resulting vector from having one behavior sum the vectors from the eight range readings is the same as summing the vectors from eight behaviors.) This is not as elegant from a software engineering perspective because the code is now specific to the robot (the function is said to have *procedural cohesion*)¹⁸⁶ and can be used only with a robot that has eight range sensors at those locations. Another approach, which produces a different emergent behavior, is to have the perceptual schema function call return the direction and distance of the single largest range reading. This approach makes the behavior more selective.

8.3.7 Advantages and Disadvantages

Potential field styles of architectures have many advantages. The potential field is a continuous representation that is easy to visualize over a large region of space. As a result, it is easier for the designer to visualize the robot's overall behavior. It is also easy to combine fields, and languages, such as C++, to support the creation of behavioral libraries. The potential fields can be parameterized: their range of influence can be limited, and any continuous function can express the change in magnitude over distance (linear, exponential, etc.). Furthermore, a two-dimensional field can usually be extended to a three-dimensional field, and so behaviors developed for 2D will work for 3D.

RANDOM NOISE FIELD

Building a reactive system with potential fields is not without disadvantages. The most commonly cited problem with potential fields is that multiple fields can sum to a vector with 0 magnitude; this is called the local minima problem. Return to figure 8.10, the box canyon. If the robot was being attracted to a point behind the box canyon, the attractive vector would cancel the repulsive vector, and the robot would remain stationary because all forces would cancel out. The box canyon problem is an example of reaching a local minimum. In practice, there are many elegant solutions to this problem. One of the earliest was to always have a motor schema producing vectors with a small magnitude from *random noise*.¹² The noise in the motor schema would serve to bump the robot off any local minima. Since the robot could not know when it would encounter a local minimum or maximum, the random noise schema was always active.

NAVIGATION TEMPLATES

Another solution is that of *navigation templates* (NaTs).¹⁹⁶ In NaTs, the avoid behavior receives, as input, the vector summed from the other behaviors. This strategic vector represents the direction the

robot would take if there were no obstacles nearby. If the robot has a strategic vector, that vector gives a clue as to whether an obstacle should be passed on the right or the left. For example, if the robot is crossing a bridge (see figure 8.11), it will want to pass to the left of obstacles on its right in order to stay in the middle of the bridge. Note that the strategic vector defines what is left and what is right.

HARMONIC FUNCTIONS

A third solution to the local minima problem is to express the fields as *harmonic functions*.⁵¹ Potential fields implemented as harmonic functions are guaranteed not to have local minima of 0. This is more computationally expensive as the entire field has to be computed for large areas, not just for the vectors acting on the robot.

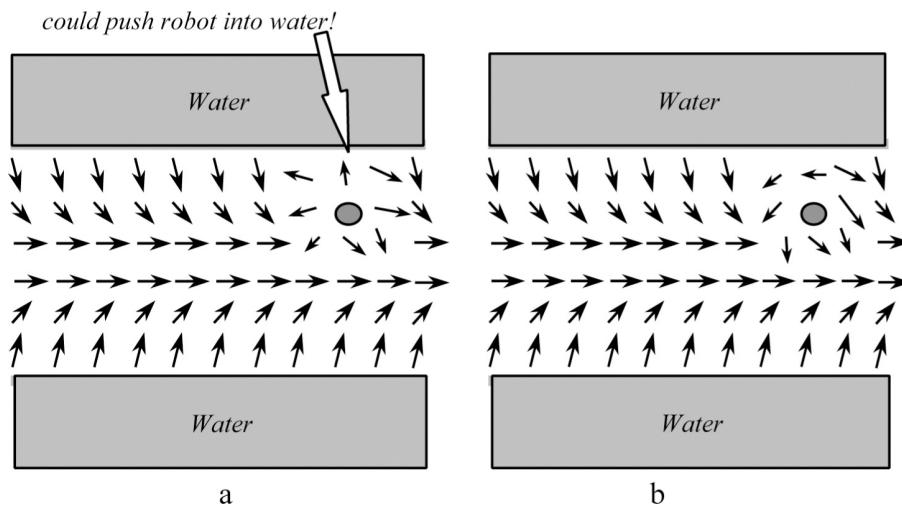


Figure 8.11 Problem with potential fields: a.) an example, and b.) the use of NaTs to eliminate the problem.

To summarize the major points about potential fields architectures:

- Behaviors consist of one or more of both motor and perceptual schemas and/or behaviors. The motor schema(s) for a behavior must be a potential field.
- All behaviors operate concurrently, and the output vectors are summed. Behaviors are treated equally and are not layered, although there may be abstract behaviors which internally sequence behaviors. The coordinated control program is not specified; the designer can use logic, finite state machines, or whatever is deemed appropriate. Sequencing is usually controlled by perceived cues or affordances in the environment, which are releasers.
- Although all behaviors are treated equally, behaviors may make varying contributions to the overall action of the robot. A behavior can change the gains on another behavior, thereby reducing or increasing the magnitude of its output. This means that behaviors can inhibit or excite other behaviors.
- Perception is usually handled by direct perception or affordances.

- Perception can be shared by multiple behaviors. A priori knowledge can be supplied to the perceptual schemas so that they emulate a specialized sensor; for example, a perceptual schema using a range sensor can use a priori knowledge of the width of a hallway to disambiguate a hall from a door.

8.4 Competing Methods: Subsumption

SUBSUMPTION ARCHITECTURE

Competing methods use some form of arbitration, usually either subsumption or voting. In subsumption, higher levels of behaviors subsume or inhibit lower behaviors. In voting, behaviors cast votes leading to a winner-take-all scenario. Rodney Brooks' *subsumption Architecture* (traditionally with a lowercase "s") is the most influential of the purely Reactive systems and the associated competing methods. Part of the influence stems from the publicity surrounding the very naturalistic robots built with subsumption. As seen in [figure 8.12](#), these robots actually looked like shoe-box size insects, with six legs and antennae. The robots were the first to be able to walk, avoid collisions, and climb over obstacles without the "move-think-move-think" pauses of Shakey.

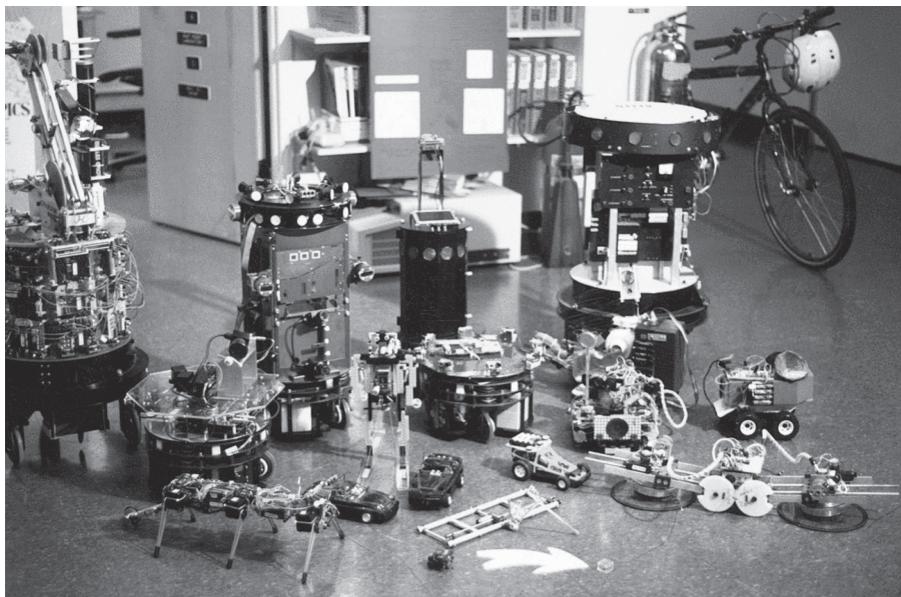


Figure 8.12 “Veteran” robots of the MIT AI Laboratory using the subsumption architecture. (Photograph courtesy of Rod Brooks.)

The term “behavior” in the subsumption architecture has a less precise meaning than in schema theory. A behavior is a network of sensing and acting modules which accomplish a task. The modules are augmented finite state machines AFSM, or finite state machines which have registers, timers, and other enhancements to permit them to interface with other modules. An AFSM is equivalent to the interface between the schemas and the coordinated control strategy in a behavioral schema. In terms of

schema theory, a subsumption behavior is actually a collection of one or more behavioral schemas which this book calls an *abstract behavior*.

Subsumption behaviors are released in a stimulus-response way, without an external program explicitly coordinating and controlling them. There are four interesting aspects of subsumption in terms of releasing and control:

LAYERS OF COMPETENCE

1. Modules are grouped into layers of competence. The layers reflect a hierarchy of intelligence or competence. Lower layers encapsulate basic survival functions, such as avoiding collisions, while higher levels create more goal-directed actions, such as mapping. Each of the layers can be viewed as an abstract behavior for a particular task.

LAYERS CAN SUBSUME LOWER LAYERS

2. Modules in a higher layer can override, or subsume, the output from behaviors in the next lower layer. The behavioral layers operate concurrently and independently, so there needs to be a mechanism to handle potential conflicts. The solution in subsumption is a type of winner-take-all, where the winner is always the higher layer.

NO INTERNAL STATE

3. The use of internal state is avoided. Internal state in this case means any type of local, persistent representation which represents the state of the world, or a model. Because the robot is a situated agent, most of its information should come directly from the world. If the robot depends on an internal representation, what it believes may begin to diverge dangerously from reality. Some internal state is needed for releasing behaviors like being scared or hungry, but good behavioral designs minimize this need.

ACTIVATE TASK BY ACTIVATING APPROPRIATE LAYER

4. A task is accomplished by activating the appropriate layer, which then activates the layers below it, and so on. However, in practice, subsumption style systems are not easily taskable, that is, they cannot be ordered to do another task without the behaviors being reorganized.

8.4.1 Example

LEVEL 0: AVOID

These aspects are best illustrated by an example, extensively modified from Brooks' original paper³² in order to be consistent with schema theory terminology and to facilitate comparison with a potential fields methodology. A robot capable of moving forward, while not colliding with anything, could be represented with a single layer, Level 0. In this example, the robot has multiple sonars (or other range sensors), each pointing in a different direction, and two actuators, one for driving forward and one for turning.

POLAR PLOT

Following [figure 8.13](#), the SONAR module reads the sonar ranges, does any filtering of noise, and produces a *polar plot*. A polar plot represents the range readings surrounding the robot in polar coordinates (r, θ). As shown in [figure 8.14](#), the polar plot can be “unwound.”

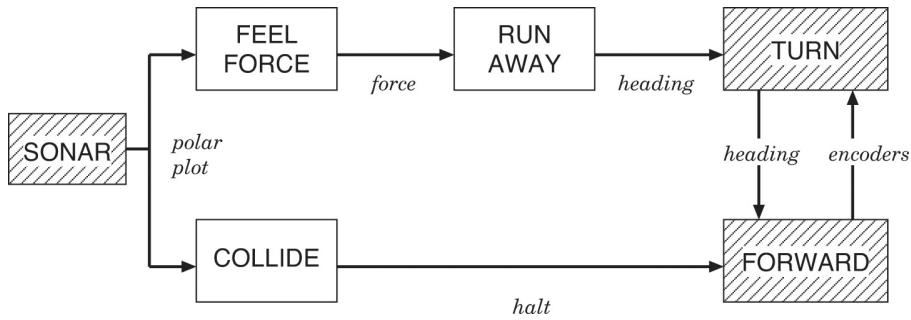


Figure 8.13 Level 0 in the subsumption architecture.

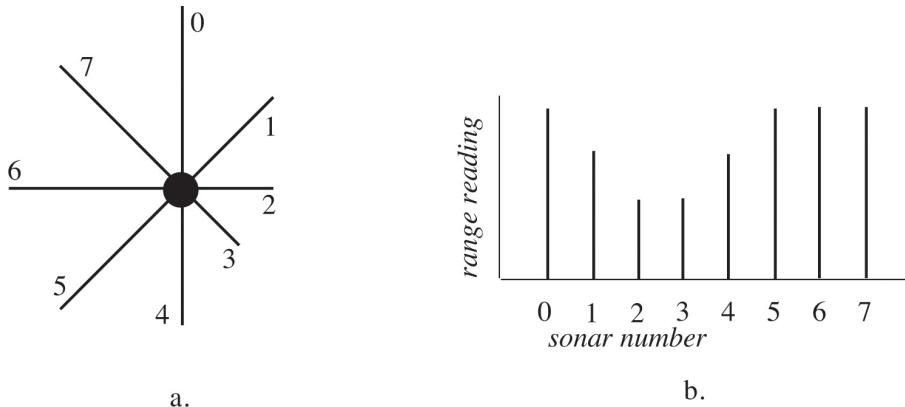


Figure 8.14 Polar plot of eight sonar range readings: a.) “robocentric” view of range readings along acoustic axes, and b.) unrolled into a plot.

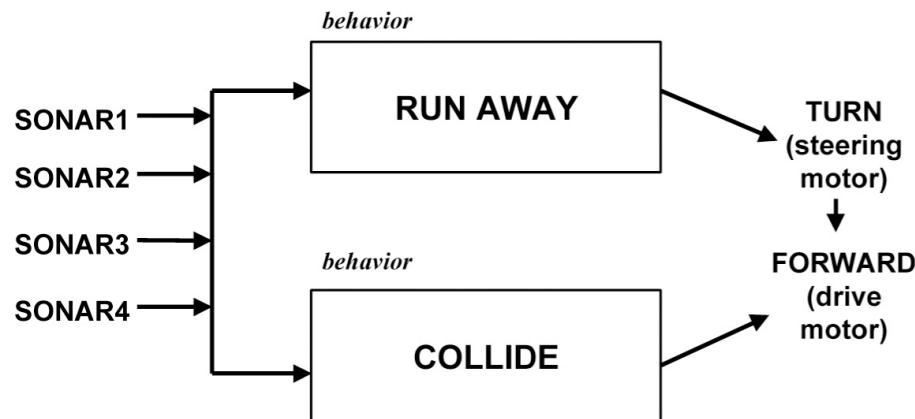
If the range reading for the sonar facing dead ahead is below a certain threshold, the **COLLIDE** module declares a collision and sends the halt signal to the **FORWARD** drive actuator. If the robot was moving forward, it now stops. Meanwhile, the **FEELFORCE** module is receiving the same polar plot. It treats each sonar reading as a repulsive force, which can be represented as a vector. Recall that a vector is a mathematical construct that consists of a magnitude and a direction. **FEELFORCE** can be thought of as summing the vectors from each of the sonar readings. This results in a new vector. The repulsive vector is then passed to the **TURN** module. The **TURN** module splits off the direction to turn and passes that to the steering actuators. **TURN** also passes the vector to the **FORWARD** module, which uses the magnitude of the vector to determine the magnitude of the next forward motion (how far or how fast). So the robot turns and moves a short distance away from the obstacle.

The observable behavior is that the robot will sit still if it is in an unoccupied space until an obstacle comes near it. If the obstacle is on one side of the robot, the robot will turn 180° the other way and move forward; essentially, it runs away. This allows a person to herd the robot. The robot can react to an obstacle if the obstacle (or robot) is motionless or moving; the response is computed at each sensor update.

However, if part of the obstacle, or another obstacle, is dead ahead (someone tries to herd the robot into a wall), the robot will stop and then apply the results of runaway. So it will stop, turn, and begin to

move forward again. Stopping prevents the robot from side-swiping the obstacle while it is turning and moving forward. Level 0 shows how a fairly complex set of actions can emerge from very simple modules.

It is helpful to recast the subsumption architecture in the terms used in this book, as shown in [figure 8.15](#). Note how this looks like the vertical decomposition in subsumption: the sensor data flow through the concurrent behaviors to the actuators, and the independent behaviors cause the robot to do the right thing. The SONAR module would be considered a global interface to the sensors, while the TURN and FORWARD modules would be considered part of the actuators (an interface). For the purposes of this book, a behavior must consist of a perceptual schema and a motor schema. Perceptual schemas are connected to a sensor, while motor schemas are connected to actuators. For Level 0, the perceptual schemas would be contained in the FEELFORCE and COLLIDE modules. The motor schemas are RUNAWAY and FORWARD modules. COLLIDE combines both perceptual processing (extracts the vector for the sonar facing directly ahead) and the pattern of action (halt if there is a reading). The primitive behaviors reflect the two paths through the layer. One might be called the runaway behavior and the other the collide behavior. Together, the two behaviors create a rich obstacle avoidance behavior, or a layer of competence.



[Figure 8.15](#) Level 0 recast as primitive behaviors.

It should also be noticed that the behaviors used direct perception, or affordances. The presence of a range reading indicated there was an obstacle; the robot did not have to know what the obstacle was.

LEVEL 1: WANDER

Consider building a robot which actually wanders around instead of sitting motionless but is still able to avoid obstacles. Under subsumption, a second layer of competence (Level 1) would be added as shown in [figure 8.16](#). In this case, Level 1 consists of a WANDER module which computes a random heading every n seconds. The random heading can be thought of as a vector. The robot needs to pass this heading to the TURN and FORWARD modules. But it cannot be passed to the TURN module directly. That would sacrifice obstacle avoidance because TURN only accepts one input. One solution is to add another module in Level 1, AVOID, which combines the FEELFORCE vector with the WANDER vector.

Adding a new AVOID module offers an opportunity to create a more sophisticated response to obstacles. AVOID combines the direction of the force of avoidance with the desired heading. This results in the actual heading being mostly in the right direction rather than having the robot turn around and lose forward progress. Notice also that the AVOID module was able to “eavesdrop” on components of the next lower layer. The heading output from AVOID has the same representation as the output of RUNAWAY, so TURN can accept from either source.

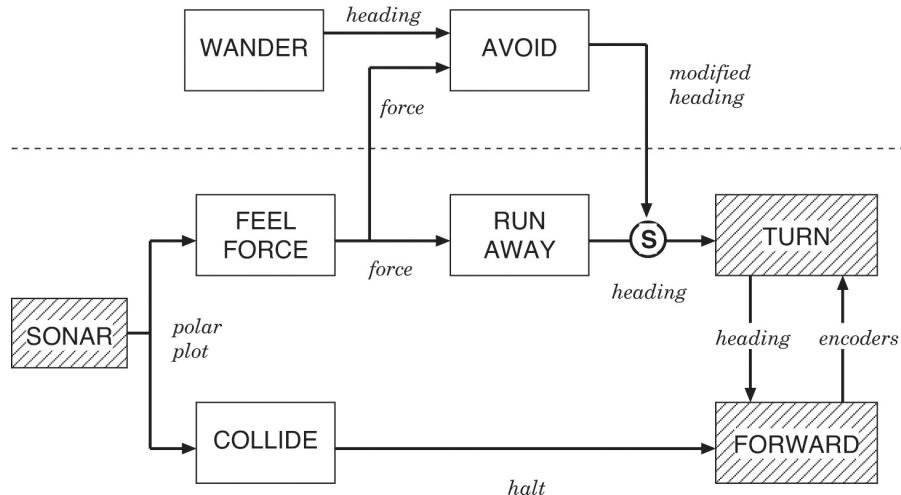


Figure 8.16 Level 1: wander.

The issue now is when to accept the heading vector from which layer. Subsumption makes it simple: the output from the higher level subsumes the output from the lower level. Subsumption is done in one of two ways:

INHIBITION

1. *inhibition*. In inhibition, the output of the subsuming module is connected to the output of another module. If the output of the subsuming module is “on” or has any value, the output of the subsumed module is blocked or turned “off.” Inhibition acts like a faucet, turning an output stream on and off.

SUPPRESSION

2. *suppression*. In suppression, the output of the subsuming module is connected to the input of another module. If the output of the subsuming module is on, it *replaces* the normal input to the subsumed module. Suppression acts like a switch, swapping one input stream for another.

In this case, the AVOID module suppresses (marked in the diagram with a S) the output from RUNAWAY. RUNAWAY is still executing, but its output does not go anywhere. Instead, the output from AVOID goes to TURN.

The use of layers and subsumption allows new layers to be built on top of less competent layers without modifying the lower layers. This is good software engineering, facilitating modularity and simplifying testing. It also adds some robustness in that if something should disable the Level 1

behaviors, Level 0 would remain intact. The robot would at least be able to preserve its self-defense mechanism of fleeing from approaching obstacles.

Figure 8.17 shows Level 1 recast as behaviors. Note that FEELFORCE was used by both RUNAWAY and AVOID. FEELFORCE is the perceptual component (or schema) of both behaviors, with the AVOID and RUNAWAY modules being the motor component (or schema). As is often the case, behaviors are named after the observable action. This means that the behavior (which consists of perception and action) and the action component have the same name. The figure does not show that the AVOID and RUNAWAY behaviors share the same FEELFORCE perceptual schema. As will be seen in the next chapter, the object-oriented properties of schema theory facilitate the reuse and sharing of perceptual and motor components.

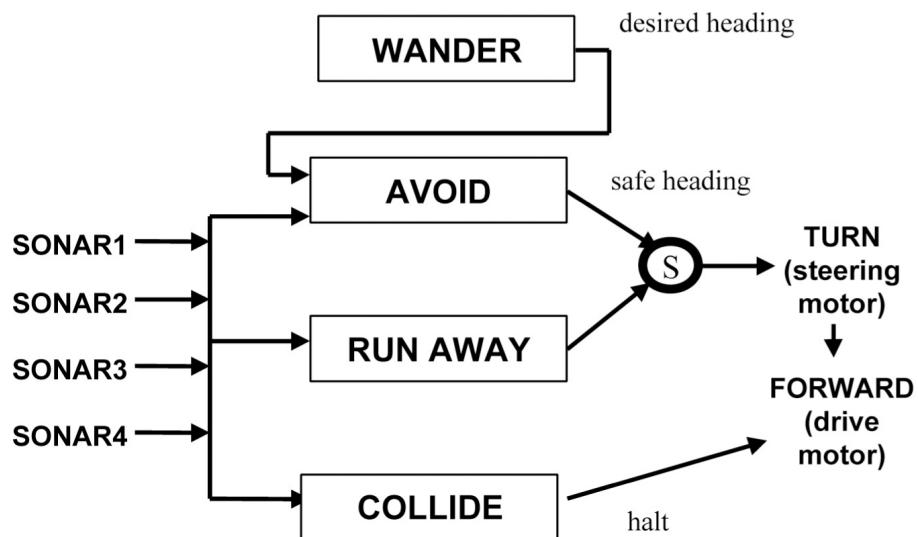


Figure 8.17 Level 1 recast as primitive behaviors.

LEVEL 2: FOLLOW CORRIDORS

Now consider adding a third layer to permit the robot to move down corridors as shown in figure 8.18. (The third layer in Brooks' original paper is “explore” because he was considering a mapping task.) The LOOK module examines the sonar polar plot and identifies a corridor. (Note that this is another example of behaviors sharing the same sensor data but using it locally for different purposes.) Because identifying a corridor is more computationally expensive than just extracting range data, LOOK may take longer to run than would behaviors at lower levels. LOOK passes the vector representing the direction to the middle of the corridor to the STAYINMIDDLE module. STAYINMIDDLE subsumes the WANDER module and delivers its output to the AVOID module, which can then swerve around obstacles.

But how does the robot get back on course if the LOOK module has not computed a new direction? In this case, the INTEGRATE module has been observing the robot's actual motions from shaft encoders in the actuators. This gives an estimate of how far off course the robot has traveled since the last update by LOOK. STAYINMIDDLE can use the dead reckoning data with the intended course to compute the new

course vector and serves to fill in the gaps in mismatches between update rates of the different modules. Notice that LOOK and STAYINMIDDLE are quite sophisticated from a software perspective.

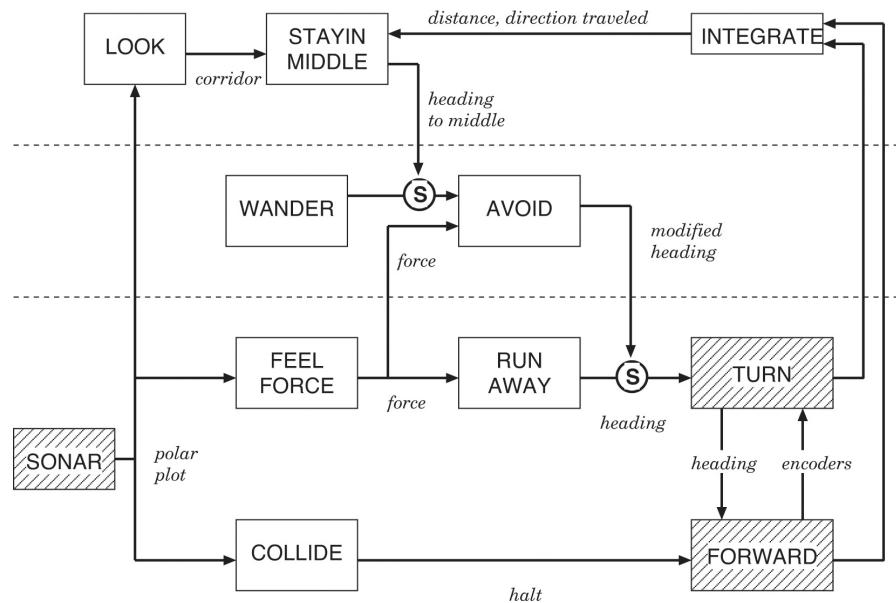


Figure 8.18 Level 2: follow corridors.

INTEGRATE is an example of a module which is supplying a dangerous internal state: it is actually substituting for feedback from the real world. If, for some reason, the LOOK module never updates, then the robot could operate without any sensor data forever. Or at least until it crashed! Therefore, subsumption-style systems include time constants on suppression and inhibition. If the suppression from STAYINMIDDLE ran for longer than n seconds without a new update, the suppression would cease. The robot would then begin to wander, and hopefully whatever problem (like the corridor being totally blocked) that had led to the loss of signal would fix itself.

But how does the robot know that it has not started going down the hallway it just came up? Answer: it does not. The design assumes that that a corridor will always be present in the robot's ecological niche. If it is not, the robot does not behave as intended. This is an example of the connotation that reactive systems are “memoryless.”

To summarize subsumption:

- Subsumption defines behavior loosely as a tight coupling of sensing and acting. Although subsumption is not a schema-theoretic architecture, it can be described in those terms. It groups schema-like modules into layers of competence, or abstract behaviors.
- Higher layers may subsume and inhibit behaviors in lower layers, but behaviors in lower layers are never rewritten or replaced. From a programming standpoint, this may seem strange. However, it mimics biological evolution. Recall that the fleeing behavior in frogs (chapter 7) was actually the result of two behaviors, one which always moved toward moving objects and the other which

actually suppressed that behavior when the object was large.

- The design of layers and component behaviors for a subsumption implementation, as with all behavioral design, is hard; it is more of an art than a science. This is true for all reactive architectures.
- There is nothing resembling a STRIPS-like plan in subsumption. Instead, behaviors are released by the presence of stimulus in the environment.
- Subsumption solves the frame problem by eliminating the need to model the world. It also does not have to worry about the open world being non-monotonic and having some sort of truth maintenance mechanism because the behaviors do not remember the past. There may be some perceptual persistence leading to a fixed-action pattern type of behavior (e.g., corridor following), but there is no mechanism which monitors for changes in the environment. The behaviors simply respond to whatever stimulus is in the environment.
- Perception is largely direct and uses affordances. The releaser for a behavior is almost always the percept for guiding the motor schema.
- Perception is egocentric and distributed. In the wander (layer 2) example, the sonar polar plot was relative to the robot. A new polar plot was created with each update of the sensors. The polar plot was also available to any process which needed it (shared global memory), allowing user modules to be distributed. Output from perceptual schemas could be shared with other layers.

8.5 Sequences: Finite State Automata

Potential fields and subsumption are good for handling behaviors that are independent. However, some behaviors form a larger abstract behavior where the interactions may be more nuanced. In these cases, a finite state automata (FSA) or a *script* [68:72;117;150](#) or a related construct called *skills*,[77;76](#) might be a good choice; the three are equivalent but reflect different programming styles. This section discusses FSA because readers should be familiar with this common computing structure.

FINITE STATE AUTOMATA

Finite state automata (FSA) is a set of popular mechanisms for specifying what a program should be doing at a given time or circumstance. The FSA is generally written as a table accompanied by a *state diagram*, giving the designer a visual representation. Most designers do both. There are many variants of FSA, but each works about the same way. This section follows the notation used in algorithm development.[116](#)

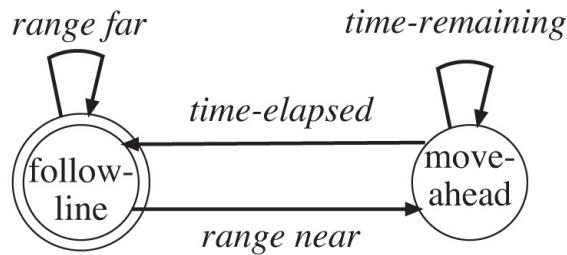
8.5.1 A Follow the Road FSA

This section will use the Colorado School of Mines' entry in the 1994 Unmanned Ground Robotics (UGR) competition as an example; the competition and the entry are detailed in chapter 19. The general idea was for a golf-cart sized robot to autonomously drive on a “road” that was a set of white lines painted on green grass. Bales of hay were placed on the edges of the “road,” which covered up the white line. The CSM entry either saw a white line with its camera and followed the line or detected a bale with a sonar and went straight until it had passed the bale.

FSA STATES

FSA START STATE

Regardless of application, the designer begins by specifying a finite number of discrete *states* that the robot should be in. The set of states is represented by K , and each state, $q \in K$, is a listing of the behaviors that should be active at the same time. In the case of the UGR competition, the CSM entry design had two possible states: turning to follow the line and moving forward. States are represented in a table under the heading q and by circles in a graph. (See [figure 8.19](#).) By convention, there is always a *Start state*, and the robot would always start there. The Start state is often written as s or q_0 and drawn with a double circle. In the case of the UGR entry, the following-line state was the start state since the robot always starts with the follow-line behavior active and not suppressed.



a.

$$M : K = \{\text{follow-line}, \text{move-ahead}\}, \Sigma = \{\text{range near}, \text{range far}\}, \\ s = \text{follow-line}, F = \{\text{follow-line}, \text{move-ahead}\}$$

q	σ	$\delta(q, \sigma)$
follow-line	range near	move-ahead
follow-line	range far	follow-line
move-ahead	time remaining	move-ahead
move-ahead	time elapsed	follow-line

b.

Figure 8.19 A FSA representation of the coordination and control of behaviors in the UGV competition: a.) a diagram and b.) the table.

FSA INPUTS

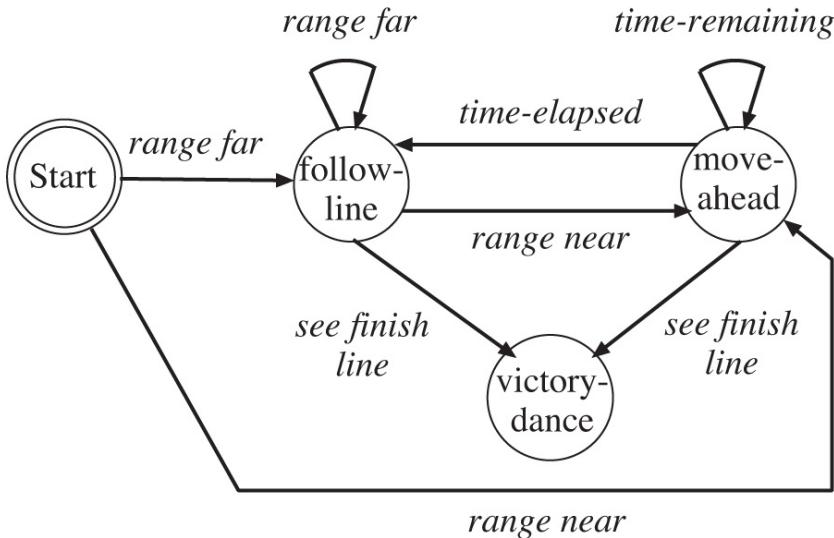
The next part of the FSA is the set of *inputs* (also called the alphabet). Inputs are the behavioral releasers and appear under the column heading σ . Unlike the IRM diagrams, the FSA table considers what happens to each state q for all possible inputs. As shown in [figure 8.19](#), there are only two releasers for the UGV example, so the table does not have many rows.

FSA TRANSITION FUNCTIONS

The third part of the FSA is the *transition function*, called δ , which specifies what state the robot is in after it encounters an input stimulus, σ . The set of stimuli or affordances σ that can be recognized by the robot is represented by Σ . These stimuli are represented by arrows. Each arrow represents the *releaser* for a behavior. The new behavior, triggered by the releaser, depends on the state the robot is in. This triggering by a releaser is the same as with Innate Releasing Mechanisms, where the animal literally ignores releasers that are not relevant to its current state.

Recall also in the serial program implementations of IRMs in chapter 7 that the agent “noticed” releasers every second. At one iteration through the loop, the agent might be hungry and “enter” the state of feeding. In the next iteration, it might still be hungry and re-enter the state of feeding. This can be represented by having arrows starting at the feeding state and pointing back to the feeding state.

For the example in [figure 8.19](#), the robot starts in the state of following a line. This means that the robot is not prepared to handle a visual distraction (indicated by the sonar sensing range `near`) until it has started following a line. If it does, the program may fail because the FSA clearly shows it will not respond to the distraction for at least one update cycle. By that time, the robot may be heading in the wrong direction. Starting in the `follow-line` state was fine for the UGR competition, where it was known in advance that there were no bales of hay near the starting line. A more general case is shown in [figure 8.20](#), where the robot can start either on a clear path or in the presence of a bale. The FSA does not make it clear that if the robot starts by a bale, it had better be pointed straight down the path!



a.

$$M: K = \{\text{follow-line}, \text{move-ahead}\}, \Sigma = \{\text{range near}, \text{range far}\}, \\ s = \text{follow-line}, F = \{\text{follow-line}, \text{move-ahead}\}$$

q	σ	$\delta(q, \sigma)$
follow-line	range near	move-ahead
follow-line	range far	follow-line
move-ahead	time remaining	move-ahead
move-ahead	time elapsed	follow-line

b.

Figure 8.20 An alternative FSA representation of the coordination and control of behaviors in the UGV competition:
a.) a diagram and b.) the table.

FSA FINAL STATE

The fourth piece of information that a designer needs to know is when the robot has completed the task. Each state that the robot can reach that terminates the task is a member of the set of *final state*, F . In the UGR example, the robot is never done and there is no final state—the robot runs until it is turned off manually or runs out of power. Thus, both states are final states. If the robot could recognize the finish line, then it could have a finish state. The finish state could be any state that was active at the time or it could be another behavior, such as a wag of the camera creating a *victory-dance*. Notice that another behavior adds more rows to the FSA table, since there must be one row for each unique

state.

The resulting table is known as a finite state machine and abbreviated M for machine. The notation:

$$M = \{K, \Sigma, \delta, s, F\}$$

is used as reminder that, in order to use a FSA, the designer must know all the q states (K), the inputs σ , the transitions between the states δ , the special starting state q_0 , and the terminating state(s) (F). There must be one arrow in the state diagram for every row in the table. [Table 8.1](#) summarizes the relationship of FSA to behaviors.

Table 8.1 Relationship of finite state automata to behaviors.

FSA	Behavioral analog
K	all the behaviors for a task
Σ	the releasers for each behavior in K
δ	the function that computes the new state
s	the behavior the robot starts in when turned on
F	the behavior(s) the robot must reach to terminate

In more complex domains, robots need to avoid obstacles (especially people). Avoidance should always be active, so it is often implicit in the FSA. move-to-goal often is shorthand for move-to-goal *and* avoid. This implicit grouping of “interesting task-related behaviors” and “those other behaviors which protect the robot” will be revisited as strategic and tactical behaviors.

Another important point about using FSA is that they describe the overall behavior of a system, but the implementation may vary. [Figure 8.19](#) is an accurate description of the state changes in the UGV entry. The control for the behaviors could have been implemented exactly as indicated by the FSA: if follow-line is active and range returns range near, then move-ahead. However, due to timing considerations, the entry was programmed differently, though with the same result.

8.5.2 A Pick Up the Trash FSA

As another example of how to construct and apply an FSA, consider the American Association for Artificial Intelligence 1996 Pick Up the Trash competition. Assume that the robot is a small vehicle, such as the ones shown in [figure 8.21](#), which were used by the winning Georgia Institute of Technology’s team. Each robot has a camera for finding trash and a bumper switch to tell when the robot has collided with something. The robot has a simple gripper. Assume that the robot can tell if the gripper is empty or full. (One way to do this is to put an IR sensor across the jaw of the gripper. When the IR sensor returns a short range, the gripper is full, and it can immediately close with a grasping reflex.) One problem with grippers is that they are not as good as a human hand; there is always the possibility that the can will slip or fall out of the gripper. Therefore the robot should respond appropriately when it is carrying a can or trash and loses it.

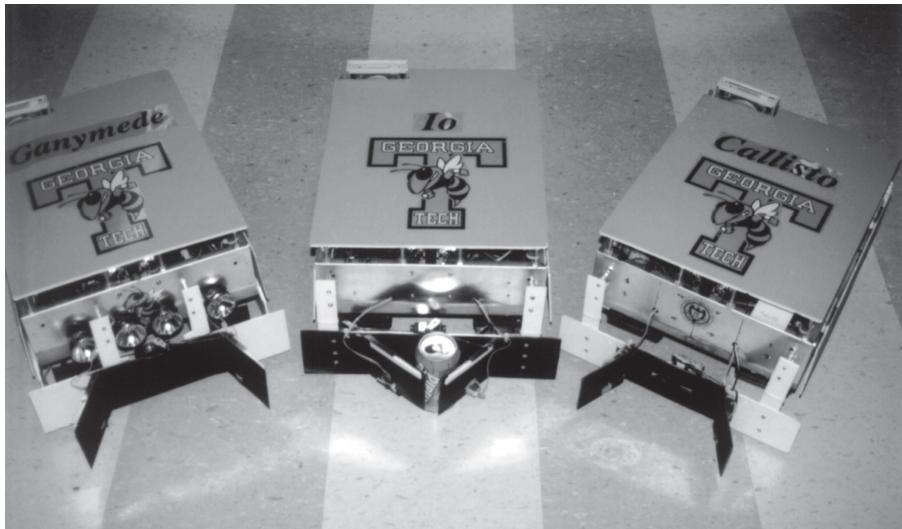


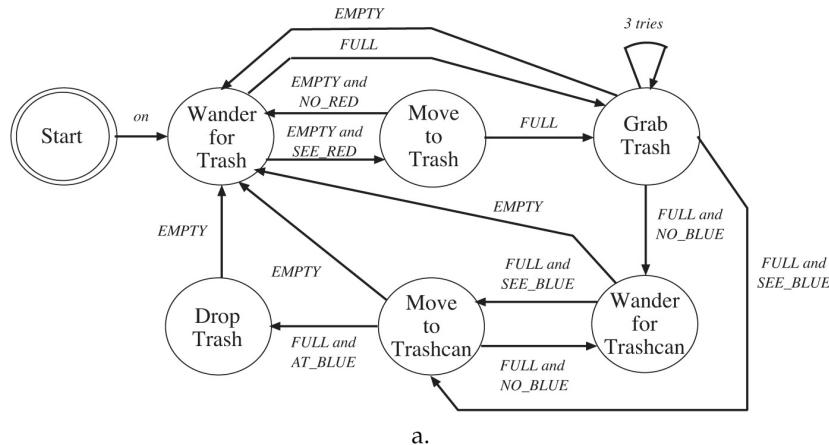
Figure 8.21 Georgia Tech’s award-winning Pick Up the Trash robots with the trash gripper (photograph courtesy of Tucker Balch and AAAI).

The Pick Up the Trash environment is visually simple, and there are obvious affordances. Coca-Cola® cans are the only red objects in the environment, and two blue recycling trash cans are the only blue objects. Therefore visually extracting red and blue blobs should be sufficient. All objects are on the floor, so the robot only has to worry about where the objects are on the X axis. A basic scenario is for the robot to start wandering around the arena looking for red blobs. It should head straight for the center of largest red blob until the can is in the gripper. Then it should try three times to grab the can, and, if successful, it should begin to wander around looking for a blue blob. There should only be one blue blob in the image at a time because the two trash cans are placed in opposite corners of the arena. Once it sees a blue blob, the robot should move straight to the center of the blob until the blob gets to be a certain size in the image (looming). The robot should stop, let go of the can, turn around to a random direction, and resume the cycle. The robot should avoid obstacles; therefore moving to a red or blue blob should be a fixed-action pattern type of behavior, instead of the robot immediately forgetting where it was heading.

Notice that the design relies on the gripper to maintain what step in the nominal sequence the robot is in. An empty gripper means the robot should be in the collecting the trash phase, either looking for a can or moving toward one. A full gripper means the robot is in the deposit phase. The looming releaser extracts the size of the blue region in pixels and compares it to the size N . If the region is greater than, or equal to, N , then the robot is “close enough” to the trash can, and the robot can drop the can.

Figure 8.22 shows a FSA for the Pick Up the Trash task. The FSA allows the designer to tinker with the sequence and represent the behavioral design graphically. However, it still does not show how the sequence would be implemented and thus can mislead the designer into creating extra internal states and behaviors. The FSA has two two wander states: `wander for trash` and `wander for trashcan`. However, these are really the same wander behavior, with each version instantiated by different releasers and terminated under different conditions. There are also two variations of the same `move-to`

behavior.



a.

$$K = \{\text{wander for trash, move to trash, grab trash, wander for trash can, move to trash can, drop trash}\}, \Sigma = \{\text{on, EMPTY, FULL, SEE_RED, NO_BLUE, SEE_BLUE, AT_BLUE}\}, s = \text{Start}, F = K$$

q	σ	$\delta(q, \sigma)$
start	on	wander for trash
wander for trash	EMPTY, SEE_RED	move to trash
wander for trash	FULL	grab trash
move to trash	FULL	grab trash
move to trash	EMPTY, NO_RED	wander for trash
grab trash	FULL, NO_BLUE	wander for trash can
grab trash	FULL, SEE_BLUE	move to trash can
grab trash	EMPTY	wander for trash
wander for trash can	EMPTY	wander for trash
wander for trash can	FULL, SEE_BLUE	move to trash can
move to trash can	EMPTY	wander for trash
move to trash can	FULL, AT_BLUE	drop trash
drop trash	EMPTY	wander for trash

b.

Figure 8.22 A FSA for the Pick Up the Trash task: a.) state diagram, and b.) table showing state transitions.

Another common mistake is to assume the FSA carries forward previous releasers. For example, the correct transition from **Grab Trash** to **Wander For Trash** can is **FULL** and **NO_BLUE**, but a designer may be tempted to label the arrow as only **NO_BLUE**, since, to get that state, the gripper had to be **FULL**. This is a very dangerous mistake because it assumes that the implementation will be keeping up with what internal state the robot is in (by setting a variable), instead of letting directly perceivable attributes of the world inform the robot as to whichever state it is in. Internal state is incompatible with reactivity.

8.6 Sequences: Scripts

SCRIPTS

SKILLS

SUB-SCRIPT

Abstract behaviors often use *scripts*,^{68;72;117;150} or a related construct called *skills*,^{77;76} to create generic templates of assemblages of behaviors. Scripts provide a different way of generating the logic for an assemblage of behaviors. They encourage the designer to think of the robot and the task in terms of a screenplay. Scripts were originally developed for natural language processing (NLP) to help the audience (a computer) understand actors (people talking to the computer or writing summaries of what they did).¹⁸⁷ In the case of robots, scripts can be used more literally, where the actors are robots reading the script. The script has more room for improvisation, though, should the robot encounter an unexpected condition (an exception), the robot begins following a *sub-script*.

CAUSAL CHAIN

Figure 8.23 shows how elements of an actor's script compares to a robot script. The main sequence of events is called a *causal chain*. The causal chain is critical because it embodies the coordination control program logic just as a FSA does. It can be implemented in the same way. In NLP, scripts allow the computer to keep up with a conversation that may be abbreviated. For example, consider a computer trying to read and translate a book where the main character has stopped in a restaurant. Good writers often eliminate all the details of an event to concentrate on the ones that matter. This missing, but implied, information is easy to extract. Suppose the book started with "John ordered lobster." This is a clue that serves as an index into the current or relevant event of the script (eating at a restaurant), skipping over past events (John arrived at the restaurant, John got a menu, etc.). Clues also focus the system's attention on the next likely event (look for a phrase that indicates John has placed an order), so the computer can instantiate the function which looks for this event. If the next sentence is "Armand brought out the lobster and refilled the wine glass," the computer can infer that Armand is the waiter and that John had previously ordered and received wine.

Script	Collection of Behaviors	Example
Goal	Task	pick up and throw away a Coca-Cola can
Places	Applicability	an empty arena
Actors	Behaviors	WANDER_FOR_GOAL, MOVE_TO_GOAL, GRAB_TRASH, DROP_TRASH
Props, Cues	Percepts	red, blue
Causal Chain	Sequence of Behaviors	WANDER_FOR_GOAL(TRASH), MOVE_TO_GOAL(TRASH), GRAB_TRASH, WANDER_FOR_GOAL(TRASH CAN), MOVE_TO_GOAL(TRASH CAN), DROP_TRASH
Sub-scripts	Exception Handling	if have trash and drop, try GRAB_TRASH three times

Figure 8.23 Comparison of script structures to behaviors.

INDEXING

FOCUS-OF-ATTENTION

When programming robots, people often like to abbreviate the routine portions of control and

concentrate on representing and debugging the important sequences of events. Finite state automata force the designer to consider and enumerate every possible transition, while scripts simplify the specification. The concepts of *indexing* and *focus-of-attention* are extremely valuable for coordinating behaviors in robots in an efficient and intuitive manner. Effective implementations require asynchronous processing; therefore the implementation is beyond the scope of this book. For example, suppose a Pick Up the Trash robot boots up. The first action on the causal chain is to look for the Coke cans. The designer though realizes that this behavior could generate a random direction and move the robot, missing a can right in front of it. Therefore, the designer wants the code to permit the robot to skip searching the arena if it immediately sees a Coke can, and begin to pick up the can without even calling the `wander-for-goal(red)` behavior. The designer also knows that the next releaser after `grab-trash` exits is to look for is blue, because the cue for moving to the trashcan and dropping off trash is blue.

The resulting script for an abstract behavior to accomplish a task is usually the same as the programming logic derived from an FSA. In the case of Pick Up the Trash, the script might look like:

```

for each update...
\\ look for props and cues first: cans, trash cans, gripper
rStatus=extract_color(red, rcx, rSize); \\ ignore rSize
if (rStatus==TRUE)
    SEE_RED=TRUE;
else
    SEE_RED=FALSE;
bStatus=extract_color(blue, bcx, bSize);
if (bStatus==TRUE){
    SEE_BLUE=TRUE; NO_BLUE=FALSE;
} else {
    SEE_BLUE=FALSE; NO_BLUE=TRUE;
}
AT_BLUE=looming(size, bSize);
gStatus=gripper_status();
if (gStatus==TRUE) {
    FULL=TRUE; EMPTY=FALSE;
} else {
    FULL=FALSE; EMPTY=TRUE;
}
\\index into the correct step in the causal chain
if (EMPTY){
    if (SEE_RED){
        move_to_goal(red);
    else
        wander();
    } else{
    % grab_trash();
    % if (NO_BLUE)
    %     wander();
    % else if (AT_BLUE)
    %     drop_trash();
    % else if (SEE_BLUE)
    %     move_to_goal(blue);
    %
}

```

8.7 AI and Behavior Coordination

Behavioral coordination, at first glance, appears to be the purview of deliberation and would engage the areas of planning and problem solving, inference, and search. However, behavioral coordination is an element within the reactive functionality of an intelligent robot. It perhaps draws only upon the knowledge representation from the seven key areas of artificial intelligence. One such knowledge representation is scripts, introduced in the previous section.

As this chapter has shown, a behavioral system can be implemented by various technical architectures of which most have very little deliberation yet caused great controversy in the early AI robotics community. Early technical architectures committed to a single class of knowledge representations and coordination algorithms and, as a result, often are referred to by that algorithm:

“pfield architecture,” “subsumption architecture,” “fuzzy logic architecture,” and so forth. This is confusing because instances of subsumption used vectors as its knowledge representation though it did not use vector summation as the sole coordination mechanism. More modern implementations do not ascribe to the same religious commitment to a single mechanism.

8.8 Summary

The coordination function maps a set of behaviors, each of which may have an output action with an associated gain, onto a single action: $\rho = C(G \times B(S))$. The multiple behaviors B that are being coordinated may be activated concurrently or sequentially.

Two different approaches to producing a single action from multiple concurrent behaviors are: cooperating and competing. The most popular cooperating method is a potential fields methodology. In a potential fields methodology, behaviors produce a vector by representing the relationship between stimulus and response over space as a “force” field. Primitive pfields may be generated and combined within a single behavior to generate the output of a behavior. Pfield methodologies are unfortunately avoided because of a fear of local minima or maxima, even though simple mechanisms exist which prevent these conditions. Fuzzy logic is another popular methodology for coordinating cooperating behaviors.

Competitive methods work by arbitration, usually producing a single behavior as the winner. The subsumption architecture is the most popular. It creates layers of behaviors where higher levels subsume lower levels via suppression or inhibition. It is more oriented towards hardware. Voting is another common competing form of arbitration, but voting methods are often brittle and not necessarily extensible. They converge to fuzzy methods, and thus it may be more practical to implement a fuzzy method given the flexibility of those methods.

Sequences can be generated explicitly through either finite state automata (FSA) or scripts and skills. FSA is a classic computing structure that should be familiar to most coders. Scripts and skills are drawn from artificial intelligence and may be less familiar.

Returning to questions posed in the overview. *What happens when several behaviors are released (instantiated) at the same time?* It depends on the coordinated function whether the behaviors will be blended or arbitrated. *What about sequences of behaviors?* Sequences were implicit through IRM and reactive behaviors but can be explicitly designed through “skills” represented by finite state automata and scripts (production rules are discouraged). The words “reactive” and “explicit” may seem at odds with other leading to *How can you do this and avoid planning, explicit representations of state or the world?* But behavioral coordination does not require deliberation. Affordances can serve to trigger implicit sequences while gains, inhibition, suppression, and homeostasis can also serve to implicitly modulate a sequence. Finite state automata, scripts, fuzzy logic, and voting are techniques that are more explicit but are not deliberative or require a world model.

This chapter concludes the introduction to the biological theory behind reactive functionality. The next two chapters will introduce locomotion and sensing, concepts needed to make a robot actually work.

8.9 Exercises

Exercise 8.1

Write out the formula for the coordination function and define each component.

Exercise 8.2

What does G stand for in the coordination function $\rho = C(G * B(S))$?

Exercise 8.3

Define each method of behavioral coordination and give at least one example methodology:

- a. cooperating
- b. competing
- c. sequencing.

Exercise 8.4

List three limitations of potential fields for behavioral coordination and their possible solutions.

Exercise 8.5

Consider that a UAV blocking an exit consists of a helicopter body with a camera that can pan and tilt independent of the body (i.e., the body can stay in one place while the camera moves). The UAV is programmed to have three behaviors: watching, approaching, and threatening. In watching, the UAV is stationary at above eye level, but the camera scans the area in front. In approaching, the UAV moves to eye level, and the camera tracks the center of the crowd. In threatening, the UAV performs random motions if the crowd gets too close. The UAV has the following motor schemas (with arguments missing for brevity): `fly-at-given-altitude()`, `body-turn()`, `body-move-sideways()`, `body-move-random-3D()`, and `camera-pan-tilt()`. It has the following perceptual schemas: `detect-motion()` and `detect-object-distance()`. Design the behavioral scheme using:

- a. potential fields methodology.
- b. subsumption architecture methodology.

Exercise 8.6

Create a FSA to coordinate behaviors for a can-recycling robot with a vision. Assume the robot starts with `GRIPPER=EMPTY`. The robot searches for red Coke cans, `RED_BLOB`, using a `SEARCH` behavior. When it sees a red can, it uses a `MOVE_TO_CAN` behavior to move to the can until it detects `AT_CAN`. It then grasps the can using the `GRASP` behavior and generates `EMIT_BEEP` to signal that it is done. Assume that once the `MOVE_TO_CAN` behavior sees a can, the robot will never lose sight of that can, though it may lose the grip on the can. You can only use capitalized terms as states and inputs.

- a. Draw the state diagram. In order to keep the diagram from being cluttered, do not show transitions that do not change behaviors. Remember to label all circles and arrows.
- b. Write out the transition table. Label the columns with the correct Greek symbols.

Exercise 8.7

Consider a robot with two vision sensors mounted on each side (like a frog). The robot searches for red Coke cans among stacks of white newspapers. When it sees a red can, it moves to and grasps the can, then begins searching for a blue recycling bin. When it sees the bin, the robot moves to the bin and deposits the can. If it sees a white bundle, the robot avoids it. The robot repeats the cycle ad infinitum. Ignoring the grasping and depositing actions, describe the motor schema for each behavior in terms of the five primitive potential fields and a random field. Specify the dropoff function and any gains.

Exercise 8.8

Consider the Pick Up the Trash example in section 8.5.2. The example assumed the arena was empty except for walls, cans, and trash cans. What would happen if there were chairs and tables present? Could the gripper accidentally grasp a

chair or table leg? How would the system react? What changes, if any, would need to be made to the behavior table and FSA?

Exercise 8.9

Describe the two methods for assembling primitive behaviors into abstract behaviors: finite state automata and scripts.

[*World Wide Web*]

Exercise 8.10

Identify at least five robot competitions and, for each, describe the basic task. Can the task be accomplished with a reactive robot? Why or why not?

[*Programming*]

Exercise 8.11

Write a script in pseudocode for following a hallway. Consider that the robot may start in a hallway intersection facing away from the hallway it is supposed to take. Also the robot might start in the middle of a hall facing a wall, rather than pointing forward along the long axis of the hall.

[*Programming*]

Exercise 8.12

The Pick Up the Trash competitions were a bit different than actually presented in this book. For example, the robots were permitted to cache up to three cans before going to the trash can, and the robots could go after white Styrofoam cups as trash. How would you integrate this into:

- a. an FSA
- b. a script.

[*Programming*]

Exercise 8.13

Could the exception handling sub-scripts for picking up trash be implemented with the exception handling functionality provided by Ada or C++?

[*Advanced Reading*]

Exercise 8.14

Read Rodney Brooks' one paragraph introduction to chapter 1 in *Cambrian Intelligence*³⁰ on the lack of mathematics in behavioral robotics. Now consider the behavior of the CSM robot around white shoes and dandelions. Certainly it would be useful to have theories which can prove when a behavioral system will fail. Is it possible?

8.10 End Notes

For the roboticist's bookshelf.

The two books that cover the fundamentals of behavioral coordination are Arkin's Behavior-Based Robotics¹¹ and Brooks' Cambrian Intelligence.³⁰ Arkin's book provides a broader overview of the field, though concentrates on potential fields, while Brooks' book serves as the reference for the subsumption architecture.

“Fast, Cheap and Out of Control: the movie.”

The term “Fast, Cheap and Out of Control” became the title of a 1997 award-winning documentary by Errol Morris about four different men, including Rodney Brooks. The movie title implied that Morris saw humankind shifting from highly individualistic relations with the world, developed over time, (as seen by the lion tamer and topiary gardener) to decentralized, reactive mobs (as seen by Brooks' insect robots). Although the movie is not about robotics per se, it features interviews with Brooks and contains stunning shots of some of Brooks' robots walking over broken glass,

shining like diamonds in the bright lights. Maja Mataric, one of Brooks' students at the time of the filming and now a dean at the University of South California, can be seen in one of the shots wearing shorts.

Reusability principle of software engineering and beer cans.

While researchers worried about picking up and disposing of soda cans, a popular magazine began discussing what it would take to have a robot go to the refrigerator and bring the user a beer. In an apparent acknowledgment of the lifestyle of someone who would have such a robot, the robot was expected to be able to recognize and avoid dirty underwater lying at random locations on the floor.

The 1995 IJCAI Mobile Robot Competition.

The competition was held in Montreal, Canada, and many competitors experienced delays receiving their robots due to delays at Customs. The University of New Mexico team spent the preparatory days of the competition frantically trying to locate their robot. Almost at midnight, the evening before the preliminary rounds were to begin, a forklift drove up with the crated robot. All the teams looked up from their frantic testing and cheered in support of the New Mexicans until the forklift came close enough and everyone could clearly see that the "This Side Up" arrow was pointing down... The New Mexicans did not even bother to uncrate the robot to catalog the damage, they instead sought solace in Montreal's wonderful night life. The teams with robots properly aligned with gravity went back to programming and fine tuning their entries.

Cheering at robot competitions.

The chant "Core Dump. Core Dump. Segmentation Fault!" has a nice cadence and is especially appropriate to yell at competitors using Unix systems.

subsumption and Soda Cans.

Jon Connell addressed this task in 1989 as his thesis work at MIT,⁵⁰ applying subsumption to a robot arm, not a set of flappers. He used a special type of FSA called an Augmented Finite State Machines (AFSMs) and over 40 behaviors to accomplish the task.

Being in a Pick Up the Trash competition without a manipulator.

As often happens, robot competitions often pose problems that are a step beyond the capability of current hardware and software technology. In 1995, arms on mobile robots were rare; indeed Nomad introduced a forklift arm just in time for the IJCAI Mobile Robot Competition. Participants, such as the Colorado School of Mines with an older robot and no arm, could have a "virtual manipulator" with a point deduction. The robot would move to within an agreed tolerance of the object, then either play a sound file or make a noise. The virtual manipulator—a human team member, either Tyler Devore, Dale Hawkins, or Jake Sprouse—would then either pick up the trash and place it on the robot or remove the trash. It made for an odd reversal of roles: the robot appeared to be the master and the student, the servant!

About grippers maintaining the state of world.

The Pick Up the Trash event mutated in 1996 to picking up tennis balls in an empty arena and, in 1997, into a variation on sampling Mars. For the 1997 Find Life on Mars event, the sponsors brought in real rocks, painted black to contrast with the gray concrete floor, and blue, green, red, and yellow toy blocks or "Martians." Because of weight considerations in shipping the rocks, the rocks were about the size of a couple of textbooks and not that much bigger than the Martians. One team's purely reactive robot had trouble distinguishing colors during preliminary rounds. It would misidentify a rock as a Martian during a random search, navigate to it, grip it, and then attempt to lift it. Since the rock was heavy, the gripper could not reach the full extension and trigger the next behavior. It would stay there, clutching the rock. Sometimes the robot would grip a rock and the gripper would slip. The robot would then try to grip the rock two more times. Each time the gripper would slip. The robot would give up, then resume a random search, and unfortunately, the search seemed invariably to direct the robot back to the same rock, where the cycle would repeat itself. Eventually the judges would go over and move the robot to another location.

9

Locomotion

Chapter Objectives:

- Describe the difference between *Ackerman* and *differential steering*.
- Describe the difference between *holonomic* and *nonholonomic* vehicles.
- Rank the following list of locomotion types in order of power demands: crawling/sliding, running, tires on soft ground, walking, and railway wheels.
- Given the number of legs, give the formula and compute the number of possible leg events.
- Describe the difference between *static* and *dynamic balance* and why *static stability* is often considered desirable.
- Define *support polygon*, *zero moment point*, and *action selection*.
- Define the role of *reference trajectory* and *central pattern generators* in locomotion.
- List three *virtual gaits*.

9.1 Overview

By this point, the reader should be familiar with principles of organizing the behavioral software for a robot system and understanding what is in the “lower brain” of the robot. The connection between what is in the lower brain and the actual movement or mobility of the robot has not been addressed. In particular, the reader may be wondering: *How do you actually make a robot move?* Considering mobility, or more precisely locomotion, of a robot for navigation, leads to a second common question: *Why don’t more robots have legs?* Existing ground robots are predominately wheeled or tracks, yet animals have legs or slither and crawl.

In general, AI is not associated with locomotion or walking research because the core concepts are in mechanics and control. Most of the enabling mechanisms for walking and swimming are related to the low-level control of the platform, just like the equations for stable flight of a UAV are not considered AI. Instead, AI research in locomotion typically focuses on generating the *reference trajectory* (where it should go) and at what velocity to execute that trajectory (when to run, not walk). The two notable exceptions to this are explorations into learning to walk, originating with Pattie Maes’

demonstrations on Ghenghis, and continuing with the exploitation of central pattern generators from the neuroethology and neural networks communities.

While AI rarely addresses locomotion, anyone working in AI robotics needs some familiarity with locomotion and its terminology. Thus this chapter provides an overview of biomimetic locomotion for ground robots, though concepts, such as pattern generators for gaits, can be applied to marine vehicles. The chapter first covers mechanical locomotion with wheels and tracks, given that those are the de facto standard mechanisms. Next it shifts to describing biomimetic forms of locomotion, or how robots can move like animals—with or without legs. Legged locomotion is introduced, dividing legged robots into those using static balance and those using dynamic balance. With the concepts of the zero moment point and gaits established, the chapter then focuses on planning oscillations for moving legs in gaits or adjusting a particular leg, with joints, to terrain. The chapter ends with a summary of how action selection and learning are alternative approaches to locomotion.

9.2 Mechanical Locomotion

In theory, ground robots can move in a variety of ways, either in some biological analog of slither, crawl, or walk, or through a purely mechanical motion, such as rolling. In practice, ground robots almost always roll using wheels or tracks. In order to control a ground robot, it is useful to understand the four ways that a wheeled or tracked robot might be steered and whether that steering method approximates holonomicity.

9.2.1 Holonomic versus Nonholonomic

HOLONOMIC

Many approaches in AI robotics assume that the robot is *holonomic*. Holonomic means that the device can be treated as a massless point capable of instantaneous turning in any direction (e.g., spin on a dime). A holonomic robot has two advantages for AI. The first is that it allows the designer to ignore the complexities involved in mechanical control of the robot. The designer does not have to worry about the robot slowing down before making a turn or having to parallel park. Second, holonomicity greatly simplifies path planning and localization, as will be seen in later challenges.

The most popular style of research robot in the 1980s and 1990s were Denning and Nomad robots (see [figure 9.1](#)). These robots approximated a holonomic vehicle in two ways. One, they emulated a point by being cylindrical, thus eliminating having to calculate whether a corner of the robot was going to hit something if the robot turned. Two, the robots had specialized wheeled designs to approximate holonomic turning and, since they did not move very fast, acceleration and velocity profiles for slowing down before stopping or making a sharp turn was not essential.



Figure 9.1 Clementine, a Denning DRV-1 robot.

NONHOLONOMIC

In reality, robots are *nonholonomic*. Robots have mass, and, no matter how clever the wheel design, there is always some *skitter* where the robot crabs to one side when it turns. Applying AI to autonomous cars also requires accepting that they are nonholonomic. Because the robots are nonholonomic, the reactive or deliberative functions will need to know how fast the robot is moving and how far. Generally robots will have shaft or optical encoders to count shaft or wheel turns as described in chapter 10. However, it is useful to keep in mind that shaft and optical encoders work reasonably well for estimating the actual distance travel for wheeled vehicles, but not for tracked vehicles, because of errors introduced by the different friction and contact surfaces generated by wheels

and tracks. Also rotational movements with either tracked or wheeled vehicles accrue errors more rapidly than do translational movements.

9.2.2 Steering

Since all robots, and especially ground robots, are nonholonomic, it is important to understand how they are steered. This section uses the taxonomy offered by Shamah.¹⁹⁰

There are two major styles of wheeled designs that approximate holonomic turning for benign conditions. Originally, these designs were research-oriented or limited to a highly specialized application. Now, they are beginning to show up in some real-world applications, such as the Roomba and telepresence robots that work on level floors. The two styles are:

- *Synchro-drive*, where the wheels must turn together, approximating turning in place, though there is some slippage of the wheels on the floor. The wheels are linked through either a drive belt or a set of gears. The wheels tend to be narrow to minimize contact with the ground and thus they skitter.
- Omnidirectional wheels, also called Mercurum wheels, are wheels with rollers that allow the robot to move perpendicular to the main wheel (see figure 9.2). By adjusting the relative speeds of the forward and lateral wheels, the robot can go sideways. Unfortunately, these wheels tend to be easily clogged with dirt and gravel in outdoor applications.

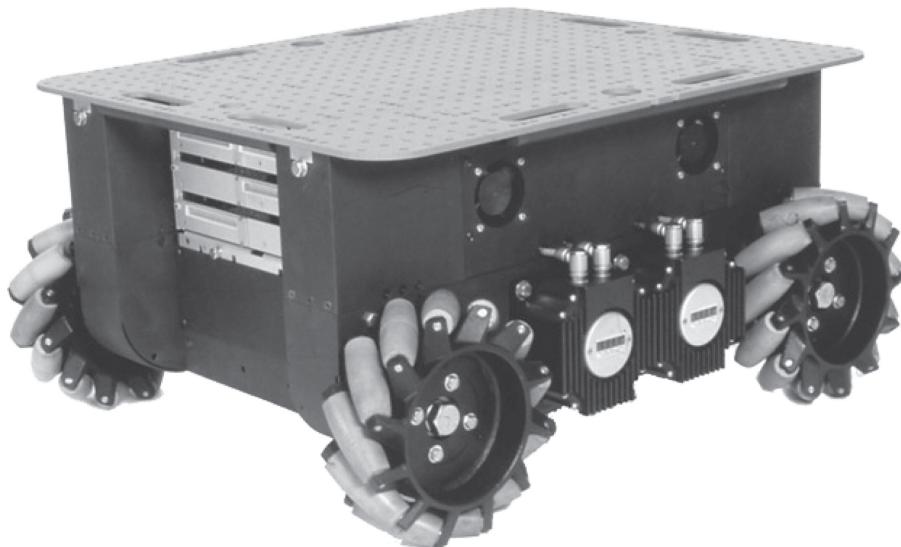


Figure 9.2 The Uranus robot with omnidirectional wheels (courtesy of Gwpcmu - Own work, CC BY 3.0, <https://commons.wikimedia.org/w/index.php?curid=11440618>).

There are also two major styles of steering for nonholonomic vehicles, and these are used extensively in real-world applications, such as bomb-squad robots, autonomous cars, and service robots. The two styles are described below:

- *Ackerman steering* is how a car is steered; the wheels in front turn to guide the vehicle. Technically Ackerman steering refers to the mechanism that adjusts the wheels so that the wheel on the inside of the turn is rotated more than the wheel on the outside of the turn in order to compensate for slight difference between the inside and outside wheel diameters. It is truly nonholonomic, considering how hard it is to parallel park.

SKID STEERING

- *skid steering*, also known as differential steering, is how a tank or bulldozer is steered. The tracks on each side can be controlled independently. If the robot is to turn left, the left track slows down while the right track speeds up. Skid steering is not restricted to tracks; it can be implemented on wheeled vehicles. Skid steering can allow the vehicle to approximate holonomicity but how closely depends on many variables, such as favorable surfaces and power of the platform.

The use of castors as a variant of skid-steering merits additional discussion and a warning. Wheelchairs and carts are often steered with two wheel skid-steering and the rest of the platform is on castors. Castors serve to support the distributed weight of the platform. The wheels of a wheelchair are differentially pushed by a person and there are castors under the legs to keep the wheelchair from falling forward. The problem is that castors do not provide any resistance or direction. As the person steers the wheels to go in one direction, the castors in front may turn in another direction, creating situations that are difficult to control. Early robotics attempted to use motorized wheelchairs as research platforms and discovered that these were very hard to control.

POLYMORPHIC SKID SYSTEMS

Polymorphic skid systems, prevalent in military robots, also merit discussion. These robots have tracks that change shape to adapt to terrain. To see the advantages of a polymorphic skid system, consider a tank. First, recall that the tracks on the tank have a long contact area with the ground. The larger the area the tracks contact, the more traction and the more likely the track will be able to move and climb. However, the larger the contact area, the more power it takes to turn as the tank pivots on the inner track and the inner track must overcome friction to act as a pivot. Thus it would be desirable to have a track system where the amount of contact area could be adjusted for the situation. Second, recall that tanks generally have a trapezoidal geometry; therefore, the front (and sometimes the back) of the track is actually elevated to make it easier for the tank to climb over objects.

Polymorphic skid systems allow tracks to have a variable geometry. One approach, shown in figure 9.3a, is to create a tracked system where the continuous track changes shape, from flat to triangular. The advantage of this system is that it is highly adaptive and requires less power. The disadvantage is that, as the tracks lift up to climb over an obstacle or turn, the lift creates a gap or opening. Objects or debris may get into the gap causing the track to separate from the wheels moving the track, eventually disabling the robot. A second disadvantage is that the tracks tend to be flexible and loosely fitted so that they can adapt to different configurations, but the tracks may come off the rollers, or detrack.

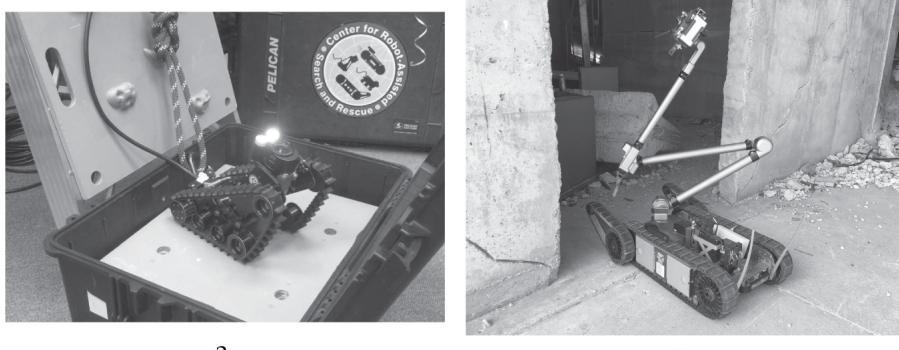


Figure 9.3 Example of skid-steered robots: a.) an Inuktun microVGTV with variable geometry and b.) an Endeavor Packbot 510 with flippers.

Another approach to polymorphic skid-steering is to add tracked flippers. The robot relies on a pair of continuous flat tracks but can raise and lower tracked flippers to go over obstacles or gain greater traction or maintain balance. An example is the Endeavor Packbot shown in [figure 9.3b](#). Other systems add flippers in the rear as well. Flippers eliminate the risk of an open track and enable the robot to right itself or crawl by using the flippers to push itself back over or pull itself forward.

9.3 Biomimetic Locomotion

The central concept in biomimetic locomotion is that there is a periodic, or repetitive, motion, either a vibration of the body or an oscillation of limbs. Using the concept of periodic motion, Siegwart, Nourbakhsh, and Scaramuzza¹⁹⁴ divide biomimetic locomotion into five major categories in order of decreasing energy consumption. Examples are shown in [figure 9.4](#),



a.



b.

Figure 9.4 Examples of biomimetic robots: a.) a long, thin active scope camera robot being inserted into rubble and b.) a snake robot. (Snake robot photograph courtesy of Howie Choset.)

CRAWLING

Crawling occurs when the agent overcomes friction through longitudinal vibration or movement. An example in nature is a caterpillar which contracts along the length of its body to move forward, creating a slow sinusoidal wave of motion. The *Active Scope Camera (ASC)* robot used to search the rubble at a building collapse in 2007 in Jacksonville is an example of a crawling robot.²⁰³ Rather than duplicate the sinusoid, the designers used vibrations to move the robot without complex mechanical deformation. The ASC is surrounded with a “skin” of VELCRO® tilted at an angle similar to the “hairs” in a hairy caterpillar. As the robot vibrates throughout the axis, the VELCRO® makes contact with surfaces and is

thrusted slightly forward due to the angle of tilt. Crawling is the most energy-intensive category of movement, so it should be no surprise that the ASC is connected to a large power supply.

SLIDING

Sliding occurs when the agent overcomes friction through transverse vibrations or movements. An example in nature is a snake which moves in a sinusoidal side-to-side pattern using its scales to increase friction enabling pivoting. A snake robot uses transverse vibration to move.

It should be noted that most robots that are called “snakes” do not use transverse locomotion. They are actually replicating the flexibility of a snake or an elephant’s trunk in order to move through tight spaces with high curvature or climb or grasp poles. The appeal of a snake or elephant trunk is that it is small but highly flexible, because it has many joints; in engineering this is called a hyper-redundant mechanism. The flexible robots often use a series of omnidirectional wheels to achieve the same mobility as a snake but without the frictional aspects.

The Snoopy robot shown in [figure 9.4b](#) is essentially an elephant’s trunk that can reach into holes in rubble created by standard concrete boring tools. Note that there is an external camera mounted on the base of the robot so that the operator can see where the trunk is in relation to the world. Recall that the camera is supplying a ex proprioceptive view. Major challenges with a hyper-redundant robot is sensing where it is and how it should move for a particular terrain and surface condition.

RUNNING

Running occurs when the agent overcomes kinetic energy with an oscillatory movement of a multi-link pendulum that leads to a predominately horizontal motion. A multi-link pendulum is an engineering representation of a leg and joints.

The RHex robot is an example of a running robot, though it does not at first appear to be anything like an animal (see [figure 9.5](#)).¹⁸³ RHex approximates the multi-link pendulum of cockroach’s leg as a spring. A cockroach does not move the joints on its legs very far; instead the joints serve as a way of making the leg into a spring to adjust to the terrain and bounce off of it. RHex does away with the joint. It duplicates the oscillatory movement of the legs by rotating the springs to push the body forward. RHex slaps the ground and pushes off of it as the leg completes the circle of rotation. RHex is one result of a style of analysis of biomechanics pioneered by Robert Full.⁷



Figure 9.5 RHex robot (Courtesy of Gadlopes at English Wikipedia (Transferred from en.wikipedia to Commons.) Public domain, via Wikimedia Commons).

Jumping is another way the agent can overcome kinetic energy with oscillatory movement of legs. In this case the oscillations produce a predominately vertical motion.

The early work of Boston Dynamic's founder Marc Raibert, who built the first robots to run and jump, used legs with pistons to duplicate how the foot and legs push off the ground. Marc Raibert's thesis, *Legged Robots that Balance*, was published in 1986. A companion video of his thesis work "Robots that run. And topple and fall" was also published. The video contained a blooper reel, showing memorable events such as when the one-legged hopper jumped on an obstacle instead of over it. The blooper reel did not detract from the multiple scientific contributions of the research; instead it increased confidence that Raibert's remarkable results were not the results of over-engineering the demonstrations.

Running, jumping, and walking are integral to legged locomotion and form a separate field of study. Raibert¹⁷³ provides a motivation for legged robots by noting that only half of the earth is accessible by wheels and that legs offer at least three advantages over other forms of locomotion. Legs can enable the agent to find isolated footholds, whereas wheels require continuous contact. Legs can adapt to provide active suspension for the body of the agent. Finally, legs are relatively energy efficient. Walking with legs is not as energy intensive as the other biological modes of crawling or sliding, and running, see figure 9.6. The energy cost of using legs compares well to the energy required for tires to drive on soft ground.

- Crawling/sliding
- Running
- Tires on soft ground
- Walking
- Railway wheels

Increasing power demand



Figure 9.6 Energy required for different modes of locomotion, adapted from.¹⁷³

9.4 Legged Locomotion

REFERENCE TRAJECTORY

LEG EVENT

As was seen in the previous section, legs can provide versatile locomotion at the lower end of energy costs for biological locomotion. In terms of AI robotics, legged locomotion has two components: *computing the reference trajectory*, or where the robot should go, and *computing the gait*, or the specific type of oscillation to move along the reference trajectory. Computing the reference trajectory is the purview of path and motion planning, described in chapter 14, while this chapter will describe computing the gait. Computing the gait can be hard because there is a large set of possible motions for the legs, called a *leg event*, to create an oscillatory motion, and the oscillatory motions have to be adapted to the terrain so that the legs land on the right footfalls in order to maintain balance.

9.4.1 Number of Leg Events

Siegwart, Nourbakhsh, and Scaramuzza note that the number of possible leg events, N , to achieve legged locomotion increases factorially.¹⁹⁴ The equation is

$$N = (2k - 1)! \quad (9.1)$$

where k = number of legs. Using this formula for an agent with two legs, $k = 2$, and thus there are $N = 6$ possible combinations of control directives to the two legs. The combinations are shown in [table 9.1](#), where the left leg can move up, U , down, D , or not at all, $-$.

Table 9.1 Combinations of leg events for two legs.

<i>LeftLeg</i>	<i>RightLeg</i>
<i>U</i>	—
—	<i>D</i>
—	<i>U</i>
<i>D</i>	—
<i>U</i>	<i>U</i>
<i>D</i>	<i>D</i>

$N = 6$ does not seem very large, but consider N for a hexapod like a bee. In that case, $k = 6$ and $N = 39, 916, 800$.

The problem of control increases if the leg has joints. Now the control signal is not just up, down, or not at all but it has to signal which parts, or linkages, of the leg move and by how much. The human leg is jointed such that there are seven degrees of freedom, or seven possible control directives to specify, just to move the leg to a desired footfall and this ignores placement of the foot and toes.

GAIT

As a result of the high number of leg events, the approach is to bundle the movements into a small set of precomputed coordinated movements called *gaits*.¹⁰⁰ Moving the agent along the reference trajectory then becomes a matter of specifying the gait to get the desired velocity, and then adding reactive control to adjust the actual footfall to the terrain. If terrain is too difficult, control will revert to a “free gait” where the agent has to compute the legged movements manually.

9.4.2 Balance

An obvious challenge in legged locomotion is maintaining balance. One approach is to create static walkers, that is, robots that are always statically balanced. The other is provide dynamic balance.

Static Balance

Whether an agent is balanced as it moves relies on the notion of a support polygon. For example, walking can be modeled as the step of an agent's legs duplicating rolling like a polygon within a circle to overcome gravity and move forward.

SUPPORT POLYGON

STATICALLY BALANCED

The *support polygon* is defined as the convex hull of the contact points with the ground. In the case of robots, the contact points are the legs touching the ground. If the center of mass (COM) of the agent is in the support polygon, then the agent is *statically balanced*. The support polygon may help illustrate why bipedal locomotion is challenging. The polygon degenerates to a line requiring more precise poses in order for the agent to remain balanced.

STATIC STABILITY

The first legged robots were designed to be statically balanced when they were stationary and also when they were moving, which is known as *static stability*. Static stability means that at every moment in time, the robot is in static balance. It also means that the legs maintain balance without any passive correction, or that if the robot put down a set of legs, the legs were firmly planted on the ground and would not slip.

Static stability is fairly rare in nature, and the best example is how a lobster walks. The agent usually has at least six legs since it has to lift some up. If it has three legs on the ground at any time to create a support polygon, then the COM stays in the changing support polygon. The agent puts down one set of legs to create a new support polygon without compromising the current static balance. The robot starts with all legs on the ground. Then the agent lifts a set of legs, L_1 , but leaves a set of legs, L_2 , on the ground to stay statically balanced. The raised legs, L_1 , are then moved forward and lowered, resulting in all legs back on the ground. Now L_2 are lifted and swing forward, pushing the agent's body forward. Keep in mind, the legs have to go nearly straight down because there is no correction of the footfalls. As a result, the agent cannot swing those legs very far forward.

Maintaining static stability while walking is called *static walking*. It is typically slow. In the case of lobsters, it helps the lobster move through strong underwater currents; speed is subordinated to safe movements. A famous static walking robot is the Ohio State Hexapod, by Ken Waldron, though disappointing from an AI perspective because traditional control theory methods for operating the hexapod were unable to provide real-time control with the processors of the time. In the end, the robot had to be teleoperated. Ambler, another static walker, was built by "Red" Whitaker at the Carnegie Mellon University (CMU) Field Robotics Center to be able to maintain a sensor platform at a level height by stepping over the majority of rocks but at a tremendous penalty in size, weight, and power. In the end, planetary rover researchers have gravitated toward wheeled vehicles with some type of articulation to maintain stability such as seen with Sandia National Laboratories' Rattler. An extension of the Ambler design philosophy was manifested in the static walking Dante robots. See [figure 9.7](#). These were built to rappel down steep canyons and volcanos on Mars (and Earth). Dante was able to lower itself successfully most of the way into a volcano in Antarctica, but it could not climb back out. It was dropped while being lifted out by a helicopter, twisting its frame.



Figure 9.7 Dante (photograph courtesy of NASA Ames Research Center).

Dynamic Balance

DYNAMIC BALANCE

Most animals rely on *dynamic balance*, and thus research is focused on creating robots that do not have to maintain static stability as they move. Consider normal human running: there is an instant of time when neither leg is on the ground as the human has pushed off of one foot and lifted it to start swinging it forward, but the other leg has not touched down yet. A challenge in dynamic balance is making sure that when the agent lands on the forward leg(s), the legs do not slip out from under causing a fall, and the legs are positioned such that the agent can spring off of them.

ZERO MOMENT POINT

This is commonly modeled as placing the legs so that there is *zero moment point* on the leg.^{100;217} Following figure 9.8, think of the leg as an inverted pendulum where the leg can make contact with the ground within a range of angles. The *zero moment point (zmp)* is the angle where the horizontal forces of momentum and friction are balanced and thus the robot should not fall. Although not legged locomotion, an example of exploiting balancing horizontal forces is the Segway®, where the rider leans forward as if an inverted pendulum, creating a horizontal force forward that unbalances the Segway and causes it to move forward to try to rebalance.

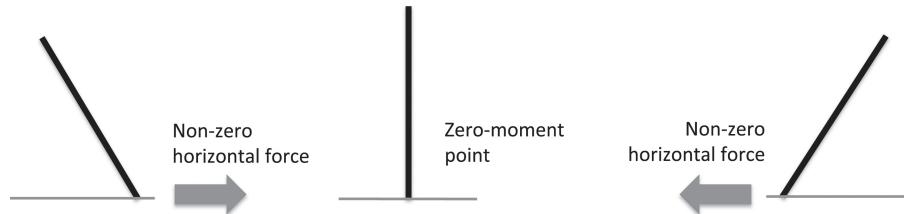


Figure 9.8 A leg as an inverted pendulum illustrating the zero moment point (ZMP).

The ZMP is used to compute where to place a leg, but it is also influenced by friction and the type of foot on the leg. Just like static walking, locomotion using the ZMP principle to place a leg typically has to “stomp” to get the foot flat against the ground to ensure that it will not slip.

9.4.3 Gaits

VIRTUAL GAIT

Being able to compute the ZMP is one step in dynamic balance of legged locomotion, but there is still the issue of how to reduce the number of legged events. A key concept in simplifying locomotion is organizing the oscillation of the legs into *virtual gaits*, first introduced by Marc Raibert.¹⁷³ Consider that a horse has well-defined gaits or patterns of leg events, such as trotting or galloping. The patterns work just the same for other organisms that have more than four legs, such as a millipede. Gaits can also be used to describe bipedal locomotion, where two legs are treated as a degenerative case.

The idea in virtual gaits is that legs are collected into two sets, *A* and *B*, and all the legs in *A* move at the same time in the same way and then next all the legs in *B* move in the same way at the same time. Unlike equation 9.1, where the computation increased factorially with the number of legs, the computation is fixed no matter how many legs an agent has.

There are three basic gaits shown in figure 9.9 based on how legs are paired:^{100;217}

- *Trot*, where diagonal pairs of legs alternate moving
- *Pace*, where lateral pairs alternate moving
- *Bound*, where front and back pairs alternate moving

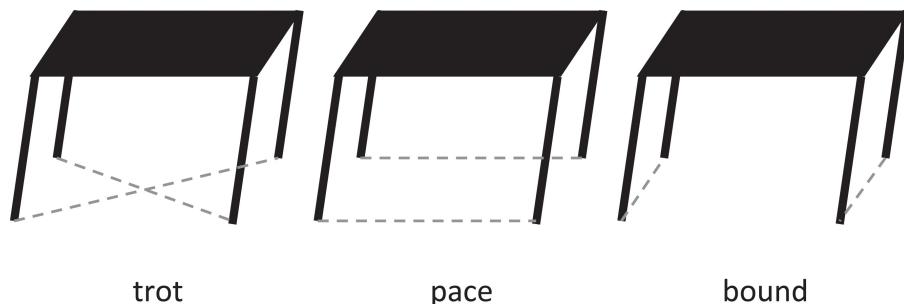


Figure 9.9 The three basic gaits: trot, pace, and bound.

9.4.4 Legs with Joints

Virtual gaits reduce the effort in planning the number of leg events for dynamic balance because it reduces the number of legs to two virtual sets. However, gaits generally treat legs as inverted pendulum, ignoring the fact that legs often have joints. The joints in a leg give the agent a mechanism by which to spring up with kinetic energy and thus increase the range of motion and the ability to adapt to terrain. They also reintroduce computational complexity. Indeed, Raibert's early robots used pistons on the legs to impart the kinetic energy without having to deal with the mechanical and control complexities of a more traditional jointed leg.

Following the idea of virtual gaits, joints in a leg are generally moved according to a small set of oscillatory patterns, or macros. Whereas gaits have been studied for thousands of years in animal husbandry (just consider horse riding throughout history and dressage in more recent times), the patterns for moving the joints on a single leg have not been well understood. There are two main approaches to determining the motion of legs with joints. One approach is to use motion capture cameras to determine how a person or animal moves their legs for a particular situation and then duplicate that movement. The other is to develop *central pattern generators (CPG)* for each joint.

CENTRAL PATTERN GENERATOR

In biology, there are neural structures called *central pattern generators (CPG)* that produce oscillations, such as breathing, swallowing, and locomotion, which require synchronized movements. Once the CPG is activated, the oscillations occur without the need for sensing or other additional inputs. However, the default pattern can be modified, or tuned, with sensing; for example, sometimes you have to concentrate on swallowing a big bolus of food or step farther than a normal pace. Ideally a CPG could be developed that creates the rolling polygon of leg motion up, forward, and down, and then the vertices on the polygon could be adjusted to adapt to sensed terrain.

While AI generally ignores locomotion, artificial intelligence researchers who specialize in neuroethology do study CPGs. The most common implementation of a CPG is with a variant of the nonlinear *Van der Pol equation*. The equation was developed to produce sinusoidal radio waves, which gives the default synchronized oscillation of limbs and explains noise in electronics, which can be used to tune the synchronization oscillation to external events. The challenges are that Van der Pol equations require a numerical method to solve and that CPGs have been studied in biology and implemented in robotics for a single joint, leaving the question of how to characterize and implement collections of CPGs to get a multijointed leg to move in a gait.

The two methods, motion capture and CPG, are not mutually exclusive. Motion capture can document how a gait changes to adapt to a terrain, such as climbing stairs, which then can be incorporated as a tuning input to a CPG or a new CPG that is released as appropriate.

9.5 Action Selection

The previous sections have covered how actions for locomotion are selected in one of two ways, either by having a planner explicitly plan each step (free walking) or by using hard-wired patterns, such as gaits and central pattern generators. However, a third way exists: learning to assemble existing motor schemas so that an agent learns to locomote. The learning approach reinforces the principles and merit

of behaviors. Robot learning is presented in more detail in chapter 16.

ACTION SELECTION

Action selection in AI robotics connotes how an agent chooses which behavior(s) to instantiate at a given time. Innate releasing mechanisms are technically a form of action selection, but those are preprogrammed and hardwired. The bigger question is how actions are selected and learned for new events. Originally it was assumed in AI that actions had to be selected deliberatively, with explicit planning, but as AI explored reactivity, researchers began to investigate the use of innate releasing mechanisms and similar mechanisms to choose what actions were executed without planning.

In 1990, Patty Maes and Rodney Brooks demonstrated Genghis, the six-legged insect robot introduced in chapter 1 (see [figure 6.3](#)), being turned on and learning to walk in a tripod gait within 1 minute and 45 seconds using only reactive action selection.^{[119](#)}

Genghis used two touch sensors, front and back, on its face and its behind. If either or both touch sensors were activated, the robot body was on the ground or severely tilted. As neither of those conditions is desirable, they produced a negative feedback signal. Genghis also had a trailing wheel to measure forward motion, which generated positive feedback if the wheel moved forward. Ideally, the robot's body would be off the ground and moving forward.

Genghis' choice of action was restricted to a small set of 13 behaviors. There were two behaviors per leg (swing forward, swing backward) for a total of 12, plus an additional behavior for horizontal balance to spread the legs out. The position of each leg as *up* or *down* was sensed with internal sensors.

Using this very simple configuration, Genghis learned to walk by learning the vector that represents the set of preconditions (or releasers) for each of the six leg behaviors. These can be *on*, *off*, *don't care*. The legs start off flailing, with each leg randomly trying its two different behaviors. At each time-step, the system used statistics to correlate the preconditions for each of the legs (What did I do that was right?) with the feedback. That would update the vector capturing the timing of each leg movement. The robot quickly learned a tripod static walking gait where, if one leg was up, the other two in the tripod were up and at least two of the remaining legs were down.

While this work was never widely adopted for locomotion in general, it was influential in the debate over when to react and when to deliberate as it was an existence proof that deliberation was not necessary for walking. It also set the stage for the neuroethological studies of gaits and CPGs.

9.6 Summary

This chapter has addressed two questions posed in the beginning of the chapter. The first question is: *How do you actually make a robot move?* Ground locomotion uses either mechanical or biomimetic principles. In practice, mechanical locomotion with wheels or tracks dominates robotics as the mechanisms are easy to build, steer, and keep the platform stable compared with multijointed legs. The two most common styles of steering are differential steering, or skid steering, and Ackerman steering, steering like a car. Skid steering is a very rough approximation of a holonomic vehicle. AI roboticists initially assumed holonomicity in order to ignore control issues and focus solely on AI functions.

Biomimetic locomotion ranges from walking to crawling or sliding. As seen by crawling and sliding, biomimetic locomotion is not guaranteed to be energy efficient, though legs in animals have good energetics (though less so in legged robots). In practice, wheels remain the most energy efficient

form of locomotion for ground robots but, in theory, legs are the most versatile. As a result, considerable research has been put into legged locomotion, including creating legs that look like springs; however, practical legs, as part of a complete robot, are still a long way off.

If an AI roboticist is working with legs, it is helpful to consider legged locomotion as two subproblems: generating the reference trajectory (where you want the robot to go) and planning the oscillation of the legs (how it will move the legs). Explicitly planning oscillations of a set of legs is called “free walking” and requires a factorial number of computations, $O(n!)$. The computational complexity and the mechanical complexity of legs answer the second question posed at the beginning of this chapter: *Why don’t more robots have legs?*

To simplify the computational complexity, roboticists pair legs and produce a set of oscillations identical to gaits in animals. The three virtual gaits commonly used are trotting, pacing, and bounding. The robot employs a gait, which may be computed by a central pattern generator and then makes adjustments to the footfall of each leg to stay balanced and adapt to terrain.

While gaits simplify planning the oscillations, the problem of balance and adapting to terrain remains non-trivial. Early legged platforms used static balance to guarantee the platform’s center of mass would always remain within the support polygon provided by the legs. Unfortunately static balance is slow and cumbersome. Dynamic balance is faster, but it is more challenging to balance static and dynamic forces. Foot placement becomes more critical, requiring that a leg land at a point that will produce a zero moment product as the robot pivots on that leg.

Gaits map nicely onto the concept of behaviors and AI. Furthermore, the CPG is an natural extension of behavior-based robotics and reinforces the value of the tradition of exploiting biological and ethological conditions. Machine learning was explored in the early days of walking robots, primarily as a motivation to study the action selection problem or how behaviors get chosen and grouped to accomplish a task. However, CPGs are more practical and suitable for multijointed legs.

9.7 Exercises

Exercise 9.1

What is the difference between Ackerman and differential steering?

Exercise 9.2

What is the difference between holonomic and nonholonomic vehicles? Give an example of a robot or vehicle exhibiting each.

Exercise 9.3

Fill in the ?? with *Ackerman steering*, *differential steering*, *holonomic*, and *nonholonomic*:

AI researchers often assume a robot is ??, meaning it is a point and has no mass because then they do not have to worry about programming control schemes for slowing down or accounting for the turning radius of different robots. Unfortunately an unmanned marine system (robot boat) is ?? because a marine vehicle cannot easily turn sharply and may keep coasting in the water after the thrust is cut. Ground robots have similar problems. ?? is sometimes used to enable turning in place, though there will always be skitter. Large ground robots often use ??, which has all the challenges of parking a car.

Exercise 9.4

Rank the following list of locomotion types in order of power demands:

- a. crawling/sliding
- b. railway wheels
- c. running
- d. tires on soft ground
- e. walking.

Exercise 9.5

List the advantages and disadvantages of polymorphic skid steering.

Exercise 9.6

What would be the number of possible leg events for a spider?

Exercise 9.7

List three advantages of legs over other forms of locomotion.

Exercise 9.8

What is the difference between static and dynamic balance? Give an example of an animal exhibiting each.

Exercise 9.9

Why is static stability often considered desirable for robots?

Exercise 9.10

Define:

- a. support polygon
- b. zero moment point
- c. action selection
- d. reference trajectory
- e. central pattern generators.

Exercise 9.11

Name and draw the three virtual gaits.

9.8 End Notes

Rising Sun

Marc Raibert's one-legged hopper appears for a few seconds in the 1993 movie *Rising Sun*. Sean Connery and Wesley Snipes walk past it as they enter a high-tech research center. It took several days to get the hopper hopping long enough and coordinated with the actors for the shot, earning Raibert the ire of Sean Connery.

Dante and the media.

The Dante project received heavy media coverage and percolated into the entertainment industry. A small, Dante-like robot appears in the movie, *Dante's Peak*, as part of the team of vulcanologists led by Pierce Brosnan. The robot is continually breaking in the movie, which is unfortunately an all-too-accurate reflection of the current state of robotics. In the "Firewalker" episode of the TV series *The X-Files*, a robot, looking like an Odetics "spider" robot, is down in a volcano and broadcasts disturbing images implying that the vulcanologists have been attacked by a mysterious life form. The culprit turned out to be a scientist who had quit taking his medication.

10

Sensors and Sensing

Chapter Objectives:

- Describe the difference between *active* and *passive* sensors, and give examples of each.
- List at least one advantage and disadvantage of common robotic sensors: *GPS*, *INS*, *IR*, *RGB-D* cameras.
- Define *image*, *pixel*, and *image function*.
- If given a small interleaved RGB image and a range of color values for a region, be able to write code to extract color affordances using 1) threshold on color and 2) a color histogram.
- Write computer vision code to enable a robot to imprint on, and track, a color.
- Define each of the following terms in one or two sentences: *proximity sensor*, *logical sensor*, *false positive*, *false negative*, *hue*, *saturation*, and *computer vision*.
- Describe the three types of behavioral *sensor fusion*: fission, fusion, fashion.
- List the attributes for designing a *sensor suite*, and apply these attributes to a particular application.
- Define *locomotion load* and *hotel load* and explain why sufficient hotel load is critical for design of intelligent robots.

10.1 Overview

To review chapter 7, perception in the Reactive layer has two roles: *to release a behavior* and *to support or guide the actions of the behavior*. All sensing is behavior-specific, where behaviors may tap into the same sensors but use the data independent of each other. Also, the connotation of reactive robots is that behaviors are most often the result of stimulus-response, relying on *direct perception* rather than requiring memory.

In the Deliberative layer, sensing is used to recognize and reason about objects, scenes, and events in order to build a world model. Whereas the Reactive layer uses, what Gibson and Neisser would call, the direct perception pathways in the brain, the Deliberative layer uses the *object recognition* pathways.

In the Interactive layer, sensing is used to detect and support social interactions. These social

interactions can be quite coarse, such as avoiding invading personal space. They can also be high resolution, where the robot may need to perceive and interpret subtle facial features and gestures.

The pervasive use of perception leads to a subtle distinction between sensors and sensing in artificial intelligence. Sensors provide the raw data, while sensing is the combination of algorithm(s) and sensor(s) that produces a percept or world model.

The need for perception leads to several general questions about sensing. The first question is the most basic: *How do you make a robot “see”?*? The previous chapters on behaviors should have already planted the seeds for the idea that perception is not just the sensor; there is additional processing. Artificial intelligence addresses how to extract or infer a percept or object or how to interpret what is going on in a scene—and how to do so with any combination of specific sensors.

Another question is: *What sensors are essential for a robot?* The specific answer depends on the mission, of course, but, in general, robots need both proprioception and exteroception. However, since robots should always have a teleoperation “backdoor” as discussed in chapter 5, a robot will generally always have a visible light camera. Exploiting that camera(s) is the purview of the field of computer vision.

The final question is: *What is sensor fusion?* The previous chapter on behavior-based robotics indicated that an individual behavior could accept inputs from more than one sensor. The description of deliberation in chapter 4 defines the notion of a world model which is the sum of all of the sensing. The sensing mechanisms, which allow multiple sensors to produce percepts and models, are generally referred to as sensor fusion.

This chapter does not answer the question of: *How to sense depth and range*. Sensing depth and range without direct contact is critical for navigation and is a field onto itself. Thus, range sensing is discussed in the next chapter.

This chapter will address the basics of sensors and sensing from an artificial intelligence perspective. It will introduce three common knowledge representations, polar plots, image functions, and logical sensors, as well as the general AI area of computer vision. It will first introduce a model of sensors and sensing that will be used to discuss active and passive sensors. The chapter defers discussion of range sensing until the next chapter (chapter 11) where RGB-D cameras, lidar, and other mechanisms are addressed in detail.

10.2 Sensor and Sensing Model

SENSOR

TRANSDUCER

Regardless of sensor hardware or application, sensing and sensors can be thought of interacting with the world and robots as shown in [figure 10.1](#). The *sensor* is a device that measures some attribute of the world. The term *transducer* is often used interchangeably with *sensor*. A transducer is the mechanism, or element, of the sensor that transforms the energy associated with what is being measured into another form of energy.⁶ A sensor receives energy and transmits a signal to a display or a computer. Sensors use transducers to change the input signal (sound, light, pressure, temperature, etc.) into an analog or digital form capable of being used by a robot. For reactive behaviors, the sensor observation is intercepted by a perceptual schema which extracts the relevant percept of the environment for the

behavior. This percept is then used by the motor schema, which leads to an action.

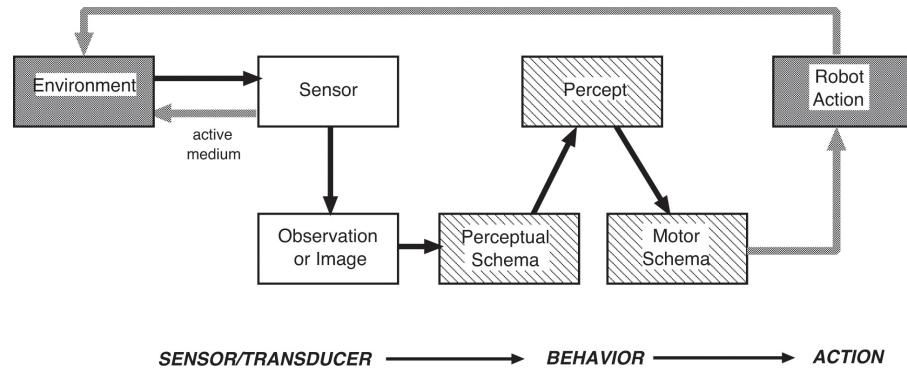


Figure 10.1 A model of reactive sensing (a world model would replace the schemas in deliberative sensing).

Figure 10.1 illustrates sensing in the Reactive layer. For the Deliberative layer, the perceptual schema–percept–motor schema configuration of modules would be replaced by the world model. The Interactive layer might employ behaviors or use the world model for its sensing needs. Next, the chapter will discuss odometry, Inertial navigation system (INS), Global Positioning System (GPS), and proximity sensors. Computer vision and image processing are described next. The field of computer vision is so broad that it is impossible to go into all the possible algorithms, so the chapter focuses on simple, common routines used with reactive behaviors in order to reinforce the notion of reactivity. It also illustrates how the perception that we have taken for granted is actually quite hard. The chapter ends with a list of attributes to be used when selecting a sensor suite and covers other concepts that will improve perceptual design.

10.2.1 Sensors: Active or Passive

PASSIVE SENSOR

ACTIVE SENSOR

ACTIVE SENSING

A sensor is often classified as being either a *passive* or an *active*. Passive sensors rely on the environment to provide the medium for observation, for example, a camera requires a certain amount of ambient light to produce a usable picture. Active sensors put out energy into the environment either to change the energy or enhance it. Sonar sends out sound waves, receives the echo, and measures the time of travel. An X-ray machine emits x-rays and measures the amount blocked by various types of tissues. Although a camera is a passive device, a camera with a flash is an active sensor. The term “active sensor” is not the same as *active sensing*. Active sensing connotes a system using an effector to dynamically position a sensor to improve data gathering. For example, a camera with a flash is an active sensor, while a camera mounted on a pan/tilt head, with algorithms to turn it to get a better view, is using active sensing.

MODALITY

Different sensors measure different forms of energy. This in turn leads to different types of processing. Sensors which measure the same form of energy and process it in similar ways form a sensor *modality*. A sensor modality refers raw input to the sensor: sound, pressure, temperature, light, and so on. In some regard, modalities are similar to the five senses in humans. A modality can be further subdivided, for instance vision can be decomposed into visible light, infrared light, and X-rays.

10.2.2 Sensors: Types of Output and Usage

Sensors can also be categorized by the type of output they produce and how the sensed data is used. The output of a sensor is either a reading or an image. Images will be formally defined later in this chapter as they are a fundamental and popular representation.

As described in chapter 5, sensing about the world is generally used in three ways. One is for proprioception, or to locate the position of limbs and joints of the robot or to determine how much they have moved. Proprioception is essential for control of the robot platform. Another is exteroception, to detect objects in the external world and often the distance to those objects. Exteroception is essential to enable the robot to move and act in the world. The third is exproprioception, to detect the position of the robot relative to objects in the world. This is also important for manipulation and for reasoning about why the robot is not moving (i.e., it is stuck).

10.3 Odometry, Inertial Navigation System (INS) and Global Positioning System (GPS)

SHAFT ENCODERS

Robots can estimate their movements with proprioceptive sensors, such as *shaft encoders*, which measure the number of turns the drive motor has made; the shaft encoder works the same way an odometer on a car works. If the gearing and wheel size are known, then the number of turns of the motor can be used to compute the number of turns of the robot's wheels, and that number can be used to estimate how far the robot has traveled. Unfortunately, this is only an estimate as the environment impacts the actual movement of the robot. A robot on a tiled floor might slip twice as much as a robot on dry grass.

INS

Inertial navigation systems (INS) and inertia movement systems (IMU), such as the accelerometers used on smart phones, can provide dead reckoning information as well and often do much better than odometry. As long as the movements are smooth, with no sudden jarring, and the samples are taken frequently, an INS can provide accurate dead reckoning to 0.1 percent of the distance traveled.⁷⁴ Unfortunately a hard bump or sudden turn can exceed the accelerometers' measurement range, introducing errors. Sojourner, the Mars rover, carried an INS system. In one trek, it would have stopped 30 cm from a rock it was to sample if it had just used proprioception. Instead, by using exteroception, it got within 4 cm.

GLOBAL POSITIONING SYSTEM

URBAN CANYONS

GPS, or *global positioning system*, is common on robots that work outdoors. GPS systems work by

receiving signals from satellites orbiting the Earth. The receiver calculates its position relative to at least four GPS satellites in terms of latitude, longitude, altitude, and change in time. GPS is not a proprioceptive sensor per se since the robot must receive signals from satellites external to the robot. However, it is not an exteroceptive sensor either, since the robot is not computing its position relative to its environment. Regardless of what kind of sensor it is, GPS is not a complete solution to the dead reckoning problem in mobile robots as GPS does not work indoors in most buildings, especially offices or factories with large amounts of steel-reinforced concrete. These structures interrupt the reception of radio signals, just like they do with cellular phone networks. Likewise, GPS may not work outdoors in major cities where skyscrapers and bridges act as *urban canyons* and interfere with reception.

10.4 Proximity Sensors

PROXIMITY SENSOR

Proximity sensors directly measure the relative distance (range) between the sensor and objects in the environment. Proximity sensors generally connote measuring short distances to an object, on the order of one meter versus a range sensor which senses over much larger distances. Proximity sensors can be either active or passive. Active proximity sensors allow the robot to detect objects before contact, while passive proximity sensors require contact with an object or surface. Infrared sensors are the most popular active proximity sensor, with bump and feeler sensors the most popular passive sensors.

IR

Infrared sensors (IR) emit near-infrared energy and measure whether any significant amount of the IR light is returned. If an obstacle is present, it returns a binary signal. IR sensors have a range of inches to several feet, depending on what frequency of light is used and the sensitivity of the receiver. The simplest IR proximity sensors can be constructed from LEDs, which emit light into the environment and have a range of 3–5 inches. [Figure 10.2](#) shows the IR emitters and receivers placed side by side in a single rectangular package on a Khepera robot. These emitters often fail in practice because the light emitted is often “washed out” by bright ambient lighting or is absorbed by dark materials (i.e., the environment has too much noise). In more sophisticated IR sensors, different IR bands can be selected or modulated to change the signal-to-noise ratio which typically ensures that an object in range does not absorb the light and causes the sensor to miss the presence of the object.

TACTILE

Another popular class of robotic sensing is *tactile*, or touch, done with bump and feeler sensors. Feelers or whiskers can be constructed from sturdy wires. Bump sensors usually consist of two layers forming a protruding ring around the robot. Contact with an object causes the two layers to touch, creating an electrical signal. Placement of bump sensors is a very important issue. The bump sensors on a Nomadic Technologies, Inc.’s Nomad 200 base were intended to protect the highly accurate synchro-drive mechanism from low obstacles. Unfortunately, in certain turning configurations, the wheels extended beyond the cowling, rendering the bump sensor totally useless for preventing damage.

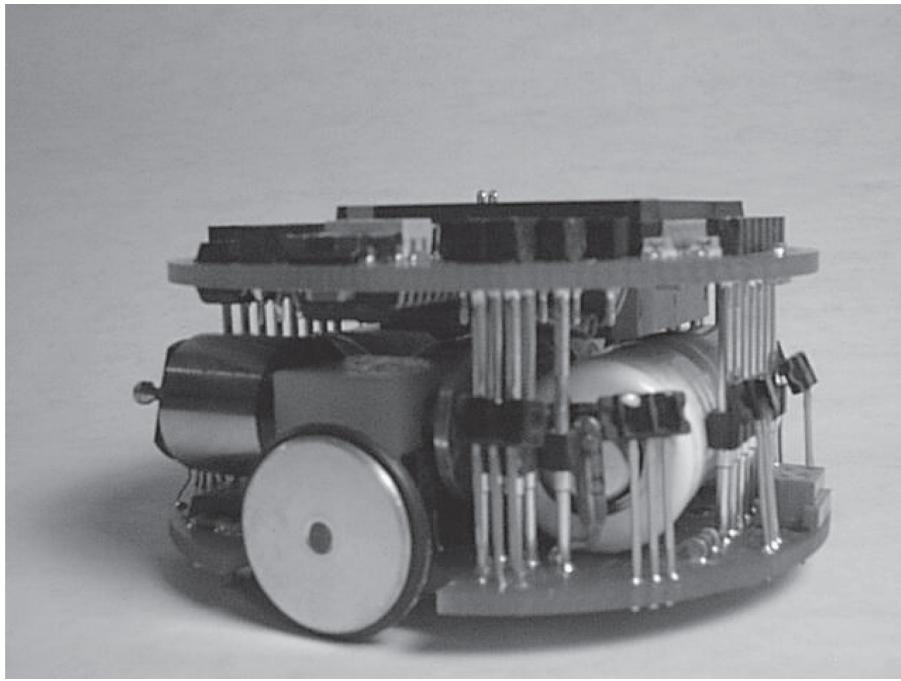


Figure 10.2 The ring of IR sensors on a Khepera robot. Each black square mounted on three posts is a emitter and receiver.

In theory, the sensitivity of a bump sensor or whisker can be adjusted for different contact pressures; some robots may want a “light” touch to create a signal rather than a “heavier” touch. In practice, bump sensors are annoyingly difficult to adjust. In the “Hors d’Oeuvres, Anyone?” event of the 1997 AAAI Mobile Robot Competition, humans were served finger food by robot “waiters.” Humans were supposed to communicate to the Colorado School of Mines’ Nomad 200 robot waiter that they were done eating by kicking the bump sensor mounted on the bottom of the robot. The sensitivity of the bump sensor was so low that it often required many kicks, producing a very comical scene with Bruce Lee overtones.

10.5 Computer Vision

Computer vision is the primary source of general purpose exteroceptive sensing for direct perception and object recognition. Computer vision can also be used for measuring distances, a method is described in the next chapter. This section introduces computer vision and a few of the basic algorithms used in direct perception. As noted in chapter 1, computer vision is a separate field of study from robotics and has produced many useful algorithms for filtering out noise, compensating for illumination problems, enhancing images, finding lines, matching lines to models, extracting shapes, and building 3D representations.

10.5.1 Computer Vision Definition

COMPUTER VISION

IMAGE

PIXEL

IMAGE FUNCTION

Computer vision refers to processing data from any modality which uses the electromagnetic spectrum to produce an image. An *image* is a way of representing data in a picture-like format where there is a direct physical correspondence to the scene being imaged. Unlike sonar, which returns a single range reading that could correspond to an object anywhere within a 30° cone, an image implies multiple readings placed in a two-dimensional array or grid. Every element in the array maps onto a small region of space. The elements in image arrays are called *pixels*, a contraction of the words “picture element.” The modality of the device determines what the image measures. If a visible light camera is used, then the value stored in each pixel is the value of the light (e.g., color). If a thermal camera is used, then the value stored is the heat at that region. The function that converts a signal into a pixel value is called an *image function*.

Computer vision includes output from any type of camera that produces images over the same electromagnetic spectrum that humans see, to more exotic technologies: thermal sensors, X-rays, laser range finders, and synthetic aperture radar. The more exotic sensors may be difficult for a human attempting to visualize the value of the data in the image. It is important to remember that it is the output of an image representation that places a sensor in the computer vision category.

10.5.2 Grayscale and Color Representation

The value of a pixel is generally represented as grayscale or as color. Grayscale representations are standard, using an 8-bit number (1 byte of computer memory). This leads to 256 discrete values of gray, with 0 representing black and 255 representing white. (Remember, 256 values means 0 ... 255.)

RGB

However, color is represented differently, and each representation has advantages and disadvantages. First, there are many different methods of expressing color. Home PC printers use a subtractive method, where cyan plus yellow make green. Most commercial devices in the U.S. use the National Television System Committee (NTSC) standard. NTSC color is expressed as the sum of three measurements: red, green, and blue, abbreviated as *RGB*.

COLOR PLANES

RGB is usually represented as three *color planes*, or axes, of a 3D cube as shown in [figure 10.3](#). The cube represents all possible colors. A specific color is represented by a tuple of three values to be summed: (R, G, B). Black is (0, 0, 0) or 0+0+0, or no measurements on any of the three color planes. White is (255, 255, 255). The pure colors of Red, Green, and Blue are represented by (255, 0, 0), (0, 255, 0), and (0, 0, 255), respectively. The representations are the same as for color graphics.

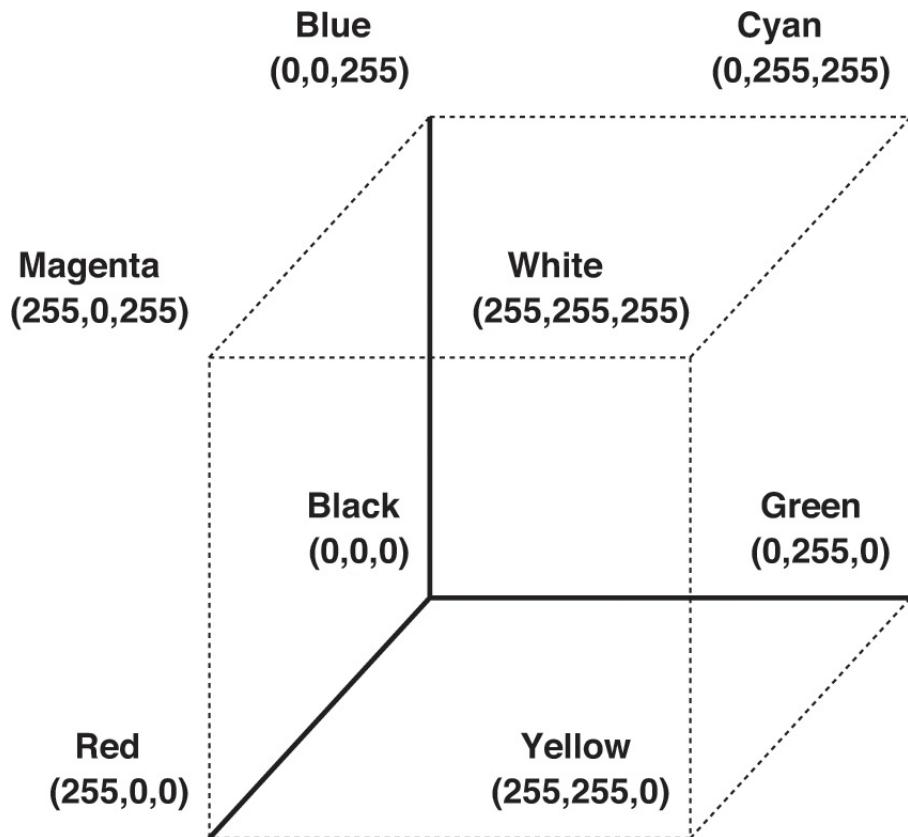


Figure 10.3 RGB color cube.

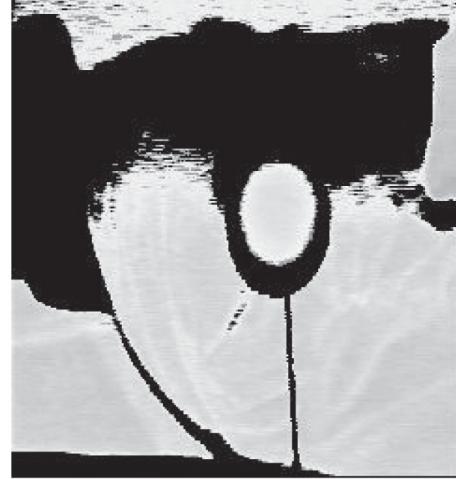
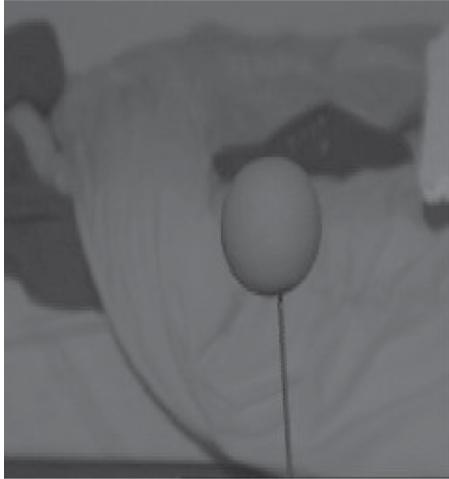
Notice that the cube dimensions in the figure are 256 by 256 by 256, where 256 is the range of integers that can be expressed with 8 bits. Since there are three color dimensions, a manufacturer may refer to this cube as 24-bit color (3×8), to distinguish a particular framegrabber from ones which map color onto a linear grayscale. The 8-bit color model is what is used to colorize old black and white movies. There are only 256 values of color, which is quite limited, and the gray values are often ambiguous. The pixel values of a woman's red lips might be 185, while her dark blue dress is also 185. A person may have to manually disambiguate regions in each frame of the film where 185=red and regions where 185=dark blue. 8-bit color is not often used for robot vision, unless the robot will be operating in an environment where the only visible colors will not have an ambiguity.

24-bit color is usually sufficient for robotics. For other applications of computer vision such as medical imaging, weather forecasting, or military reconnaissance, an 8-bit resolution is often insufficient. Those applications may use 10 bits for each color plane. Since 10 bits do not fall on a byte boundary, it can be awkward to program algorithms for representing and manipulating these kind of images. Special computers are often made for these applications.

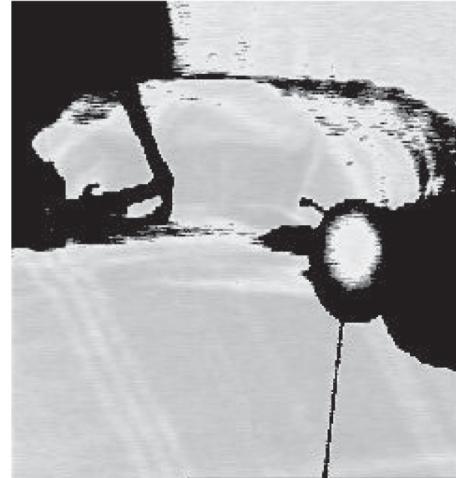
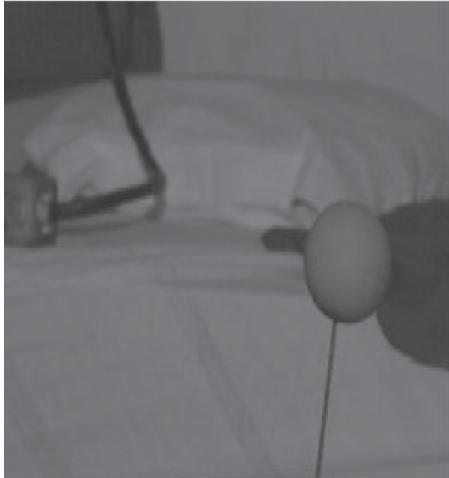
VISUAL EROSION

The RGB representation has disadvantages for robotics. Color in RGB is a function of the

wavelength of the light source, how the surface of the object modifies it (surface reflectance), and the sensitivity of the sensor. The first problem is that color is not absolute. RGB is based on the sensitivity to reflected light of the three-color sensing elements. An object may appear to have different values at different distances due to the intensity of the reflected light. [Figure 10.4](#) shows the sensitivity to reflected light. The photos show two images taken of orange landmark that is serving as a “flag” for tracking a small robot. The images were taken as the camera moved, thus changing the view. Even though the images were processed by the same program and parameters to segment an just outside of the image, the results are different. The RGB segmentation in [Figure 10.4a](#) is more correct than in [figure 10.4b](#). The only difference is that flagged robot has moved, thereby changing the angle of the incidence of the light. This degradation in segmentation quality is called *visual erosion* because the object appears to erode with changes in lighting. Moreover, digital camera devices are notoriously insensitive to red. This means that one of the three color planes is not as helpful in distinguishing colors.



a.



b.

Figure 10.4 Images showing visual erosion of an orange landmark sticking up from a small robot (not visible): a.) Original image and RGB segmentation and b.) original image and degradation in RGB segmentation as robot moves farther from the object.

HUE

HSI

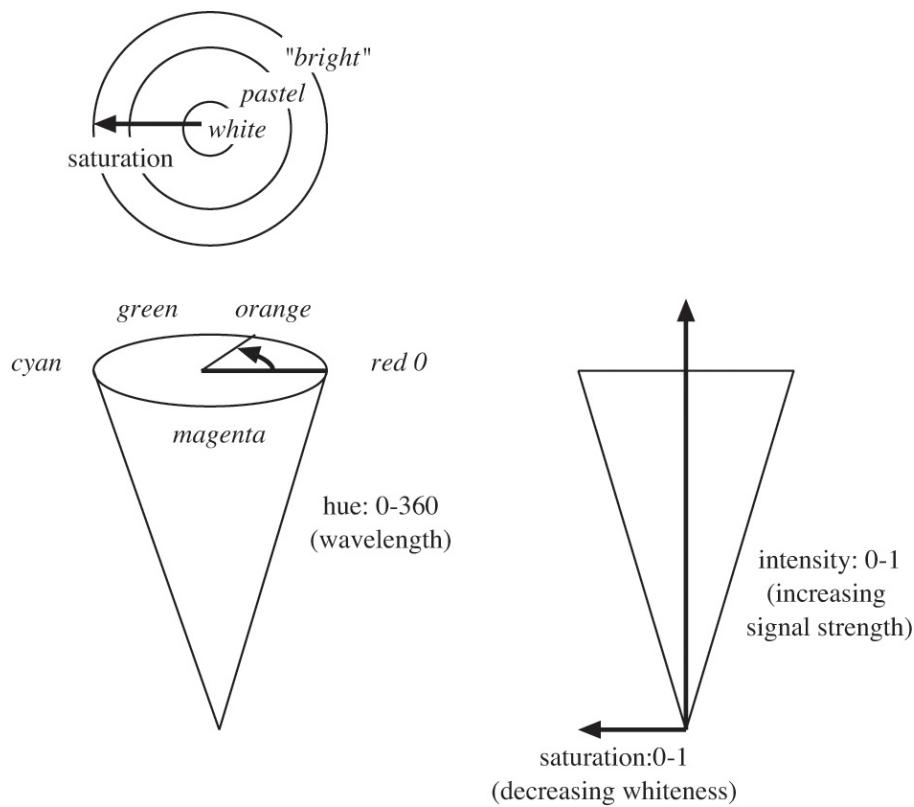
SATURATION

Clearly a camera which is sensitive to the absolute wavelength of the reflected light (called the hue) would be more advantageous than having to work around the limitations of RGB. The *hue* is the dominant wavelength and does not change with the robot's relative position or the object's shape. Such a camera works on the *HSIHSIHSI* (hue, saturation, intensity) representation of color. *Saturation* is the

lack of whiteness in the color; red is saturated, pink is less saturated. The *value* or *intensity* measure is the quantity of light received by the sensor, and HSI representations are often called HSV. HSV is a very different color scheme than RGB.

HSV

HSV is a three-dimensional space in that it has three variables, but it is definitely not a cube representation; it is more of a cone as seen in [figure 10.5](#). The hue, or color, is measured in degrees from 0 to 360. Saturation and intensity are real numbers between 0 and 1. These are generally scaled to 8-bit numbers. Accordingly, red is both 0 and 255, orange is 17, green is 85, blue is 170, and magenta is 200.



[Figure 10.5](#) HSV space representation.

HSV space is challenging for roboticists for many reasons. First, it requires special cameras and framegrabbers to directly measure color in HSV space. This equipment is prohibitively expensive. Second, there is a software conversion from the RGB space, but it is not used because the conversion is computationally expensive and it has singularities where the algorithm fails. These singularities occur at places where the three colors for a pixel are the same; the flatness of the red color plane in CCD cameras increases the likelihood that a singularity will occur.

An alternative color space that is currently being explored for robotics is the Spherical Coordinate

Transform (SCT).²⁰⁹ That color space was designed to transform RGB data into a color space that more closely replicates the response of the human eye. It is used in biomedical imaging but has not been widely considered for robotics. The shape of the color space is triangular, as shown in [figure 10.6](#). Initial results indicate it is much more insensitive to changes in lighting than RGB.⁹⁶ [Figure 10.7](#) shows an image and the results of segmenting a color in RGB, HSI, and SCT spaces.

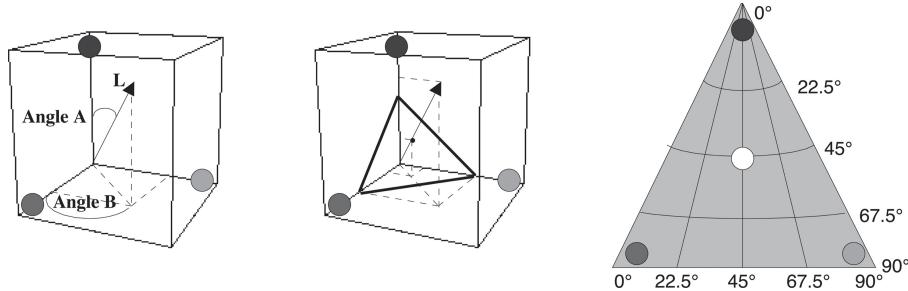


Figure 10.6 Spherical coordinate transform.

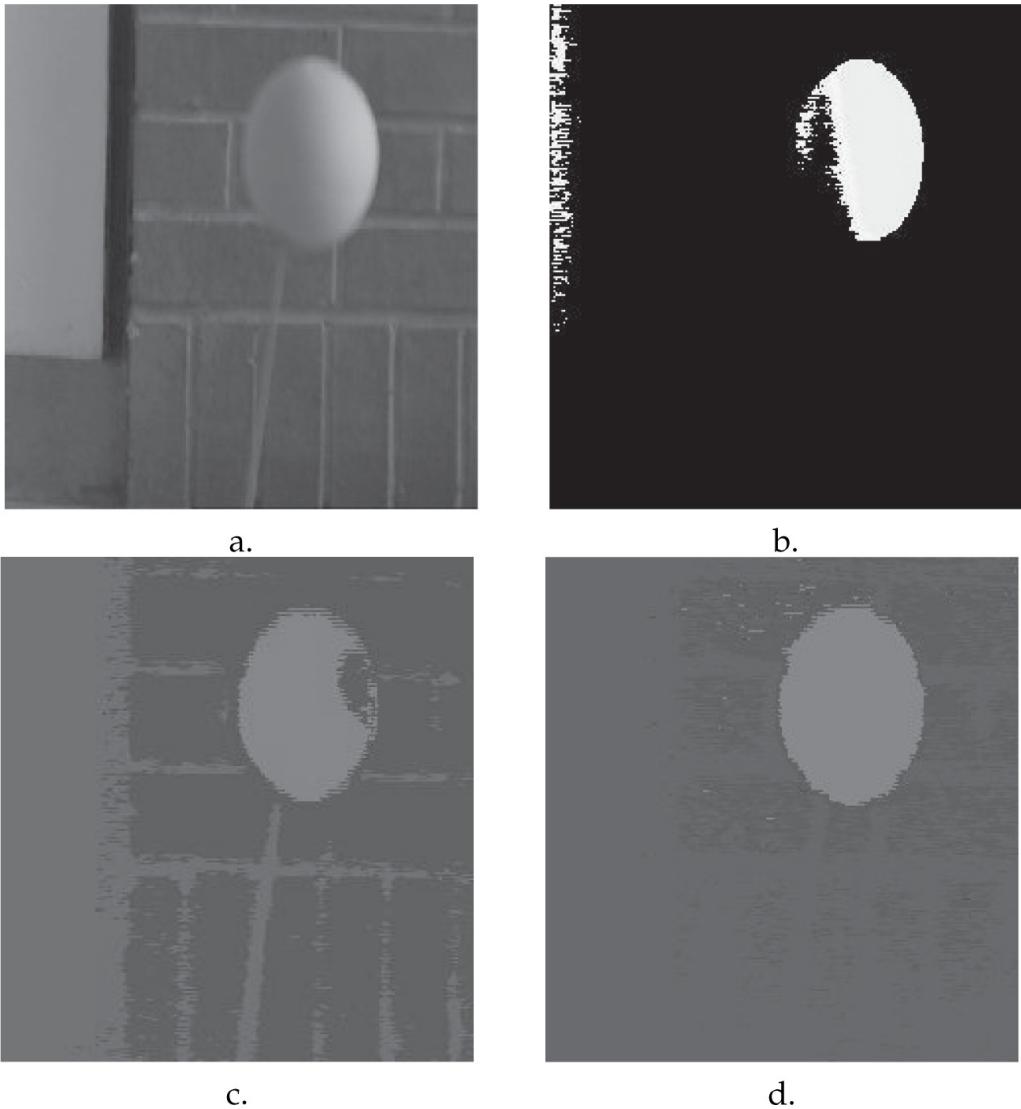


Figure 10.7 A comparison of color spaces: a.) a landmark in the original image, b.) RGB segementation, c.) HSI segementation, and d.) SCT segmentation.

10.5.3 Region Segmentation

REGION SEGMENTATION

The most ubiquitous use of computer vision in reactive robotics is to identify in the image a region of a particular color, a process called *region segmentation*. Region segmentation and color affordances are staple perceptual algorithms for successful entries to many different international robot competitions, including RoboCup. The basic concept is to identify all the pixels in an image which are part of the region and then to navigate to the region's center (centroid). The first step is to select all pixels which share the same color (thresholding), then group those together and discard any pixels which do not

seem to be in the same area as the majority of the pixels (region growing).

BINARY IMAGE

THRESHOLDING

Chapter 19 describes a robot which used red to signify a red Coca-Cola can for recycling. Ideally, the robot during the searching-for-the-can behavior would see the world as a *binary image* (having only two values) consisting of red, not-red. This partitioning of the world can be achieved by *thresholding* the image and creating a binary image. A C/C++ code example is shown below:

```
for (i= 0; i < numberRows; i++)
    for (j= 0; j < numberColumns; j++) {
        if ((ImageIn[i] [j] [RED] == redValue)
            && (ImageIn[i] [j] [GREEN] == greenValue)
            && (ImageIn[i] [j] [BLUE] == blueValue)) {
            ImageOut[i] [j] = 255;
        }
        else {
            ImageOut [i] [j] = 0;
        }
    }
```

Note that the output of a thresholded color image is a two-dimensional array since there is no need to have more than one value attached to each pixel. Also, in theory, a binary image would permit only values of 0 and 1. However, on many compilers there is no particular benefit to doing a bit-level representation, and it can complicate code reuse. Also, most display software is used to displaying at least 256 values. The difference between 1 and 0 is not detectable by the human eye. Therefore, it is more common to replace the 1 with a 255 and use a full byte per pixel.

Thresholding works better in theory than in practice due to its lack of color constancy. The shape of an object means that, even though a human sees the object as a solid color, the computer sees it as a set of similar colors. The common solution is to specify a range of high and low values on each color plane. The C/C++ code now becomes:

```
for (i= 0; i< numberRows; i++)
    for (j= 0; j<numberColumns; j++) {
        if (((ImageIn[i] [j] [RED] >= redValueLow)
            && (ImageIn[i] [j] [RED] <= redValueHigh))
            &&((ImageIn[i] [j] [GREEN]>=greenValueLow)
                &&(ImageIn[i] [j] [GREEN] <= greenValueHigh))
            &&((ImageIn[i] [j] [BLUE]>=blueValueLow)
                &&(ImageIn[i] [j] [BLUE] <= blueValueHigh))) {
            ImageOut[i] [j] = 255;
        }
        else {
            ImageOut [i] [j] = 0;
        }
    }
```

FOREGROUND

BACKGROUND

The change in viewpoints and lighting means that the range of pixel values which defines the object in the robot's current position is likely to change when the robot moves to a new position. One approach is to make the color range for the object even wider to include the set of all possible pixel values for object when seen from different viewpoints. If the object color is unique for that environment, this increase in the color range is acceptable. Otherwise, if there are objects which have a color close enough to the object of interest, those objects may be mistaken for the target. In some circles, the object of interest is called the *foreground*, while everything else in the image is called the *background*. To be successful, thresholding an image requires a significant contrast between the background and foreground. Fortunately, there are now "camshift" algorithms which can adapt the color range of a region using statistics.

Figure 10.8 illustrates the output of a threshold on color alone. If a robot was to move to the "red" in the image, how would it know which pixel to follow? The perceptual schema could be instantiated for each red pixel; this is simple, but it would waste a lot of execution cycles. The perceptual schema could take the weighted centroid of all the red pixels. In this case, the center would be located somewhat near where most people would say the can was, but the location would be closer to the white cup. Or, the perceptual schema could attempt to find the largest region where red pixels were adjacent to each other and then take the centroid of that region. (The region is often referred to as a "blob," and the extraction process is known as blob analysis.)

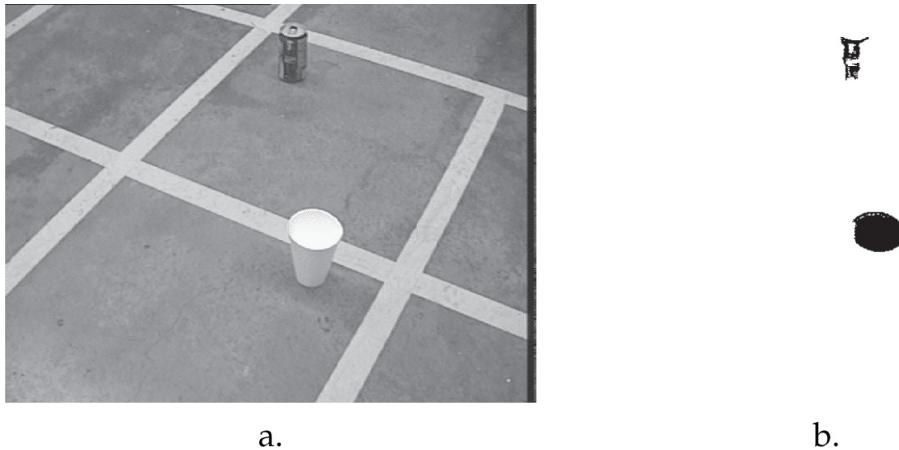


Figure 10.8 Segmentation of a red Coca-Cola can: a.) original image and b.) resulting red regions. Note that some pixels not associated with the can showed a reddish color.

Color regions can also be helpful in cluttered environments. Figure 10.9 shows a Denning mobile robot simulating a search of a collapsed building. The international orange vest of the workman provides an important cue. The robot can signal a teleoperator when it sees bright colors.

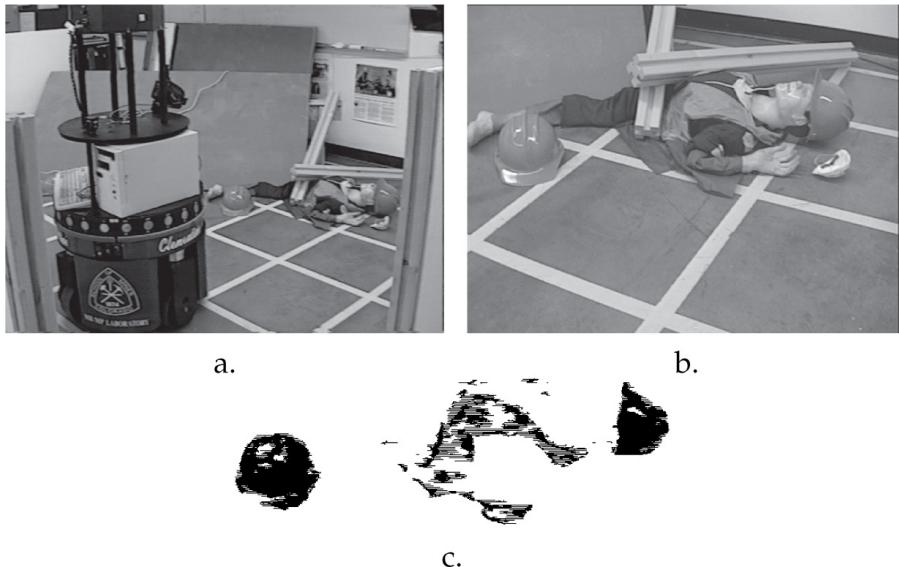


Figure 10.9 An urban search and rescue scene: a.) a Denning mobile robot searching, b.) image from the camera, and c.) segmentation for orange.

10.5.4 Color Histogramming

COLOR HISTOGRAMMING

Thresholding works well for objects which consist of one color or one dominant color. A different technique, *color histogramming*, can be used to identify a region with several colors.²⁰² Essentially, color histogramming is a way to match the proportion of colors in a region.

A histogram is a bar chart of data. The user specifies the range of values for each bar, called buckets. The length of the bar represents the number of data points with values that fall into the range for that bucket. For example, a histogram for a grayscale image might have eight buckets (0-31, 32-63, 64-95, 96-127, 128-159, 160-191, 192-223, 224-251), and each bucket contains the number of pixels in the image that fell into that range. Constructing a color histogram is straight forward for a region in hue space, as shown in [figure 10.10](#).

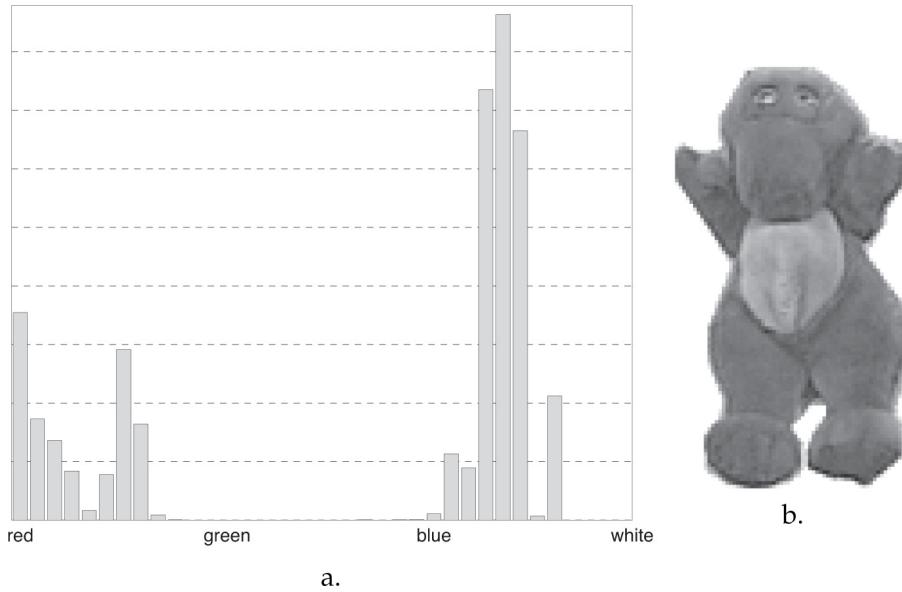


Figure 10.10 a.) A histogram for b.) the image of the children’s toy, Barney®.

A color histogram in RGB or any other distributed color space is a bit harder to visualize. The grayscale and hue image histograms had only one axis for buckets because these images have only one plane that matters. But a color image has three planes in the RGB coordinate system. As a result, it has buckets for each color plane or axis. Assuming that each plane is divided into eight buckets, the first bucket would be the number of pixels which fell into the range of (R, G, B) of (0-31, 0-31, 0-31).

The real advantage of a color histogram for reactive robots is that color histograms can be subtracted from each other to determine if the current image (or some portion), I , matches a previously constructed histogram, E . The histograms are subtracted bucket by bucket (j buckets total), and the differences indicate the number of pixels that did not match. The number of mismatched pixels divided by the number of pixels in the image gives a percentage match. This is called the histogram intersection:

$$(10.1) \quad \text{intersection} = \frac{\sum_{j=1}^n \min(I_j - E_j)}{\sum_{j=1}^n E_j}$$

For example, a robot can “wake up” and imprint the object in front of it by constructing the color histogram. Then a perceptual schema for a releaser or behavior can compute the color histogram intersection of the current image with the imprint. The robot can use the color histogram to determine if a particular object is of interest or not.

Because the color histogram of a current image can be matched with another image, the technique appears to be model-based, that is, a form of recognition. But reactive systems do not permit recognition types of perception. Is this a contradiction? No, a color histogram is an example of a local, behavior-specific representation which can be directly extracted from the environment. For example, a robot could be shown a Barney doll with its distinct purple color with green belly as the percept for the

goal for a move-to-goal behavior. However, the robot will follow a purple triangle with a green region because the ratio of colors is the same. There is no memory and no inference, just a more complex stimulus.

Note that the intersection can be considered to be a measure of the strength of the stimulus, which is helpful in reactive robotics. In one set of experiments, a robot was presented a poster of Sylvester and Tweety ([figure 10.11](#)). It learned the histogram and then, after learning the object (e.g., fixating on it), it would begin to move towards it, playing a game of tag as a person moved the poster around. The robot used a simple attractive potential fields-based move-to-goal behavior, where the perceptual schema provided the location of the poster and the percent of intersection. The motor schema used the location to compute the direction to the poster, but the intersection influenced the magnitude of the output vector. If the person moved the poster into a dark area or turned it at an angle, the intersection would be low and the robot would move slower. If the match was strong, the robot would speed up. Overall, it produced a very dog-like behavior where the robot appeared to play tag quickly (and happily) until the human made it too difficult. Then, if the human moved the poster back to a more favorable position, the robot would resume playing with no hard feelings.



Figure 10.11 A Denning mobile robot using a color histogram to play tag with a poster of Sylvester and Tweety.

10.6 Choosing Sensors and Sensing

A designer has to commit to a specific set of sensors and algorithms. In doing so, it is helpful to be aware of three concepts. The first concept is the idea of logical or equivalent sensors, that it may be possible to generate the same percept from different sensors or algorithms. The second is behavioral

sensor fusion, which describes the general methods of combining sensors to get a single percept or to support a complex behavior. The third concept is that attributes of a sensor suite that can be used to help design a system.

10.6.1 Logical Sensors

LOGICAL SENSOR

A powerful abstraction of sensors is *logical sensors*, first introduced by Henderson and Shilcrat.⁸⁸ A logical sensor is a unit of sensing, or module, that supplies a particular percept. It consists of the signal processing from the physical sensor and the software processing needed to extract the percept; it is the functional building block for perception. A logical sensor can be easily implemented as a perceptual schema.

LOGICAL EQUIVALENCE

An overlooked aspect of a logical sensor is that it contains all available alternative methods, or schemata, for obtaining that percept. For example, a percept commonly used for obstacle avoidance is a polar plot of range data. The logical sensor for the percept might be named `range_360` and return a data structure or object specifying the polar plot. The logical sensor would go further and list all the possible ways the robot had for constructing a polar plot of that form. The robot might be able to use sonar, a laser, stereo vision, or texture. Each of those modules would be. *logically equivalent*; that is, they would return the same percept data structure so they can be used interchangeably. However, they would not necessarily be. Therefore, the logical sensor contains a selector function which specifies the conditions under which each alternative is useful and therefore should be selected.

A powerful practical use of logical sensors or logical equivalence is that the world model can serve as the virtual sensor. In some hierarchical systems architectures, all sensing is used to generate a global world model. The architecture may have behaviors, but the perceptual input is a perceptual schema that is extracting the percept from the world model, not an actual sensor.

PHYSICAL REDUNDANCY

There are two types of redundancy. *Physical redundancy* means that there are several instances of physically identical sensors on the robot. [Figure 10.12](#) shows a robot with redundant cameras. In this case, the cameras are mounted 180° apart, and when one sensor fails, the robot has to “drive backwards” in order to accomplish its task.

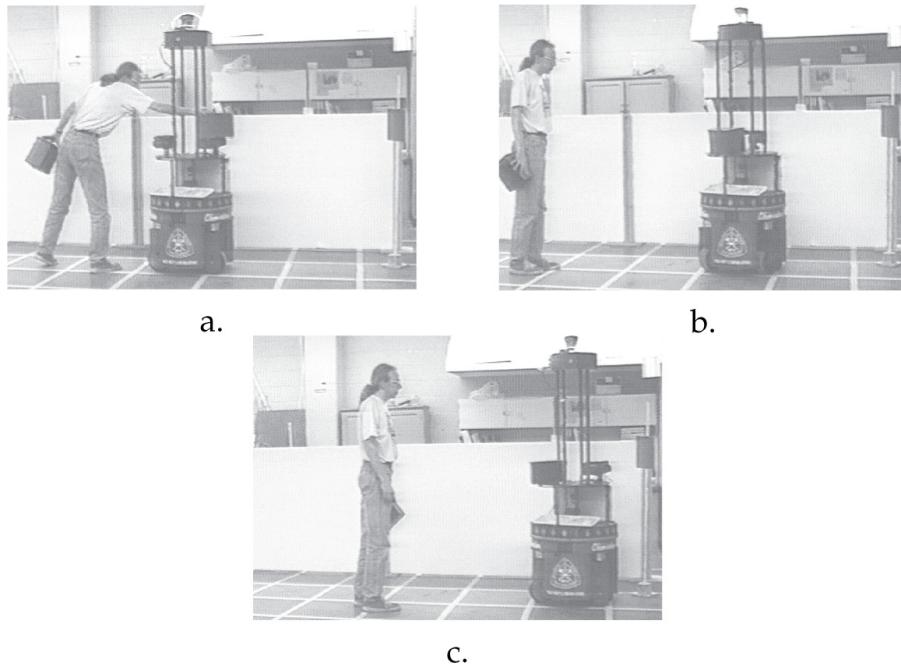


Figure 10.12 Sequence showing a Denning mobile robot with redundant cameras, responding to sensing failures introduced by Dave Hershberger.

LOGICAL REDUNDANCY

Logical redundancy means that another sensor, using a different sensing modality, can produce the same percept or releaser. For example, the Mars Sojourner mobile robot had two range sensors, a stereo vision pair and a laser striping system for determining the range to obstacles. The sensors are not physically redundant, but they produce the same overall information: the location of obstacles relative to the robot. However, logically redundant sensors are not necessarily equivalent in processing speed or accuracy and resolution. The stereo range sensor and algorithm computed the range much slower than a laser striping system.

FAULT TOLERANCE

Physical redundancy introduces new issues which are an area of active research investigation. Possibly the most intriguing is how a robot can determine that a sensor (or algorithm) has failed and needs to be swapped out. Surviving a failure is referred to as *fault tolerance*. A robot can be programmed in most cases to tolerate faults as long as it can identify when they occur.

10.6.2 Behavioral Sensor Fusion

SENSOR FUSION

REDUNDANT

COMPLEMENTARY

COORDINATED

Sensor fusion is a broad term used for any process that combines information from multiple sensors into a single percept. The motivation for sensor fusion stems from three basic combinations of sensors: *redundant* (or competing), *complementary*, and *coordinated*. Although many researchers treat sensor fusion as a means of constructing a global world model in a hierarchical or deliberative system, sensor fusion can be incorporated into behaviors through *sensor fission*, *action-oriented sensor fusion*, and *sensor fashion*.

FALSE POSITIVE

FALSE NEGATIVE

The motivation for using multiple sensors is that one sensor may be too imprecise or noisy to give reliable data. Adding a second sensor can give another “vote” for the percept. When a sensor leads the robot to believe that a percept is present, but it is not, the error is called *false positive*. The robot has made a positive identification of percept, but it was false. Likewise, an error where the robot misses a percept is known as a *false negative*. Sensors will often produce different false positive and false negative rates. Whether a robot can tolerate a higher false positive or false negative rate depends on the task.

Work in digital signal processing (DSP) treats the problem of combining sensors as a problem in registering or enforcing similarities in the data streams. For example, the sensors may be covering slightly different areas and thus the DSP problem is to align the sensor readings with each other. One sensor may be producing data more frequently than another. The idea is to get the data streams aligned in space and time so that they can be summed.

REDUNDANT

PHYSICAL REDUNDANCY

LOGICALLY REDUNDANT

COMPETING SENSORS

Behavioral sensor fusion takes a different approach. When the sensors are both returning the same percept, the sensors are considered *redundant*. An example of *physical redundancy* is shown in [figure 10.13](#), where a Nomad 200 has two sonar rings. The sonar software returns the minimum reading (shortest range) from the two rings, providing a more reliable reading for low objects which would ordinarily specularly reflect the beam from the upper sonar. Sensors can also be *logically redundant*, where they return identical percepts but use different modalities or processing algorithms. An example is extracting a range image from stereo cameras and from a laser range finder. Sometimes redundant sensors are called *competing sensors* because the sensors can be viewed as competing to post the “winning” percept.

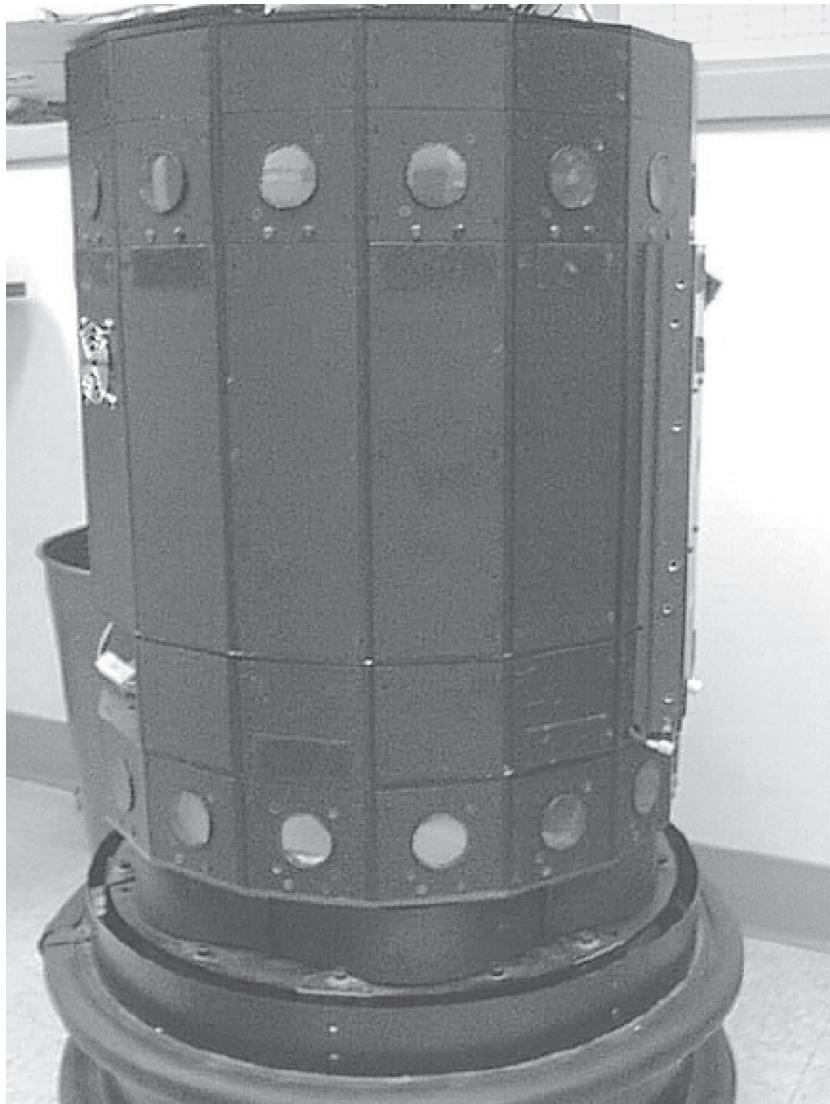


Figure 10.13 Example of redundant top and bottom sonar rings.

Complementary sensors provide disjoint types of information about a percept. In behavioral sensor fusion for urban search and rescue, a robot may search for survivors by fusing observations from a thermal sensor for body heat with those from a camera detecting motion. Both logical sensors return some aspect of a “survivor,” but neither provides a complete view.

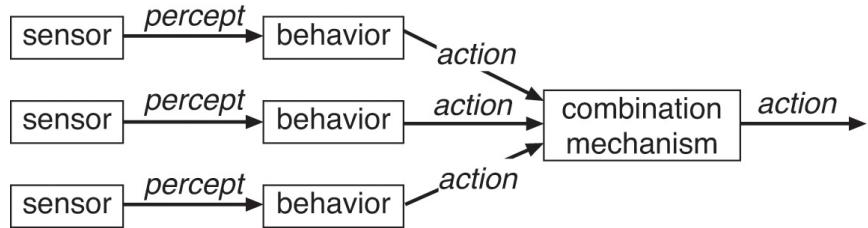
SENSOR FASHION

Coordinated sensors use a sequence of sensors, often for cueing or providing focus-of-attention. A predator might see motion, causing it to stop and examine the scene more closely for signs of prey. Traditional sensor fusion and DSP approaches leave the category of coordinated sensors untouched. Arkin filled in the apparent gap by calling that type of coordination *sensor fashion*,¹¹ with “fashion”

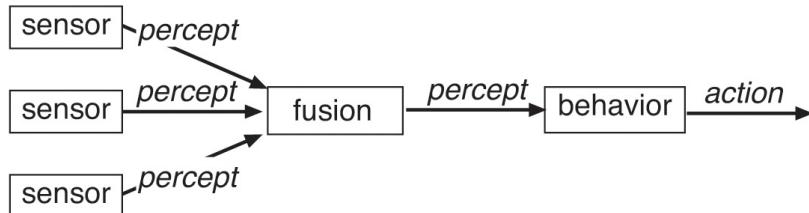
alliterative with “fusion” and intending to imply the robot was changing sensors with changing circumstances, just as people change styles of clothes with the seasons.

SENSOR FISSION

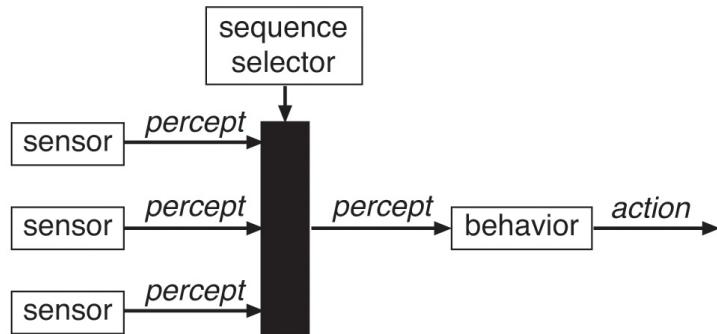
Most of the work treats sensor fusion as if it were a deliberative process: one that requires a global world model. Early work in reactive systems used robots with only a few simple sensors, a sonar or sonar ring for range and a camera for color, texture, or motion affordances. As a result, there was a design philosophy of using one sensor per behavior. Behaviors could share a sensor stream, but without knowing it. This philosophy led to the approach taken by Brooks that sensor fusion at the behavioral level was a mirage; it was really a combination of multiple instances of the same behavior, each with different sensor inputs. To an outside observer, it would look like some complicated process was being enacted inside the robot, but, in fact, it would be a simple competition with an emergent behavior. Brooks dubbed this *sensor fission* in part as a take-off on the connotation of the word “fusion” in nuclear physics. In nuclear fusion, energy is created by forcing atoms and particles together, while in fission, energy is creating by separating atoms and particles. [Figure 10.14a](#) shows a diagram of sensor fission.



a.



b.



c.

Figure 10.14 Three types of behavioral sensor fusion: a.) sensor fission, b.) action-oriented sensor fusion, and c.) sensor fission.

ACTION-ORIENTED SENSOR FUSION

Murphy reported on studies from cognitive psychology and neurophysiology showing that behavioral sensor fusion does occur in animals, and therefore should be part of a robot's behavioral repertoire.¹⁴⁹ The sensor pathways throughout the brain remain separate and can be routed to multiple behaviors in the superior colliculus. Only when the sensor signals routed to the portion of the brain associated with a particular behavior arrive at that location does there appear to be any transformation into a new combined representation. Any or all of these sensor streams for a behavior can be active and influence the resulting behavior. For example, consider the predation behavior in cats. If a cat hears a

noise and sees a movement, it will react more strongly than if it receives only a single stimulus. This type of sensor fusion is called *action-oriented sensor fusion* in order to emphasize that the sensor data are being transformed into a behavior-specific representation in order to support a particular action, not for constructing a world model. [Figure 10.14b](#) shows a diagram of action-oriented sensor fusion.

Sensor fission and action-oriented sensor fusion cover competing and complementary sensing. Sensor fission is, by definition, a competitive method, though complementary sensors may be used to support a particular instance of a behavior. Action-oriented sensor fusion is not restricted to either competing or complementary sensors, since the behavior makes a local transformation anyway. A diagram of sensor fashion is shown in [Figure 10.14c](#).

10.6.3 Designing a Sensor Suite

Historically, reactive robots used either inexpensive IR or ultrasonic transducers to detect range. Now they have moved to using RGB-D cameras or depth-from-X algorithms. The earliest behaviors focused on basic navigational skills, such as obstacle avoidance and wall following. The percept for these behaviors involve knowing the distance to an occupied area of space. Now with the advent of inexpensive miniature cameras and laser range finders for consumer applications, the use of computer vision is becoming increasingly common. In agricultural and transportation applications of reactive robots, GPS technology has become popular as well. This chapter attempts to cover the basics of these sensing modalities and how they are used in mobile robots. Because the sensor market is rapidly changing, the chapter will focus on how to design a suite of sensors for use by a robot, rather summarize the device details.

PROPIROCEPTION

EXTEROCEPTION

EXPROPRIOCEPTION

An artificially intelligent robot has to have some sensing in order to be considered a true AI robot. If it cannot observe the world and the effects of its actions, it cannot react. As noted in the chapter on “Action-Oriented Perception” in *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems*, the design of a set of sensors for a robot begins with an assessment of the type of information that needs to be extracted from the environment.¹⁰⁸ This information can be obtained from *proprioception* (measurements of movements relative to an internal frame of reference), *exteroception* (measurements of the layout of the environment and objects relative to the robot’s frame of reference), or *exproprioception* (measurement of the position of the robot body or parts relative to the layout of the environment).

The Colorado School of Mines fielded an entry to the 1995 UGV competition; the 1994 competition is discussed in chapter 19. This entry provides an example of a different type of sensing for a path-following robot. In 1995, the follow-path behavior was expanded to track both lines of the path using a wide angle lens on the camera. *follow-path* could be considered exteroceptive because it acquired information about the environment. However, the camera for the robot was mounted on a panning mast, which was intended to turn to keep the path in view, no matter in what direction the path turned. Therefore, the robot had to know where the camera was turned relative to the robot’s internal frame of reference in order to correctly transform the image coordinates of the location of white lines to a

steering direction. This meant the information needed for follow-path had to have both proprioceptive and exteroceptive components, making the perception somewhat exproprioceptive. (If the robot was extracting the pose of its camera from exteroception, it would be clearly exproprioceptive.)

Due to a programming error, the follow-path behavior incorrectly assumed that the exteroceptive camera data had been transformed by the proprioceptive shaft encoder data from the panning mast into exproprioceptive data. The robot needed the exproprioception to determine where it should move next: turn to follow the path in camera coordinates, plus compensation for the current camera pan angle. The programming error resulted in the robot acting as if the camera were aligned with the center of the robot at all times. But the camera might be turned slightly to maintain the view of both lines of the path through the pan-camera behavior. The resulting navigational command might be to turn, but not enough to make a difference, or even to turn the wrong way. This subtle error surfaced as the robot went around hair-pin turns, causing the robot to go out of bounds consistently.

SENSOR SUITE

As can be seen from the above example, robots may have dead reckoning capabilities, but will always have some type of exteroceptive sensor. Otherwise, the robot cannot be considered reactive: there would be no stimulus from the world to generate a reaction. The set of sensors for a particular robot is called a *sensor suite*. Following *Sensors for Mobile Robots*,⁷⁴ in order to construct a sensor suite, the following attributes should be considered for each sensor:

1. Field of View and Range. Every exteroceptive sensor has a region of space that it is intended to cover. The width of that region is specified by the sensor's *field of view*, often abbreviated as FOV. The field of view is usually expressed in degrees; the number of degrees covered vertically may be different from the number of degrees covered horizontally. Field of view is frequently used in photography, where different lenses capture areas of different size and shape. A wide angle lens will often cover up to 70°, while a "regular" lens may have a field of view of only around 27°. The length of the field is called the range.

FIELD OF VIEW (FOV)

HORIZONTAL FOV

VERTICAL FOV

RANGE

The *Field of View (FOV)* can be thought of in terms of egocentric spherical coordinates, where one angle is the *horizontal FOV* and the other is the *vertical FOV*. The other aspect is the *range*, or how far the sensor can make reliable measurements. In spherical coordinates, these measurements would be the values of r that defined the depth of the operating range.

Field of view and range are obviously critical in matching a sensor to an application. If the robot needs to be able to detect an obstacle when it is eight feet away in order to safely avoid it, then a sensor with a range of five feet will not be acceptable.

RESOLUTION

2. Accuracy, Repeatability, and Resolution. Accuracy refers to how correct the sensor reading is. But if a reading for the same condition is accurate only 20% of the time, then the sensor has little

repeatability. If the sensor is consistently inaccurate in the same way (always 2 or 3 cm low), then the software can apply a bias (add 2 cm) to compensate. If the inaccuracy is random, then it will be difficult to model, and the applications where such a sensor can be used will be limited. If the reading is measured in increments of one meter, that reading has less *resolution* than a sensor reading which is measured in increments of one cm.

3. Responsiveness in the target domain. Most sensors have particular environments in which they function poorly. The environment must allow the signal of interest to be extracted from noise and interference (e.g., have a favorable signal-to-noise ratio). As will be seen below, sonar is often unusable for navigating in an office foyer with large amounts of glass windows and partitions because the glass reflects the sound energy in ways almost impossible to predict. It is important to have characterized the ecological niche of the robot in terms of what will provide, absorb, or deflect energy.

4. Power consumption. Power consumption is always a concern for robots. Since most robots operate on batteries, the less power they consume, the longer they run. For example, the battery life on a Nomad 200, which carries five batteries, was improved from four hours to six by shutting off all sensors. Power is so restricted on most mobile robots that many robot manufacturers will swap to more energy-efficient microprocessor chips just to reduce the power drain. Sensors which require a large amount of power are less desirable than those which do not. In general, passive sensors have lower power demands than active sensors because passive sensors are not emitting energy into the environment.

HOTEL LOAD

LOCOMOTION LOAD

The amount of power on a mobile robot required to support a sensor package (and any other electronics, such as a microprocessor and communications links) is sometimes called the *hotel load*. The sensor suite is the “guest” of the platform. The power needed to move the robot is called the *locomotion load*. Unfortunately, many robot manufacturers focus only on the locomotion load, balancing power needs with the desire to reduce the overall weight and size. This leads to a very small hotel load and often prevents many sensors from being added to platform.

5. Reliability. Sensors often have physical limitations as to how well they work. For example, Polaroid® sonars will produce incorrect range reading when the voltage drops below 12V. Other sensors have temperature and moisture constraints which must be considered.

6. Size. The size and weight of a sensor does affect the overall design. A microrover on the order of a shoebox will not have the power to transport a large camera or camcorder, but it may be able to use a miniature “Quick-Cam” type of camera.

The above list concentrated on considerations of the physical aspects of the sensor. However, the sensors only provide observations; without the software perceptual schemas, the behaviors cannot use the sensors. Therefore, the software that will process the information from a sensor must be considered as part of the sensor selection process.

7. Computational complexity. Computational complexity is the estimate of how many operations an algorithm or program performs. Computational complexity is often written in function notation as O , called the “order,” where $O(x)$ means the number of major operations is proportional to x . For example, an algorithm that scrolls through a list of 100 numbers can be thought of having 100 operations. If the same algorithm is given a list of 1,000 numbers, the number of operations would be

1,000; the number of operations depends on the size of the input, which is x . x is often a function itself and n is used instead of x to denote a function. Lower orders are better. An algorithm that executes with $O(n)$ equally consuming operations is faster than one with $O(n^2)$ operations. (If you doubt this, see if you can find a positive, whole number value of n such that $n \leq n^2$.) Also, if the number of inputs to an $O(n)$ algorithm is doubled, the complexity is linearly proportional; the algorithm will take twice as long to execute. But if the number of inputs to an $O(n^2)$ algorithm is doubled, the algorithm will take four times as long to execute. Computational complexity has become less critical for larger robots with the rapid advances in processors and miniaturization of components. However, it remains a serious problem for smaller unmanned systems, especially since computer vision algorithms can be $O(m^4)$.

8. Reliability. The designer should consider how reliable the sensor will be for the ecological conditions and for interpretation. The robot will often have no way of determining when a sensor is providing incorrect information. As a result the robot may “hallucinate” (think it is seeing things that are not there) and do the wrong thing. Many sensors produce output which is hard for human to interpret without years of training; medical X-rays are one example, and synthetic aperture radar (SAR), which produces polar plots, is another. If a sensor algorithm is not working properly in these modalities, the designer might not be skilled enough to notice it. Therefore, the algorithms themselves must be reliable.

10.7 Summary

The success of a robot depends on the suitability of its sensors and sensing. It is often more useful to think of sensing in terms of perceptual schemas or logical sensors needed to accomplish a task rather than focus on the characteristics of a particular transducer or modality.

Almost all mobile robots have some form of proprioception, most likely shaft or wheel encoders used to estimate the distance traveled based on the number of times the motor has turned. Outdoor robots generally use GPS, but GPS accuracy and reliability may be greatly reduced in urban canyons and be non-existent indoors. Thus navigation in GPS-denied areas remains a challenge. Range sensors, discussed in the next chapter, do not entirely solve this challenge.

Due to the low price and availability of consumer electronics and the usual need to have a camera on the robot for backup teleoperation, computer vision is becoming more common in robotic systems. Computer vision processing operates on images, regardless of the modality which generated it. Color-coordinate systems tend to divide an image into three planes. The two most common color coordinate systems are RGB and HSV. HSV treats color in absolute terms, but RGB is favored by equipment manufacturers. A color space used in biomedical imaging, SCT, appears to be less sensitive to lighting conditions than are RGB and RGB-derived HSV. Many reactive robots exploit color as an affordance by thresholding an image and identifying regions of the appropriate color. A color affordance method, which works well for objects with multiple colors, is color histogramming.

A designer has to commit to a specific set of sensors and algorithms. In doing so, it is helpful to be aware of three concepts. The first concept is the idea of logical or equivalent sensors, that it may be possible to generate the same percept from different sensors or algorithms. The second is behavioral sensor fusion, which describes the general methods of combining sensors to get a single percept or to support a complex behavior. The third concept is understanding the attributes of a sensor suite that can

be used to help design a system.

Returning to the questions posed in the introduction, the answer to *How do you make a robot “see”?* is that it depends on the sensors, which provide data, and the sensing algorithms, which extract percepts or generate world models from that data. Another question is: *What sensors are essential for a robot?* The specific answer depends on the mission, of course, but, in general, robots will need both proprioception to keep track of the pose of its physical body and exteroception to navigate and recognize objects. Since people tend to think of perception in terms of eyes, a teleoperated robot will generally have a visible light camera to make it more natural for the teleoperator to control the robot. Exploiting the use of camera(s) is the purview of the field of computer vision.

The chapter has also addressed: *What is sensor fusion?* It has discussed sensor fusion, or the mechanisms which allow multiple sensors to produce percepts and models. Unlike digital signal processing, sensor fusion in artificial intelligence does not assume that the sensor data must be registered and sampled at the same instance in time; instead AI methods explore both redundant and complementary modalities of perception.

Despite the diversity of sensors, robotics is remarkable for its lack of sophistication in sensing. This may stem from the split between computer vision and robotics in the formative years of the field. The reader is encouraged to explore the computer vision literature thoroughly rather than try to reinvent the wheel.

10.8 Exercises

Exercise 10.1

Define sensor suite, active sensors, passive sensors, dead reckoning, and computer vision.

Exercise 10.2

What are the sensors used for in a fully autonomous car, such as STANLEY, which won the 2005 DARPA Grand Challenge, or a Tesla?

Exercise 10.3

Compare and contrast the RGB and HSV color representations specifying the advantages and disadvantages of each type.

Exercise 10.4

True or false: Hue is what we think of as the invariant color, and it can reliably be extracted from standard RGB digital imagery.

Exercise 10.5

Ultrasonic sensors have many positive and negative attributes. Name and describe three positive and three negative attributes.

Exercise 10.6

What is the difference between physical and logical redundancy?

Exercise 10.7

Describe, with examples, the three attributes that should be considered for an entire sensing suite.

Exercise 10.8

Consider a Lego Mindstorms robot. Classify the available sensors for it in terms of modalities.

Exercise 10.9

Can a world model be used as a sensor?

Exercise 10.10

Which is NOT a major concern in designing a sensor suite?

- a. field of view, range of the sensor
- b. accuracy and repeatability
- c. responsiveness in the target domain
- d. power consumption
- e. size
- f. be geo-tagged.

Exercise 10.11

List the metrics for rating individual sensors and a sensor suite, and apply these metrics to a particular application.

Exercise 10.12

List and describe advantages and disadvantages of three different sensors, including one type of computer vision.

Exercise 10.13

Describe all the different characteristics of sensors that must be evaluated when designing a sensor suite. In addition, give priorities for each to determine which you would consider to be the most and least important for a robot that was going to be designed for the 1994 AUVS Unmanned Ground Robotics Competition.

Exercise 10.14

Pick a sensor suite that you think would do a good job if you were designing a robot for the 1995 UGV competition described in an earlier chapter. Explain what each sensor would do and describe the sensors and sensor suite in terms of the attributes listed in this chapter.

Exercise 10.15

You are to design a sensor suite for a new robot for use by fire fighters. The robot is designed to seek out people in a smoke-filled building. Keep in mind the following:

1. Visibility is often very limited due to smoke.
2. Heat can be both an attractive force (e.g., human) or repulsive (e.g., open flame).
3. Obstacles may have a wide variety of sound absorption (e.g., carpeting or furniture).

Describe the types of sensors that may be needed and how they will be used. Do not focus on how the robot moves around. Concentrate on the sensors it will need to use. Extra credit: Comment on using simple voice recognition software and a microphone to seek out human voices (e.g., cries for help).

Exercise 10.16

How are the concepts of logical sensors in robotics and polymorphism in object-oriented programming similar?

Exercise 10.17

Define *image function*. What is the image function for

- a. the left-right images in a stereo pair?
- b. the depth map?

Exercise 10.18

What are two disadvantages of light stripers?

Exercise 10.19

Consider an obstacle avoidance behavior which consists of a perceptual schema that provides a polar plot of range and motor schema which direct the robot to the most open sector. List all the logical sensors covered in this chapter that can be used interchangeably for the perceptual schema. Which of these are logically redundant? Physically redundant?

Exercise 10.20

Assume you had a mobile robot about 0.5 meters high and a planar laser range finder. At what angle would you mount the laser if the robot was intended to navigate

- a. in a classroom?
- b. in a hallway or reception area where the primary obstacle is people?
- c. outdoors in unknown terrain?

State any assumptions your design is based on. Is there more information needed; if so, what?

Exercise 10.21

What is the difference between *locomotion load* and *hotel load*?

Exercise 10.22

Why is insufficient hotel load a problem for the design of intelligent robots?

[*Programming*]

Exercise 10.23

Write your own code to threshold a small interleaved image.

[*World Wide Web*]

Exercise 10.24

Search for Kludge, another robot built by Ian Horswill. Describe Kludge's homemade whisker system and discuss how well it works.

[*Programming*]

Exercise 10.25

Program a mobile robot to find and bump an orange tennis or soccer ball.

[*Programming*]

Exercise 10.26

Create a color histogram program. Construct the color histogram, E , for four different brightly colored objects, such as South Park dolls or the Simpsons cartoon characters. Present the program with a different image, I , of one of the characters and compute the histogram intersection with each of the four E s. Does the highest match correctly identify the character? Why or why not?

10.9 End Notes

For the roboticist's bookshelf.

Hobart Everett literally wrote the book, *Sensors for Mobile Robots*,⁷⁴ on robotic sensors, which provides both analytical details of sensors plus practical experience based on Everett's many years with the Navy, working on robots. He has built a series of mobile robots called ROBART (a pun on Everett's nickname, Bart); ROBART II has been in continuous operation since 1982. Everett's laboratory has to be one of the most ideally situated in the world. It is in San Diego, overlooks the Pacific Ocean, and is adjacent to a frequently used volleyball court.

Hardware, software, Pixar®.

Pixar, famous for its animated movies, actually started out building the computer hardware for high-resolution image processing.

Hans Moravec.

If Joe Engelberger is known as the father of industrial robotics, Hans Moravec is known as the father of AI robotics. He has also become a well-known author, arguing for the inevitability of machine intelligence in controversial books such as *Mind Children*¹³⁴ and *Robot: mere machine to transcendent mind*.¹³⁵ His work with the Stanford Cart was a catalyzing event in attracting attention to robotics after years of slow progress flowing Shakey. Documentaries occasionally run edited footage of the Stanford Cart navigating outdoors, avoiding obstacles. Since the cart traveled in a stop-start fashion, with 15 minutes or so between updates, the location of the shadows noticeably change. Moravec's office mate, Rodney Brooks, helped with the recording.

Ballard and Brown.

Dana Ballard and Chris Brown wrote the first textbook on computer vision, entitled appropriately *Computer Vision*. Both have worked with robots. I met both of them at a workshop at a beach resort in Italy in the early 1990. I had just arrived from a harrowing bus ride on narrow mountain roads from the airport and had to go past the pool to my room in the hopes of recovering from jet lag before the meeting started in the morning. As I walked past the pool, one of my friends said, "Oh, Chris Brown is over there!" I immediately turned to look for what I thought would be an older, dignified author wearing a suit and tie. Instead, I got soaked by a tall, youngish man impishly doing a cannonball into the pool. From that day on, I never assumed that textbook authors were dull and dignified.

The tallest roboticist.

Tom Henderson at the University of Utah was one of the founders of the concept of logical sensors. Henderson is also the tallest known roboticist; he played basketball in college for Louisiana State University.

Run over Barney.

The figures and materials on color histogramming used in this chapter were part of research work conducted by Dale Hawkins in persistence of belief. His use of a stuffed Barney doll started out from a class project: program a mobile robot to find a Barney doll and run over it. This is actually a straightforward reactive project. The Barney doll is a distinctive purple, making it easy for the vision system to find it. The project allowed the programmers to use the flat earth assumption, so trigonometry could be used to estimate the location to the doll, based on the location in image coordinates. Hawkins' program was the clear winner, running completely over Barney.

Reactive soccer.

Color regions are often used to simplify tracking balls (and other robots) in robot soccer competitions, such as RoboCup. One amusing aspect is that many of these behaviors are purely reflexive; if the robot sees the ball, it responds, but if it loses the ball, it stops. Ann Brigante and Dale Hawkins programmed a Nomad 200 to reflexively track a soccer ball to compare what would happen if the robot had some concept of object permanence. Because of the angle of the camera, the robot would lose sight of the ball when almost touching it. The behaviors that emerged worked but always generated much laughter. The robot would see the ball and accelerate rapidly to its estimated location to "kick" it. When it got to the ball, it suddenly decelerated but still had enough momentum to bump the ball. The ball would slowly roll forward, back into the now-stationary robot's field of view. The robot would again jump forward, and the cycle would repeat endlessly.

Photographs and scanning.

Dale Hawkins, Mark Micire, Brian Minten, Mark Powell, and Jake Sprouse helped photograph robots, sensors, and demonstrations of perceptual behaviors.

11

Range Sensing

Chapter Objectives:

- Describe the problems of *specular reflection*, *cross-talk*, and *foreshortening* with an ultrasonic transducer, and, if given a 2D line drawing of surfaces, illustrate where each of these problems would likely occur.
- Describe the *point cloud* knowledge representation and the types of algorithms associated with it.
- Describe the two popular types of active range sensors for producing point clouds: *lidar* and *RGB-D cameras*.
- Be familiar with *Sense-Register-Reconstruct* sequence for processing point clouds.

11.1 Overview

In order to function intelligently in the world, robots need to be able to measure or extract depth or range. This leads to the question: *What is the difference between depth and range?* The terms “depth” and “range” are used synonymously in robotics to refer to the distance between the robot and a surface in the world. However, the computer vision community often uses “depth” to refer to small changes in distance that indicate the sides of objects for surface reconstruction and volumetric estimation of an object. High resolution sensors that can be used to determine the three-dimensional shape of an object are generally called depth sensors to differentiate those sensors from range sensors. The robotics community typically uses range as a measure of the distance to an obstacle or the elevation of terrain; the measurements from range sensors can be much lower resolution for navigation than for object manipulation.

This chapter also answers: *Why is it so hard for robots to sense depth and range?* Range can be sensed with either passive or active sensors. A passive range sensor is generally a single camera or a stereo pair of cameras. One approach is to use optic flow, as described in chapter 7.3.1. Another approach to calculating range is to extract depth from the image based on triangulation of features seen in two images or from cues in the image. With that approach, the challenge is finding features in the images that correspond to each other, usually with algorithms called *interest operators*. The most popular algorithm for detecting corresponding features is the *scale-invariant feature transform (SIFT)*.

Fortunately, there are many cues for distance and depth, for example, relative size of an object, and these form a field of computer vision known as *depth from X*, where *X* is the cue is being used. Active sensors generally “paint” a region of space with some type of emission. An active sensor can either directly measure the range or it can extract the depth. Active sensors, such as sonars and lidar, send a pulse of sound or light, respectively, into the environment and measure the time of flight for the signal to return. Since the speed of the energy emission is known, the range to the object is $\frac{1}{2}time_{flight} \times speed$. An active sensor can also paint the region with light and use the light pattern as the features to be observed on a single image. This eliminates the need to have multiple images, apply interest operators to match the features, and then extract depth from those features using stereo or other depth from X algorithms.

Historically, roboticists first investigated RGB cameras and camera pairs starting in the late 1960s by emulating biological vision, but the computational costs of algorithms exceeded computer capabilities of the time and there were few compelling successes. Laser range sensors were being developed for the Department of Defense, but the size and cost (more than \$100,000) were prohibitive. Fortunately, inexpensive ultrasonic sensors (\$30) used for autofocus of a Polaroid camera were co-opted. The sensors had very low spatial resolution and were noisy but they were cheap and small enough that a robot could carry an array or ring of them. A series of probabilistic algorithms and knowledge representations were explored to try to take multiple low resolution readings and produce high resolution range maps. These methods contributed the occupancy grid spatial representation and evidential reasoning methods that would later form the foundation of simultaneous localization and mapping (SLAM) algorithms discussed in chapter 15. The field shifted to laser range sensors when SICK, a German-based company specializing in sensors, introduced a highly accurate, high resolution laser ranger for surveying that produced a range map in the horizontal plane in cylindrical coordinates. Eventually the cost and size of laser rangers with range maps in three dimensions dropped, and rangers, especially the Velodyne® sensor, have become commonplace on outdoor robots. The introduction of high-resolution, egocentric range information led to a new knowledge structure, the *point cloud*. The point cloud may be mapped to an absolute coordinate system where every 3D region of space is marked as either occupied or empty. The map is essentially an *occupancy grid*, but the term point cloud now dominates.

Perhaps the biggest shift in range sensing over the years has been the development of RGB-D systems, where D stands for depth, such as Microsoft’s Kinect, which used inexpensive active and passive sensors with sophisticated stereo algorithms now computationally tractable on modern processors. No matter how much of a breakthrough Kinect was, the answer to *doesn’t the Microsoft Kinect solve everything?* is *no*, as the system is sensitive to environmental conditions and works only over short ranges. The Google Tango® project is another example of RGB-D sensors. While the consumer applications for such sensors have not emerged, RGB-D sensors have heavily influenced robotics.

This chapter covers the major classes of algorithms and knowledge representations for determining the range to an object or surface. It starts with passive stereo vision pairs followed by active ultrasonics. Although ultrasonics are rarely used due to their significant disadvantages that will be discussed later in this chapter, they did lead to the occupancy grid knowledge representation and to the use of evidential reasoning to build up certainty from noisy readings. That legacy persists today.

Computer vision pursued less computationally complex alternatives to stereo pairs, most notably depth from X algorithms. Light stripers, which project a light onto the scene and infer depth from the distortion in the straight line, were another attempt at low cost active sensing. Lidar is the most effective and accurate sensor for directly measuring range, but it remains the most expensive. RGB-D cameras are a reformulation of light stripers. Both lidar and RGB-D cameras produce extremely dense data sets which challenge computational complexity; these data sets are represented as point clouds which require additional processing to register with other sensors and to extract surfaces.

11.2 Stereo

STEREOPSIS

STEREO PAIR

Humans extract depth from vision. In most cases, though not always, depth perception is due to having two eyes and being able to triangulate as shown in [figure 11.1](#), also known as *stereopsis*. Using two cameras to extract range data is often referred to as *range from stereo*, *stereo disparity*, *binocular vision*, or just plain “stereo.” One way to extract depth is to try to superimpose a camera over each eye as in [figure 11.1a](#). Each camera finds the same point in each image, turns itself to center that point in the image, and then the relative angle is measured. The cameras are known as the *stereo pair*.

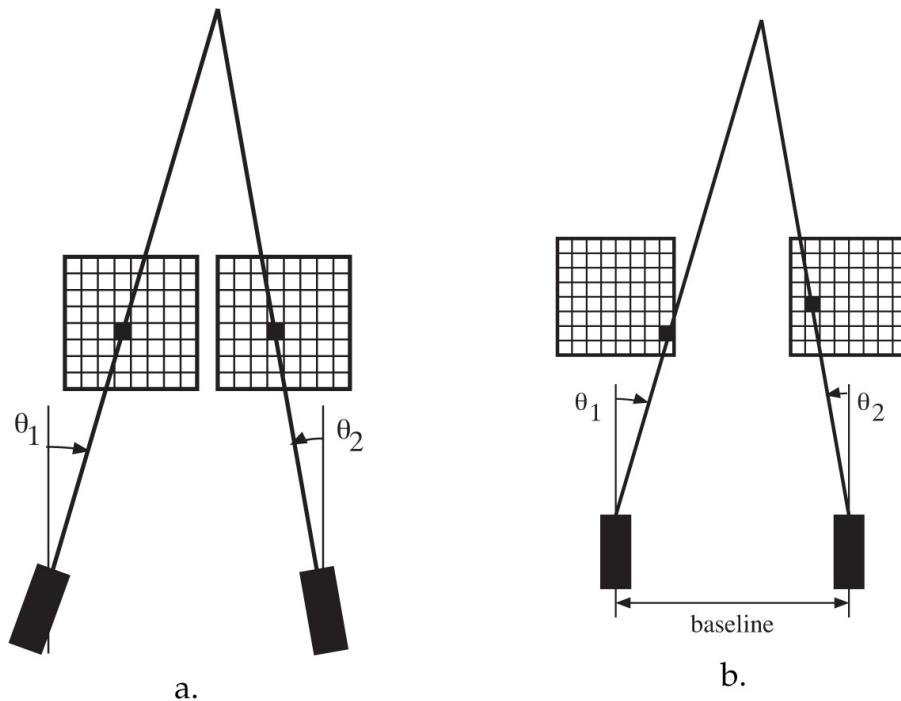


Figure 11.1 Ways of extracting depth from a pair of cameras: a.) vergence of the cameras to determine the depth of a point, and b.) a set of rectified stereo images.

CORRESPONDENCE

INTEREST OPERATOR

This method has two challenges. The first is that it is hard to design and build a mechanism which can precisely move to converge on the points. (It is even harder to design and build an inexpensive vergence mechanism.) The second challenge is even more fundamental and raises the question: *How does the robot know that it is looking at the same point in both images?* This challenge is referred to as the *correspondence* problem, since the task is to find a point in one image that corresponds to a point in the other image. A common approach is to identify “interesting,” or potentially uniquely valued, pixels in the image, such as very bright or dark spots or edges. The algorithm that selects interesting pixels is called an *interest operator*. Since even minute changes in lighting make a difference in the image, there is no guarantee that the two images, even though acquired at precisely the same time from two cameras, will “see” the same values for corresponding pixels. Therefore, interest operator algorithms usually return a list of interesting pixels, not just one, and a matching algorithm tries to find the best correspondence between all of them. After the interest points are established, the rest of the pixels have to be labeled with a depth relative to those points.

RECTIFIED IMAGES

DISPARITY

Fortunately, it is not necessary to have a mechanical vergence system. Instead, cameras can be mounted in place with the optic axes parallel to each other and perpendicular to the mount, producing *rectified images*.⁵⁵ This type of traditional stereo “head” is shown in [figure 11.2](#). The space between the axes of two cameras is called the *baseline*. The distance in the location of the point of interest between the images is called the *disparity*; the distance of the point from the cameras is inversely proportional to disparity.⁹² [Figure 11.1b](#) shows the geometry behind a stereo pair.

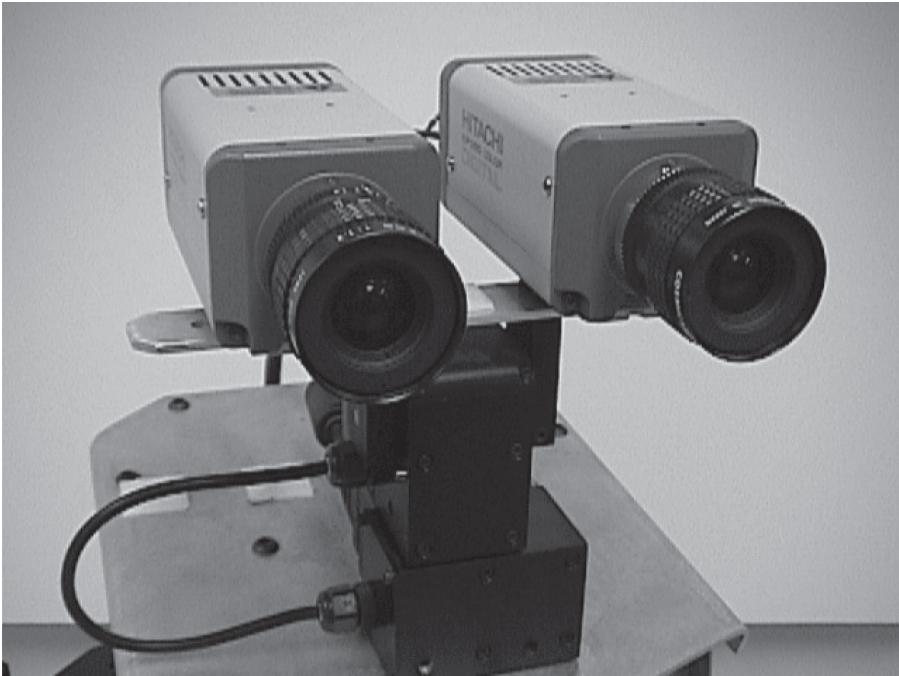


Figure 11.2 A stereo camera pair mounted on a pan/tilt head.

EPIPOLAR LINES

While rectified images eliminate the need for expensive mechanical vergence systems, they do not solve the correspondence problem. If the cameras are aligned precisely, then one row in the left image will correspond to a row in the right image. These rows are said to be *epipolar lines* or projections onto an *epipolar plane*. Whenever the robot finds an interesting point in one image, it has to consider only the pixels along the epipolar line in the other image. This is a tremendous computational savings. However, it works only if the cameras are perfectly matched optically and remain in alignment. In practice, robots move, bump, and suffer alignment drifts, and the cameras may have some flaws in their optics. The alignment can be periodically compensated for in software through a *camera calibration* process, where the robot is presented with a standard and then creates a calibration look-up table or function. [Figure 11.3](#) shows the Carnegie Mellon University Uranus robot calibrating its camera system. As a result, many researchers are turning to units which package a stereo pair in one fixed case, where the alignment cannot be altered. [Figure 11.6](#) shows the results of a stereo range system using three cameras in a fixed configuration.

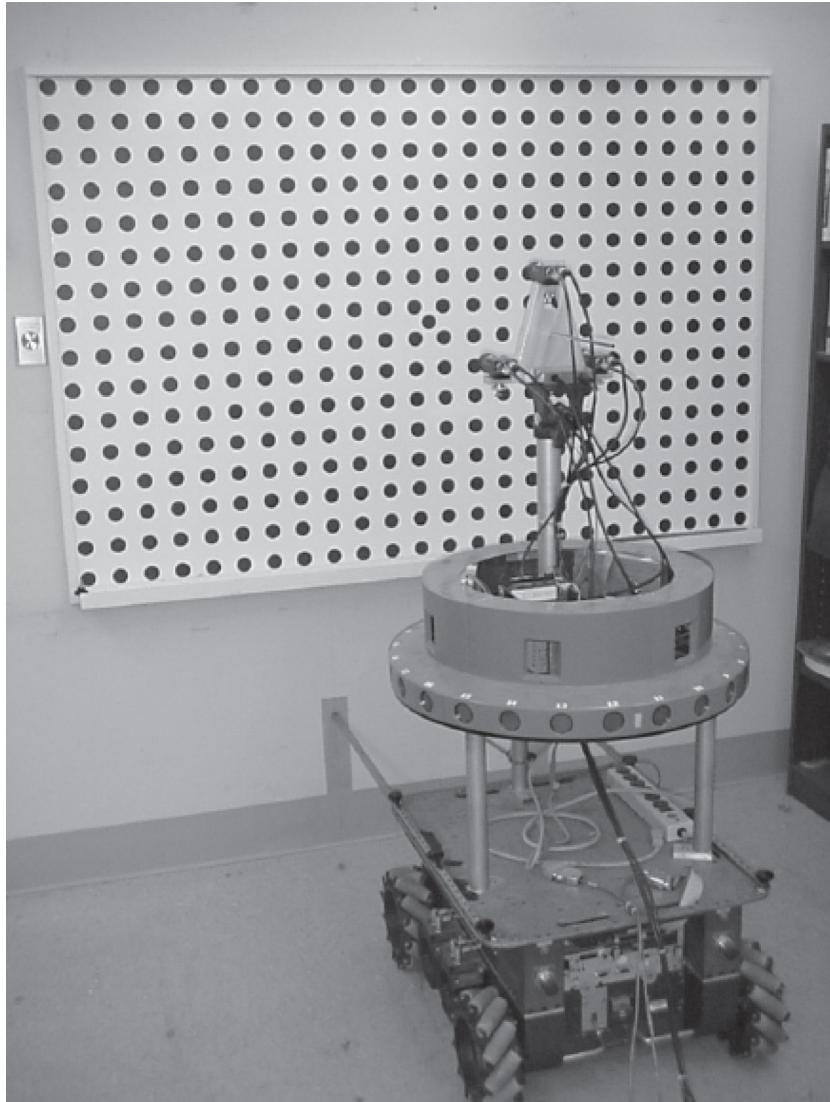
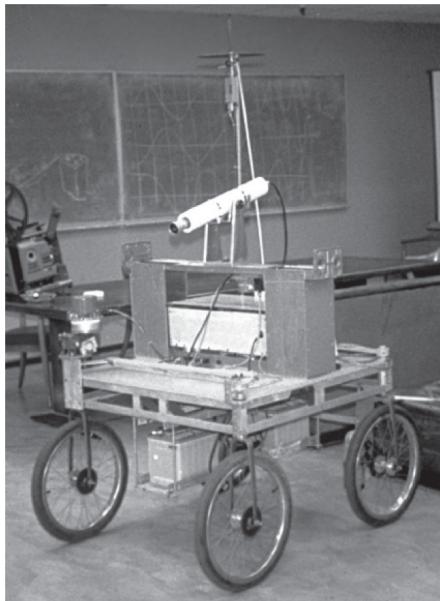


Figure 11.3 Uranus undergoing calibration. (Photograph courtesy of Hans Moravec.)

The first robot to use stereo vision successfully was Hans Moravec's Stanford Cart shown in [figure 11.4a](#). Moravec worked on the Cart while at graduate school at Stanford between 1973 and 1980. [Figure 11.4b](#) shows the Marsokhod rover developed in the late 1990s, which used a stereo pair for real-time navigation. Longer baselines tend to be more accurate because a slight measurement error has a smaller impact, but smaller baselines have a smaller "footprint," and in effect, take up less room. The same point in both images still has to be identified.



a.



b.

Figure 11.4 Robots and stereo: a.) The Stanford Cart developed in the late 1970s (Photograph courtesy of Hans Moravec.), and b.) The Marsokhod rover developed in the late 1990s jointly by scientists from McDonnell Douglas, Russia, and NASA Ames Research Center. (Image courtesy of the National Aeronautics and Space Administration.)

RANGE IMAGE

DEPTH MAP

[Figure 11.5](#) shows the simplified flow of operations in extracting a range from a pair of images. The process begins with two images, the left-right pair, and results in a third image called the *range image* or the *depth map*. The left-right pair can be grayscale or in color, but the depth map is a grayscale map, where intensity is proportional to the distance from the object to the cameras. [Figure 11.6](#) shows two stereo images and the resulting depth map.

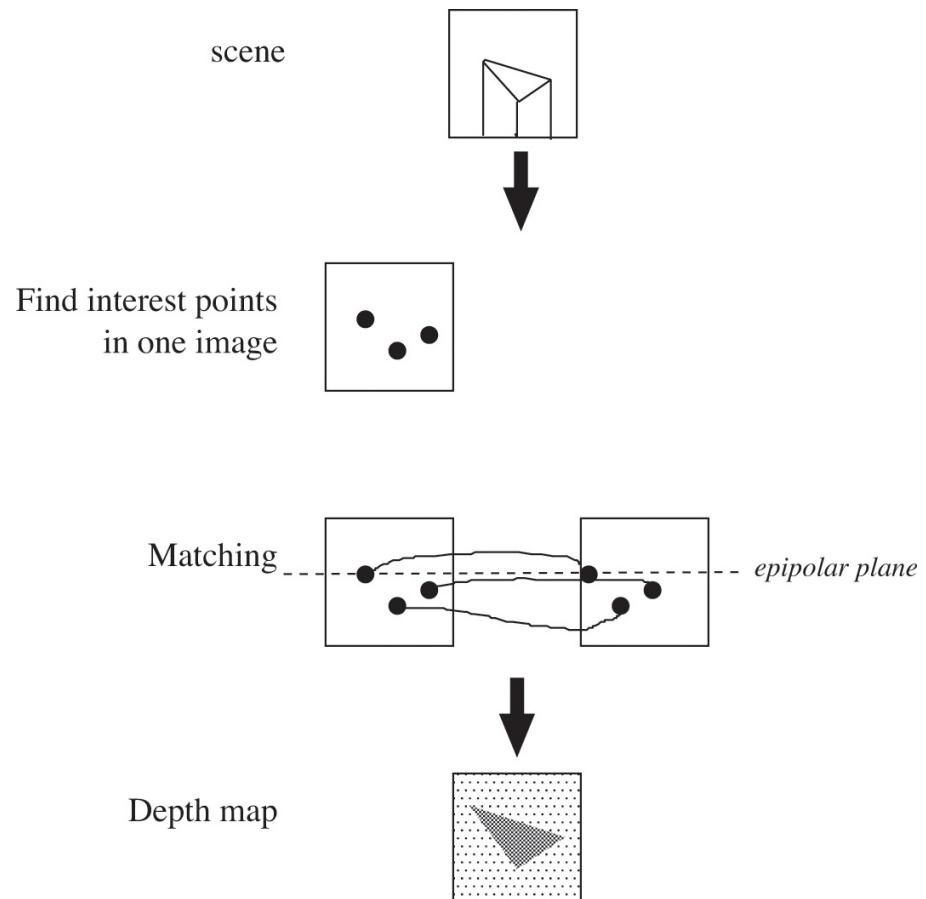


Figure 11.5 A simplified flow of operations in extracting range from a stereo pair.

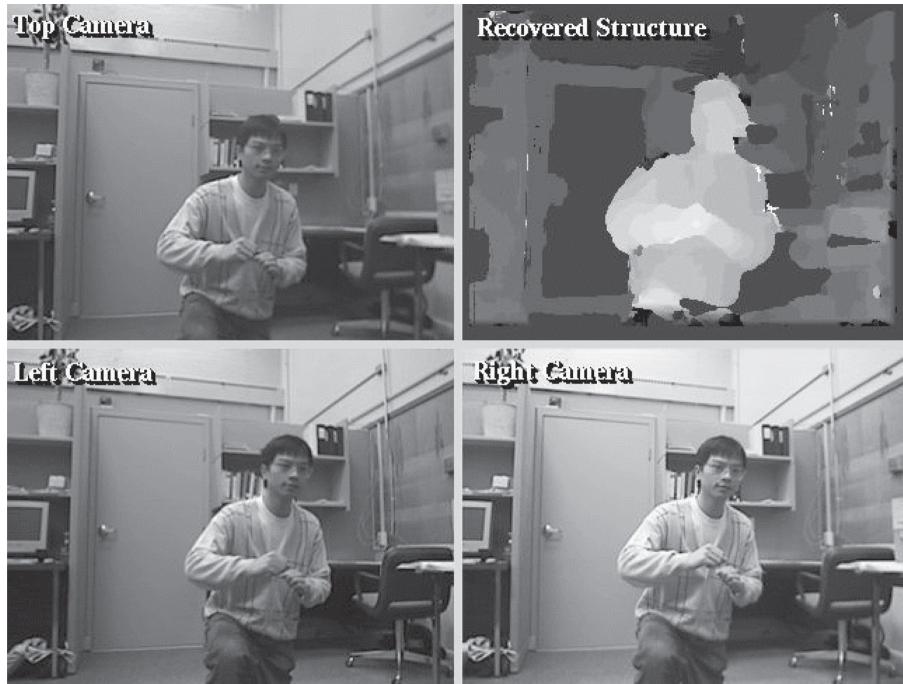


Figure 11.6 A set of stereo images from a Triclops stereo camera and resulting depth map. (Images courtesy of Chandra Kambhamettu.)

The major drawback with extracting range information from vision is that the algorithms tend to be computationally complex. Stereo matching algorithms execute typically on the order of $O(n^2m^2)$. This means that an image of size 640 by 480 takes on the order of 9×10^{10} instructions to process, while the number of instructions required to segment a color is on the order of $O(nm)$ or 3×10^5 . Even with advances in microprocessors, a stereo range map can take minutes to compute.

11.3 Depth from X

Computer vision has long exploited simple cues about the depth and orientation of a surface. These cues include the location of shadows or shading, the focal length at which an autofocus lens converges on an object, and changes in texture. The cues are often referred to as “depth from shading,” “depth from focus,” “depth from texture,” and so on, leading to the “depth from X” sobriquet.

Polly is an example of a robot that could calculate depth from texture (see [figure 11.7](#)). It served as an autonomous tour-guide at the MIT AI Laboratory and Brown University during the early 1990s. At that time vision processing was slow and expensive, which was totally at odds with the high update rates needed for navigation by a reactive mobile robot. The percept for the obstacle avoidance behavior was based on a clever affordance: texture. The halls of the AI Lab were covered throughout with the same carpeting. The “color” of the carpet in the image tended to change due to lighting, but the overall texture or “grain” did not. In this case, texture was measured as edges per unit area, as seen with the fine positioning discussed in chapter 7.

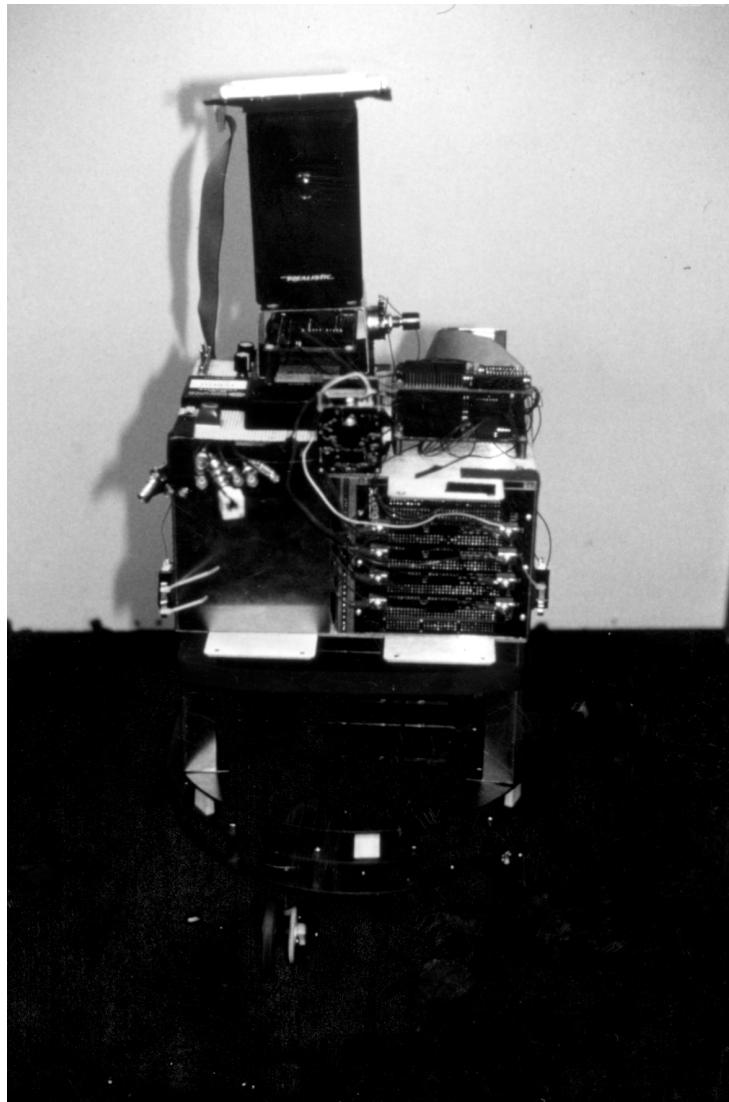


Figure 11.7 Polly a visually guided reactive robot using a black and white camera. (Photograph courtesy of Ian Horswill and AAAI.)

RADIAL DEPTH MAP

The robot divided the field of view into angles or sectors, creating a *radial depth map*, or the equivalent of a polar plot. Every sector that produced a texture measure close to the texture of the carpet was treated as empty. If a person was standing on the carpet, that patch would have a different texture, and the robot would mark the area as occupied. Although this methodology had some problems—for example, strong shadows on the floor created “occupied” areas—it was fast and elegant.

11.4 Sonar or Ultrasonics

Sonar refers to any system using sound to measuring range. Sonars for different applications operate at different frequencies; for example, a sonar for underwater vehicles would use a frequency appropriate for traveling through water, while a ground vehicle would use a frequency more suited for traveling through air. Sonars for ground vehicles use an ultrasonic frequency, just at the edge of human hearing. As a result the terms “sonar” and “ultrasonics” are used interchangeably when discussing extracting range from acoustic energy.

TIME OF FLIGHT

Ultrasonics are active sensors which emit a sound and measure the time it takes for the sound to bounce back. The *time of flight* (time from emission to bounce-back), along with the speed of sound in that environment (remember, even air changes density with altitude), is sufficient to compute the range of the object.

POLAR PLOT

Ultrasonics were extremely common during the 1980s and 1990s for several reasons. Their evolution paralleled the rise of the Reactive Paradigm. In the mid-1980s, Hans Moravec did impressive robot navigation with a ring of sonars. The ring configuration gave 360° coverage as a *polar plot*. This ring was developed by one of the first mobile robot manufacturers, Denning Robotics. Since then sonar rings are often referred to as “Denning rings,” regardless of manufacturer. Besides providing direct range measurements, the sonar transducers in the rings were cheap, fast, and had terrific coverage. In the early 1980s, the Polaroid Corporation developed small, inexpensive sonars for use as camera range finders. A bigger version, the Polaroid laboratory grade ultrasonic transducer, cost on the order of \$30 USD and could measure ranges from 1 to 25 feet with inch resolution over a field of view of 30°. Furthermore, the measurement time was on the order of seconds versus hours for computer vision. Ultrasonics became the sensor of choice for behavior-based robots.

A robotic sonar transducer is shown in [figure 11.8](#). The transducer is about the size and thickness of a dollar coin and consists of a thin metallic membrane. A very strong electrical pulse through the transducer generates a waveform, causing the membrane on the transducer to produce a sound. The sound is audible as a faint clicking noise, like a crab opening and closing its pincers. Meanwhile a timer is set, and the membrane becomes stationary. The reflected sound, or *echo*, vibrates the membrane. The vibration is amplified and then thresholded on return signal strength; if too little sound is received, then the sensor assumes the sound is noise and so ignores it. If the signal is strong enough to meet the threshold, the timer is tripped, yielding the time of flight.



Figure 11.8 Polaroid ultrasonic transducer. The membrane is the disk.

Determining whether the sonar data from a particular environment will be useful requires understanding how the sound wave is generated by the transducer. In reality, the sound beam produces multiple secondary sound waves around the transducer which interact over different regions of space before dissipating. Secondary sound waves are called *side lobes*. Most robot systems assume that sound from the main, or centermost, lobe is solely responsible for a range measurement. The width of the main lobe is often modeled as being 30° wide at about five meters away. However, in practice, reactive robots need to respond to obstacles in the 0.3- to 3-meter range. As a result, many algorithms treat the lobe as being between 8° and 15° wide, depending on how reliable the range readings are in a particular environment. Chapter 15 will go over reliability in range readings in more detail.

The strength of the main lobe in the environment determines the maximum range that the sonar can extract reliability. In ideal indoor venues, a sonar might return ranges up to 25 feet, while in the outdoors, the same sonar might go no further than eight feet with any repeatability. The upper limit of the range reading depends on the sensor and the environment; the lower limit does not. Ultrasonic transducers have a “dead time” immediately following emission while the membrane vibration decays. The decay time translates into an inability to sense objects within 11 inches because measurements made during this period are unreliable since the membrane may not have stopped ringing.

Regardless of the maximum allowed range return (i.e., does the program ignore any reading higher than three meters?) and the width of the lobe, most computer programs divide the area covered by a sonar into the three regions shown in [figure 11.9](#). Region I is associated with the range reading. It is an arc, because the object that returned the sound could be anywhere in the beam. The arc has a width because there are some resolution and measurement errors; the width of Region I is the tolerance. Region II is the area that is empty. If that area were not empty, the range reading would have been shorter. Region III is the area that is theoretically covered by the sonar beam, but it is unknown whether it is occupied or empty because it is in the shadow of whatever was in Region I. Region IV is outside of the beam and not of interest.

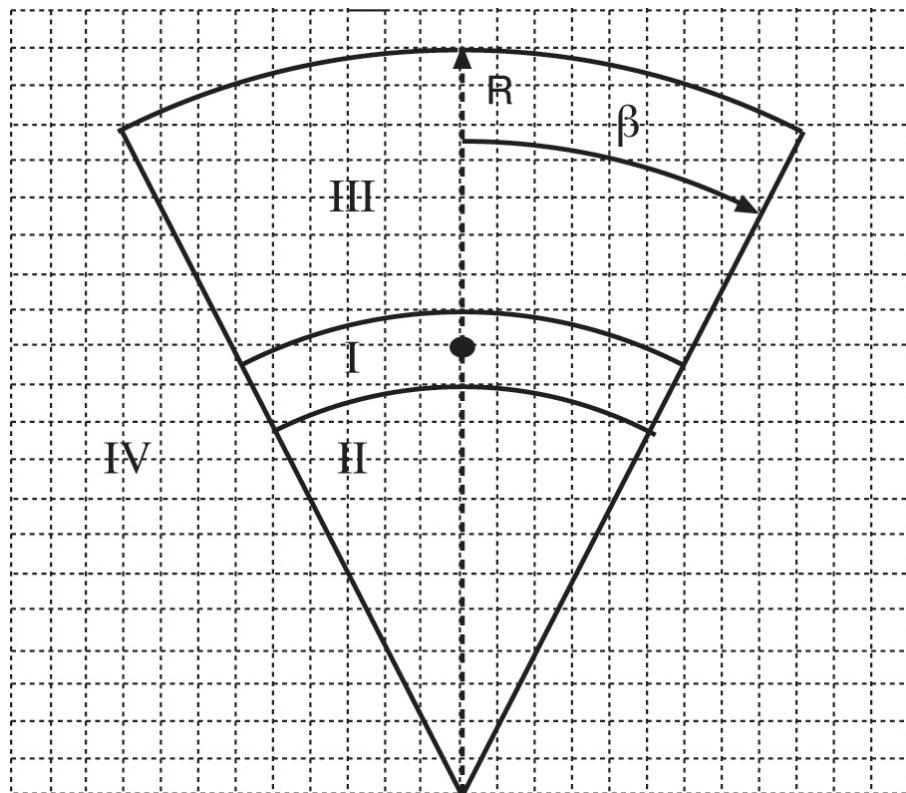


Figure 11.9 The regions of space observed by an ultrasonic sensor.

Although they are inexpensive, fast, and have a large operating range, ultrasonics have many shortcomings and limitations which a designer should be aware of. Ultrasonic sensors rely on reflection, and so they are susceptible to *specular reflection*. Specular reflection occurs when the wave form hits a surface at an acute angle and the wave bounces away from the transducer. Ideally, all objects should have a flat surface perpendicular to the transducer, but, of course, this rarely happens. To make matters worse, the reflected signal may bounce off of a second object, and so on, until, by coincidence, they return some energy to the transducer. In that case, the time of flight will not correspond to the true relative range.

FORESHORTENING

Even at severely acute angles, the surface is usually rough enough to send back some amount of sound energy. An exception is glass, which induces serious specular reflection, a common problem in hospitals and offices where robots operate. Fortunately, a portion of energy may not be scattered and is often sufficiently strong to pass the thresholding in the transducer circuit. However, a new problem, *foreshortening*, may occur. Recall that a sonar has a 30° field of view. This means that sound is being broadcast in a 30°-wide cone. If the reflective surface is not perpendicular to the transducer, one side of the cone will reach the object first and return a range first. Most software assumes the returned reading is along the axis of the sound wave. If the software uses the reading (which is really the reading for 15°), the robot will respond to erroneous data. There is no solution to this problem.

CROSS-TALK

Specular reflection is not only a significant source of erroneous readings, it can introduce a new type of error in rings of sonars or arrays. Consider a ring of multiple sonars giving 360° coverage around a robot. Suppose the sonars fire (emit a sound) at about the same time. Even though they are each covering a different region around the robot, some specularly reflected sound from a sonar might wind up getting received by a completely different sonar. The receiving sonar is unable to tell the difference between sound generated by itself and that generated by its peers. This erroneous reading is called *cross-talk* because the sound waves are getting crossed. Most robot systems stagger the firing of the sonars, for example in a Denning ring, with 16 sonars; four sonars fire at one time, one from each quadrant of the ring. This helps with cross-talk, but it is not a complete or reliable solution. If the sonar sound frequency and firing rate can be changed (which is generally not the case), then sophisticated aliasing techniques can be applied. These techniques are outside the scope of this book.

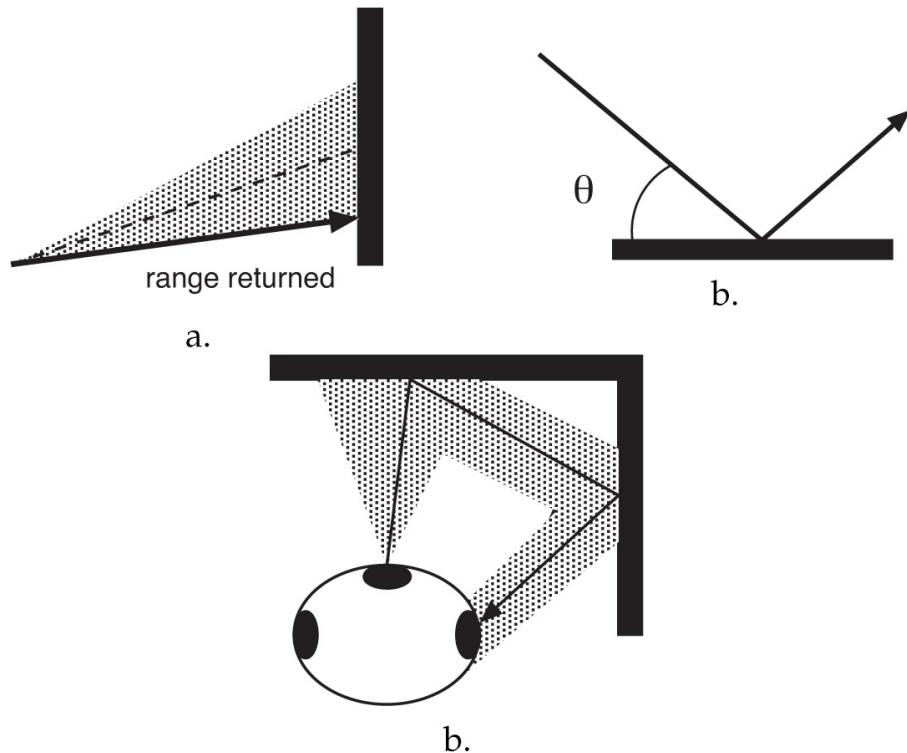


Figure 11.10 Three problems with sonar range readings: a.) foreshortening, b.) specular reflection, and c.) cross-talk.

One researcher, Monet Soldo, tells the story of developing a reactive mobile robot for IBM's T.J. Watson Laboratories during the late 1980s. The robot used sonar as its primary sensors, and Soldo had written behaviors to guide the robot through doors, rooms, and halls successfully at reasonable speeds. The day came for the big demonstration, which was to be held not in the hallways of the laboratory but in the front reception area. The robot navigated successfully out of the lab, down the halls, and then went berserk when it got to the atrium. Soldo rebooted, and tried again but got the same result. After days of trying to debug the code, she realized it was not a code problem, it was an environment problem: most of the atrium reception area consisted of glass partitions. The specular reflection and cross-talk caused the robot to hallucinate, although in different ways each time.

I had a similar “berserk robot” problem when my sonar-based robot started navigating in an office environment. In that environment, the robot was expected to navigate among office cubicles delimited by partitions. The partitions were covered with cloth to dampen the sound from the workers. Unfortunately, the cloth also absorbed the sound from the sonars! These stories emphasize the need to consider the operating environment for the sensor and its impact on the signal.

The impact of the problems of specular reflection and cross talk become easier to see with plots of sonar returns overlaid on the perimeter of the area they were taken in, see [figure 11.11](#). Some walls are invisible; others are too close. The sensor readings are highly uncertain, and, as a result, are now handled using probabilistic methods that will be covered in chapter 14.

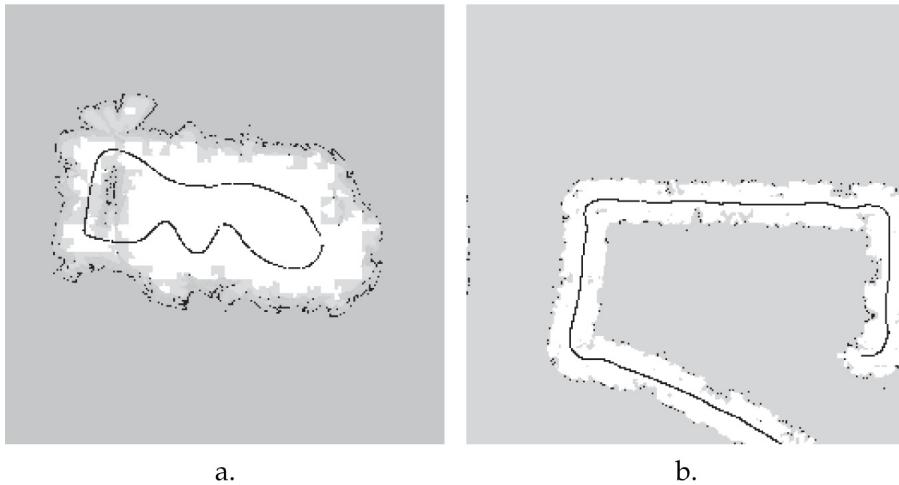


Figure 11.11 Maps produced by a mobile robot using sonars in: a.) a lab and b.) a hallway. (The black line is the path of the robot.)

The 30° cone also creates resolution problems. While sonars often have excellent resolution in depth, they can achieve that only at large distances if the object is big enough to send back a significant portion of the sound wave. The further away the object is from the robot, the larger the object has to be. Most desk chairs and table tops present almost no surface area to the sensor, and so the robot often will not perceive their presence and run into them.

In practice, another problem leads to spurious sonar readings: power. The generation of a sound wave requires a significant pulse of energy. If the robot is operating at low power levels, the correct waveform will not be generated, and the return signal will be worthless.

One method of eliminating spurious readings, regardless of cause, is to take the average of three readings (the current reading plus the last two) from each sensor. This method is fairly common on purely reactive robots, but it is an ad hoc process. As will be seen in later chapters, other approaches treat the reading as uncertain and apply formal evidential reasoning techniques to refine the reading. These uncertainty techniques are employed by architectures operating under the Hybrid Paradigm and will be covered in chapter 15.

11.4.1 Light Stripers

Light striping, light stripers, or structured light detectors work by projecting a colored line (or stripe), grid, or pattern of dots on the environment. Then a regular vision camera observes how the pattern is distorted in the image. Current RGB-D cameras are based on light stripers, but instead of visible lines, the camera systems project random dots of infrared light that are not visible to the human eye, see section 11.4.3.

NEGATIVE OBSTACLE

A simplified example of how a light stiper works is given in figure 11.12. In figure 11.12a, the stiper projects four lines. The lines should occur at specific, evenly spaced rows in the camera image if the surface is flat. If the surface is not flat, as shown in figure 11.12b, the lines will have breaks or

discontinuities. A vision algorithm can quickly scan each of the designated rows to see if the projected line is continuous or not. The location of the breaks in the line give information about the size of the obstacle. The vision algorithm can also look at the location where the dislocated line segments appear, since the distance in image coordinates is proportional to the depth of the object. The relative placement of the lines indicates whether the object is above the ground plane (an obstacle) or below (a hole or *negative obstacle*). The more lines or finer-grained grid, the more depth information.

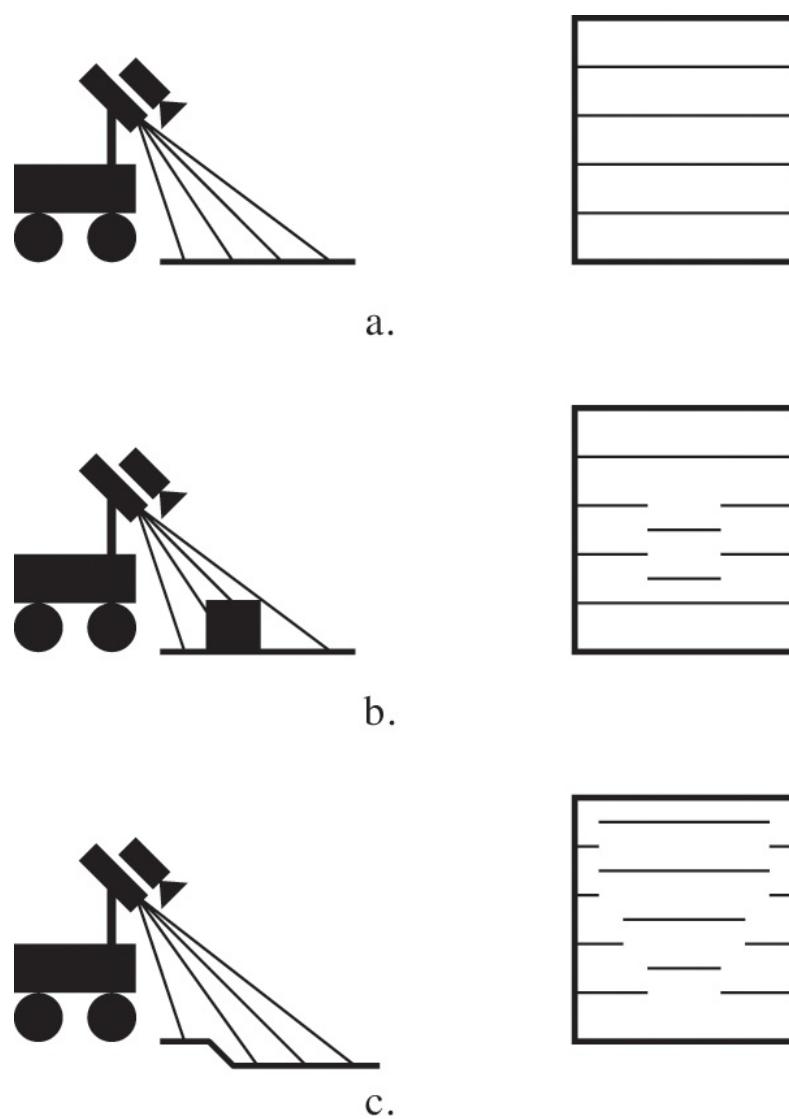


Figure 11.12 Situations and resulting images of a.) flat surface, b.) an obstacle, and c.) a negative obstacle.

Light stripers are less expensive than laser range finders for many reasons. First, since they are producing a line or pattern to be measured, expensive time-of-flight detectors are unnecessary. The

detection is done by the camera. Second, producing a thick line that can be detected by a camera does not require a laser. Instead it can be done with structured light, an optical method which allows “regular” light to mimic desirable properties of laser light. Finally, light stripers produce a fairly coarse pattern; they do not project a line or grid onto every pixel in the image. This means the light stiper is less demanding, and therefore less expensive to build.

Light stripers were popular in the late 1980s and early 1990s, and laboratories, such as the University of Pennsylvania’s General Robotics, Automation, Sensing and Perception Laboratory (GRASP Lab) under the direction of Ruzena Bajcsy, produced excellent results in extracting 3D information. However, these efforts focused on using the depth map to recognize an object under laboratory conditions. The results did not transfer particularly well to mobile robots in less favorable conditions. Reactive robots are not concerned with recognition, so many of the algorithms were not particularly useful or able to provide a quick, reflexive response. Also, in the open world, objects were often the same color as the projected light or near enough to it to confuse the vision system. The amount and direction of lighting could also confuse the stiper, with brightly lit rooms making it difficult for the light stiper to see the bright laser or structured light.

The Sojourner Mars rover used a light striping system for obstacle avoidance. The system projected five lines ahead of the vehicle.¹⁹⁸ This worked well because Mars has little color or light diversity. Interestingly enough, the light stiper used one member of a stereo pair, but unlike the Marsokhod, Sojourner did not use stereo for navigation. Instead, the robot periodically took a stereo pair for reconstruction and map-making on Earth.

11.4.2 Lidar

LIDAR

Ultrasonics introduce acoustic energy into the environment and measure the time of flight of the signal to return. The same principle can be used with lasers; a laser beam is emitted and the reflectance measured. Unlike a sonar beam which has a very wide field of view (almost 30°), a laser produces an almost infinitesimal field of view. If the laser is directed in a scan, just like a raster scan on a CRT, the device can cover a reasonable area and produce a image, where the image function produces depth values. Devices which use lasers to produce a depth map or image are often called *laser radar*, *ladar*, or *lidar*. They can generate as many as one million range pixels per second,⁷⁴ with a range of 30 meters and an accuracy of a few millimeters. The mechanical scanning component makes lidars very expensive, on the order of \$30,000 to \$100,000 USD. A less expensive solution for navigation is to create a planar laser range finder.

RANGE SEGMENTATION

A lidar produces two images: *intensity* and *range*. Figure 11.13 shows the images produced by an Odetics laser range (LADAR) camera. The intensity map is essentially a black and white photograph that measures the intensity of the light reflected or absorbed by objects in the scene. This corresponds to how humans perceive the scene. The image function for the range image represents depth from the camera. Pixels that are black, or have a value of 0, are nearer to the range camera than white pixels. A flat floor usually appears as a radiating set of semi-circles going from black (near) to white (far). Trigonometry is then used to compute that the circles represent a flat surface. This process is called

range segmentation and it can be quite difficult to reconstruct surfaces from the data.

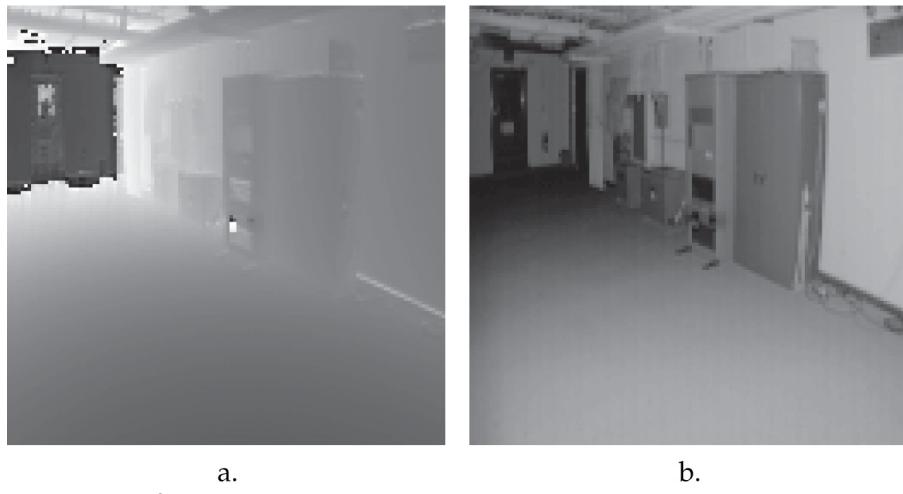


Figure 11.13 Lidar images from an Odetics LADAR camera: a.) range and b.) intensity. (Images courtesy of University of South Florida and Oak Ridge National Laboratory.)

Lidars have some problems in practice. Figure 11.13 shows an area on the range image that is pure black or very near. But as can be seen from the intensity image the area is actually far away. Likewise, the black moulding between the wall and floor appear to be very far away on the range image. The errors were due to out of range conditions, absorption of the light (not enough light returned), or to the optical equivalent of specular reflection (light hitting corners gets reflected away from the receiver).

A planar laser range finder, such as the SICK® shown in figure 11.14, emits a plane of laser light. This provides a thin horizontal slice of space and produces essentially a high resolution polar plot. Robots initially came with a SICK laser mounted in parallel to the floor. This was useful for obstacle avoidance (as long as the obstacle was tall enough to break the plane projected by the laser), but it was not particularly helpful for extracting 3D information. Also, as with sonars, robots ran the risk of being decapitated by obstacles, such as tables, which did not appear in the field of view of the range sensor, but which could hit a sensor pod or antenna. To combat this problem, researchers have recently begun mounting planar laser range finders at a slight upward angle. As the robot moves forward, it gets a different view of upcoming obstacles. In some cases, researchers have mounted two laser rangers, one tilted slightly up and the other slightly down, to provide coverage of overhanging obstacles and negative obstacles.



Figure 11.14 SICK laser, covering a 180° area.

11.4.3 RGB-D Cameras

RGB-D CAMERA

RGB-D cameras are a combination of a IR light striping system (except instead of stripes of light, random dots are projected) with a regular RGB visible light camera. These are sometimes called RGB+D or just RGBD cameras. RGB-D cameras project IR light read by a camera tuned for the IR frequency of light. The IR dots are invisible to the human eye. Thus a separate RGB camera can be mounted next to the IR camera to perceive the same scene without dots and make it visible to the human eye. The low-cost Microsoft Kinect and ASUS Xtion cameras use essentially the same technology for videogames and were immediately adopted for use by roboticists.

However, as noted in Suarez and Murphy,²⁰¹ the RGB-D cameras are not sufficient for general robotics for three reasons. First, RGB-D cameras are typically optimized for the videogame market, where the camera is stationary, indoors, and the gamer adjusts the lighting in the room and turns the camera system until it produces consistent results. In mobile robots, the robot may move through regions that have much different lighting conditions than does an indoor recreational room. Bright light and direct sunlight wash out the IR projection. Second, the IR light is an active sensor. Thus objects with reflective surfaces or sharp edges and angular surfaces that create specular reflection cause the IR image to be noisy. Third, RGB-D cameras are tuned for projection and receptivity over specific ranges. In the case of a Kinect, that range is 1.2 m to 3.5 m. If a robot needs to perceive depth closer than 1.2 m or farther than 3.5 m, the Kinect will not be of use.

11.4.4 Point Clouds

POINT CLOUD

Point clouds are a knowledge representation of the range to surfaces. The cloud is a collection of points, usually thousands of points. The cloud can be an array or a list of points. Each point has either a (x, y, z) coordinate, called a *vertex*, representing the distance between the sensor and the surface of an object or a vector (*direction, distance*). The point itself actually represents a volume of space, just like a sonar return, but the resolution is generally several orders of magnitude higher than what is produced by an ultrasonic sensor. A point may also have a value, such as the saturation of the light, which can produce a image similar to a black and white photograph. While point cloud technically refers to the collection of points, the term invokes the associated algorithms and styles of visualization for working with the points.

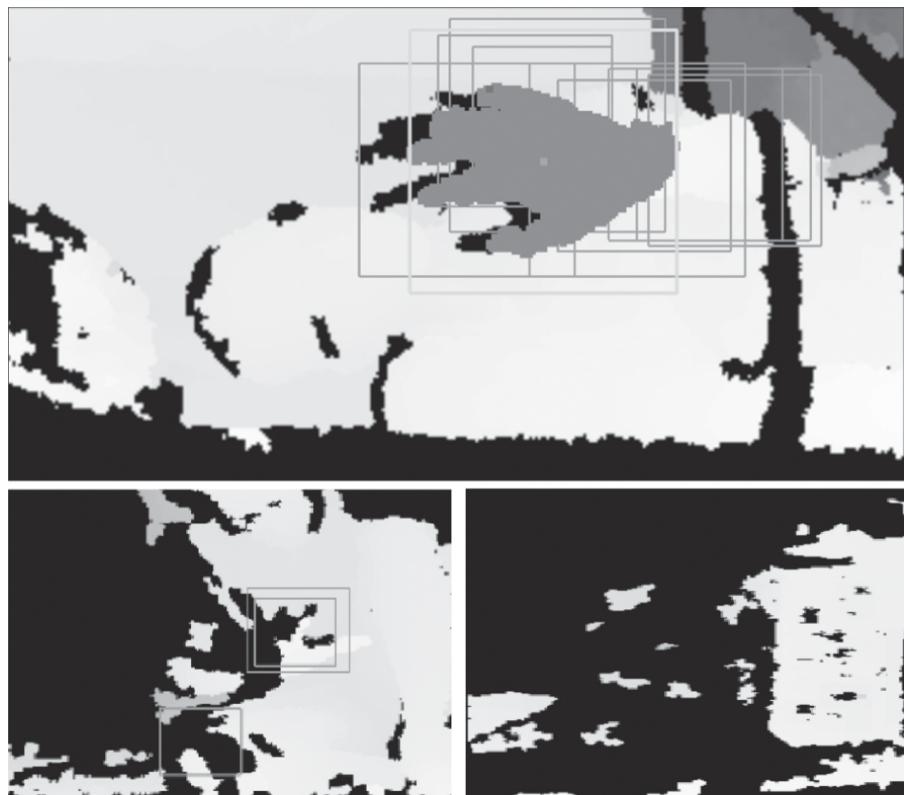


Figure 11.15 Three views of a person's hand taken with a Kinect in different lighting. Above is indoor lighting, below left is partial shade outdoors, and below right is full sun. The green boxes show where an extraction algorithm hypothesizes the presence of a hand. (Images courtesy of Carlos Soto.)

Point clouds are usually generated by active sensors, either *lidar* range sensors or *structured light* sensors. Active range sensors, such as models made by SICK, Velodyne®, and Hokuyo® emit a laser beam and then record the time it takes the beam to bounce off a surface and return. The sensors scan either in a two-dimensional plane, like a sheet of paper, or scan in three-dimensions. Scanning in three dimensions is more expensive as it is much more complicated optically and electromechanically. SICK sensors were originally developed for precision surveying, where 2D is sufficient. Other 2D scanners,

sometimes called *planar laser rangers*, such as the Hokuyo, are restricted to 2D in order to keep costs down. Structured light sensors usually emit light in a near infrared frequency band that is invisible to the human eye. The best-known structured light range sensor is the Microsoft Kinect. The points may be generated systematically, such as a raster scan which produces an array of thousands or millions of points, or the sensor may sense the region randomly to reduce the number of points on each scan.

Originally the term point cloud implied a much higher density of points than those generated from stereo vision—so high that a different class of algorithms was needed to efficiently work with the large number of points. Some algorithms do not even try to use all the points but instead randomly sample the points in the cloud in order to reduce the amount of computation time needed. The knowledge representation of a point cloud and associated algorithms for efficiently processing the points is now used generically. The pixels in a depth image from geospatial stereo satellite imagery or photogrammetric methods are now considered point clouds rather than arrays.

SENSE

The algorithms for working with a point cloud are often applied in a sequential process shown in figure 11.16. The computational effort increases as the process goes from acquiring points to trying to group the points into the correct surfaces that reconstruct the environment. The first step in the process is for the sensor to *sense* the environment and generate the set of points.

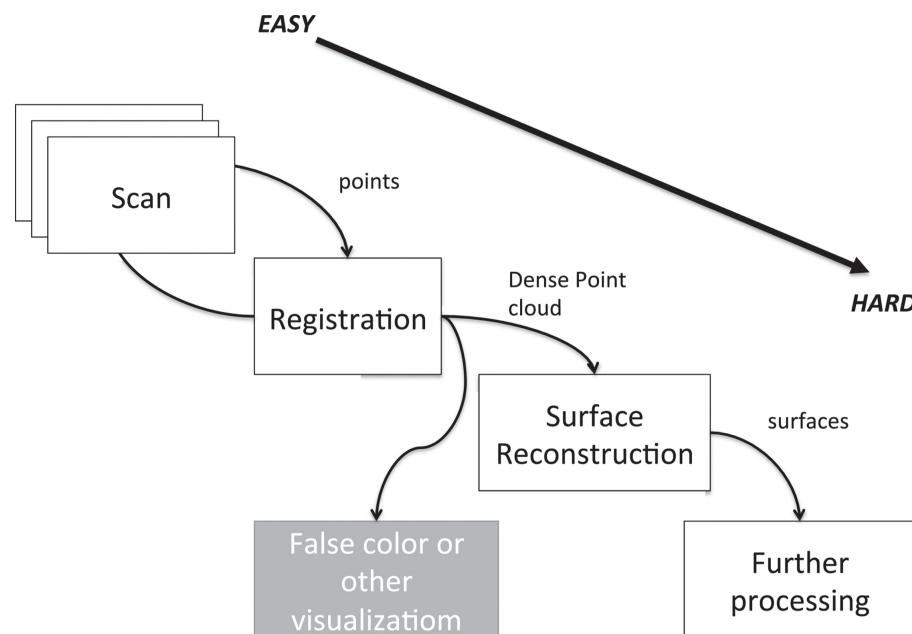


Figure 11.16 The sequence of processing activities associated with point clouds.

REGISTRATION

Second, this set of points needs to be *registered* with other sets of points. For example, a room may require multiple scans and those views have to be stitched together to form a single unified view. Another example is incorporating scans over time, such as when a robot moves toward an object.

Usually a major component of registration is assigning each point an x,y,z location in absolute coordinates, because the scan will be reported in relative, or egocentric, coordinates which have to be transformed into a common set of absolute coordinates. Registration can also overlay the visual image on the point cloud.

Point clouds by themselves are often so dense that just displaying the points at different distances from the robot with false color is a reasonable visualization of the depth. For a single object or a sparse scene, the points on a surface are often at the same distance from the viewpoint and thus the same color. However, with more complex scenes, the false color visualization requires the human to mentally segment the world into surfaces.

RECONSTRUCT

The robot duplicates the mental effort in converting points into surfaces through *surface reconstruction*. Essentially the points in the cloud are vertices of patches of surfaces. One approach to creating the surfaces is to apply *mesh algorithms*, such as Delaunay triangulation or marching cubes, which try to convert the vertices into a mesh of polygons or triangles that represent the surfaces in the point cloud. Another approach is to apply nonuniform rational Basis spline (NURBS) algorithms which try to link the vertices using curves.

Once the point cloud for a region has been converted into surfaces, further processing may be applied. The surfaces may become part of a 3D reconstruction of the environment and used for object recognition or path planning through the environment. Other processing includes extracting the pose of a hand or an arm for controlling a game (or robot).

11.5 Case Study: Hors d’Oeuvres, Anyone?

The University of South Florida’s (USF) entry in the 1999 AAAI Mobile Robot Competition Hors d’Oeuvres, Anyone? event provides a case study of selecting sensors, constructing reactive behaviors, and using behavioral sensor fusion. The entry used two cooperative robots. The goal was to push the envelope in robotic sensing by using six sensing modalities with 40 different physical devices on one robot and four modalities with 23 devices on the other.

Step 1: Describe the task. The “Hors d’Oeuvres, Anyone?” event required fully autonomous robots to circulate in the reception area at the AAAI conference with a tray of finger food, find and approach people, interact with them, and refill the serving tray. Each robot was scored on covering the area, noticing when the tray needed refilling, interacting with people naturally, having a distinct personality, and recognizing VIPs. The USF entry used two robots, Borg Shark and Puffer Fish, shown in [figure 11.17](#), costumed by the USF Art Department in order to attract attention and suggest different personalities. Borg Shark was the server robot which navigated through the audience following a preplanned route. The robot would stop and serve at regular intervals or whenever a treat was removed from the tray. It used a voice synthesizer to broadcast audio files inviting audience members to remove a treat from its mouth, but it had no way of hearing and understanding natural language human commands. In order to interact more naturally with people, Borg Shark attempted to maintain eye contact with people. If it saw a person, it estimated the location in image coordinates of where a VIP’s colored badge was in relation to a person’s face.

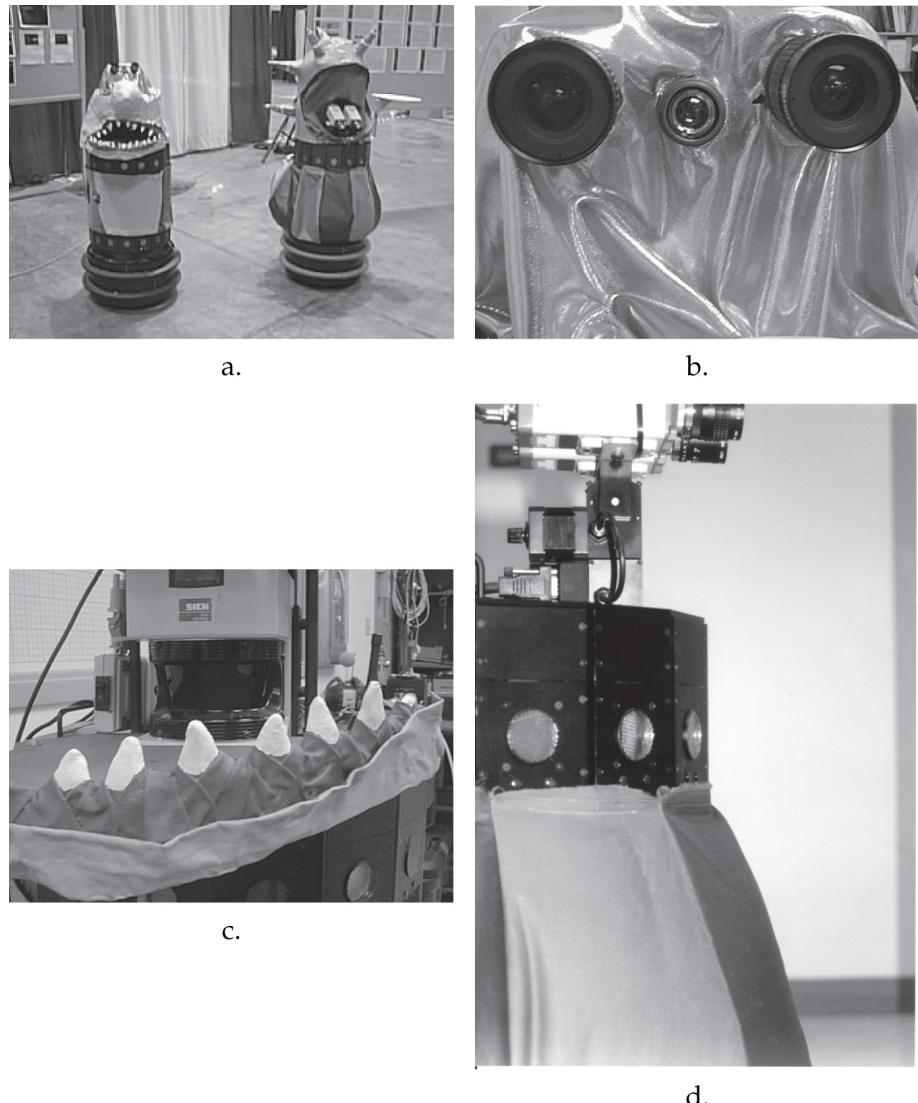


Figure 11.17 USF robots in the “Hors d’Oeuvres, Anyone?” event. a.) Family portrait, where Borg Shark is on the left, Puffer Fish on the right, with b.) the thermal sensor located as the Borg Shark’s “third eye,” c.) the SICK laser located behind Borg Shark’s teeth (head piece is removed for better view), and d.) a profile of Puffer Fish’s skirt showing spatial relationship to sonar.

When Borg Shark was almost out of food, she would call, over radio ethernet, her assistant robot, Puffer Fish. Puffer Fish would be stationary in sleep mode, inhaling and exhaling through her inflatable skirt and turning her cameras as if avoiding people crowding her. When Puffer Fish awoke, she would head with a full tray of food (placed on her stand by a human) to the coordinates given to her by Borg Shark. She would also look for Borg Shark’s distinctive blue costume, using both dead reckoning and visual search to move to goal. Once within two meters of Borg Shark, Puffer Fish would stop. A human would swap trays physically, then kick the bumpers to signal that the transfer was over. Borg Shark

would resume its serving cycle, while Puffer Fish would return to its home refill station.

Both robots were expected to avoid all obstacles: tables, chairs, people. Since there was a tendency for people to surround the robots, preventing coverage of the area or refilling, the robots had different responses. Borg Shark, who was programmed to be smarmy, would announce that it was coming through and begin moving. Puffer Fish, with a grumpy, sullen personality, would vocally complain while loudly inflating her skirt and then make a rapid jerk forward, usually causing spectators to back up and give her room.

Step 2: Describe the robots. The robots used for the entry were Nomad 200 bases with customized sensors. The original sensors on Borg Shark included a pair of color cameras mounted on a pan-tilt head, redundant sonar rings, and a SICK planar laser, see [table 11.1](#). The sensors on Puffer Fish are shown in [table 11.2](#).

Table 11.1 Sensors for Borg Shark.

Type	Modality	Devices
exteroceptive	vision	2 cameras
exteroceptive	laser	1 planar ranger
exteroceptive	sonar	15 ultrasonics (upper) 15 ultrasonics (lower)
exteroceptive	tactile	1 bumper switch
proprioceptive	motor encoders	3 drive, steer, turret control 2 pan, tilt control

Table 11.2 Sensors for Puffer Fish.

Type	Modality	Devices
exteroceptive	vision	2 cameras
exteroceptive	sonar	15 ultrasonics
exteroceptive	tactile	1 bumper switch
proprioceptive	motor encoders	3 drive, steer, turret control 2 pan, tilt control

Step 3: Describe the environment. The environment was a convention center arena with variable lighting and many people casually dressed. The major perceptual challenge for the robots was to maintain eye contact with people and to determine if a person was a VIP. The VIPs badges were marked with distinctively colored ribbons, so if the robot was sure it was looking in the right area, the color would *afford* a VIP. However, the colors of the ribbons were fairly common. If the robot scanned for ribbons at about chest height, it would likely find a shirt of that color in the crowd who would be wearing bright tourist wear (the competition was in Orlando). This would cause it to misidentify a VIP, losing points. Another approach was to use sonars to find a person by reading a range, pointing the camera in that direction and tilting to the angle where the eyes of a person of average height would be

located. This focus-of-attention mechanism could also be used to look in the likely location of a badge, if so desired. However, it was known from experience that people rarely stand along the acoustic axis of a sonar. If a single person was picked up by two sonars, the robot would look for 10 seconds to the left of the person and then 10 seconds to the right. If multiple people were present, the robot would seem even more dysfunctional, and the focus-of-attention mechanism would not work correctly.

A better solution would be for the robots to detect a person using vision. Notice that detection is not the same as recognition. Detection means that the robot is able to identify a face, which is reactive. Recognition means labeling the face and being able to recognize it at a later time, which is a deliberative function. Is there a simple visual affordance of a face? Actually, human skin, regardless of ethnicity, is remarkably similar in color to a vision system. Once the robot has found a colored region about the size and shape of a head, it could then more reliably find the VIP badges.

The other opportunity for an affordance was Puffer Fish's navigation to Borg Shark. Although Puffer Fish would receive Borg Shark's coordinates, it was unlikely that Puffer Fish could reliably navigate to Borg Shark using only dead reckoning. The coordinates were likely to be incorrect because of Borg Shark's own drift over time. Then Puffer Fish would accumulate dead reckoning error, more so if it had to stop and start and avoid people. Therefore, it was decided that Puffer Fish should look for Borg Shark. Borg Shark's head piece was deliberately made large and a distinctive blue color to afford visibility over the crowd and reduce the likelihood of Puffer Fish fixating on someone's shirt.

Step 4-7: Design, test, and refine behaviors. The choice of sensors for other behaviors, such as treat removal, was influenced by the physical location of the sensors. For example, the SICK laser for Borg Shark came mounted on the research platform as shown in [figure 11.17b](#). The research platform, nominally the top of the robot, was at hand height, making it a logical place to attach a tray for holding food. It was obvious that the laser could be used to monitor the food tray area. Other teams tried various approaches such as having a colored tray and counting the area of that color that was visible (more color means fewer treats on the tray). Another approach was to build a scale and monitor the change in weight of the tray.

The robot costumes impacted the sensor suite indirectly. As part of giving the robots personality, each robot had a costume. The Puffer Fish had an inflatable skirt that puffed out when the robot was crowded or annoyed. The team had to empirically test and modify the skirt to make sure it would not interfere with the sonar readings. [Figure 11.17c](#) shows the profile of the skirt.

The initial behaviors for Borg Shark are given in [table 11.3](#) and Puffer Fish in [table 11.4](#). As seen in the tables, the only behavior using any form of sensor fusion was move - to - goal in Puffer Fish, which had two competing instances of the goal making it sensor fission.

Table 11.3 Behaviors for Borg Shark.

Releaser	Behavior	Motor Schema	Percept	Perceptual Schema
always on	avoid()	vfh()	most-open-direction	polar-plot(sonar)
FOOD-REMOVED=treat-removal(laser)	track-face	center-face(face-centroid) track-face() check-VIP()	face-centroid ribbon-color	find-face(vision) look-for-ribbon(VIP-color)
SERVING-TIME-OUT, TRAY-FULL=bumper()	move-to-goal	pfields.attraction(waypoint)	waypoint	list of waypoints
FOOD-DEPLETED=treat-removal(laser)	track-face	center-face(face-centroid)	face-centroid	find-face(vision)

Table 11.4 Behaviors for Puffer Fish.

Releaser	Behavior	Motor Schema	Percept	Perceptual Schema
always on	avoid()	vfh()	most-open-direction	polar-plot(sonar)
AT-HOME=dead-reckoning(encoders)	sleep()	turn-camera-head() cycle-skirt()	obstacle	polar-plot(sonar)
AWAKE=radio-signal()	move-to-goal()	pfields.attraction(location)	relative-location	read-encoders()
AWAKE=radio-signal()	move-to-goal()	pfields.attraction(shark)	shark	find-shark-blue(camera)
TRAY-FULL=bumper()	move-to-goal()	pfields.attraction(home)	relative-location	read-encoders()

The `vfh` behavior is an obstacle avoidance behavior using polar plots derived from models described in chapter 15. As the team tested the behaviors individually, problems were identified with the `find-face` and `treat-removal` behaviors. While color was a reasonable affordance for a face, the algorithm often returned false negatives, missing faces unless they were in bright light. Meanwhile the laser occasionally appeared to get a reflection from the costume's teeth, also generating false positives, and more than 75% of the time the laser would miss a person's hand if the hand motion was quick. The rates are shown below:

Logical Sensor	False Positives	False Negatives
Face-Find	1.7%	27.5%
Food-Count	6.7%	76.7%

The solution to the `find-face` performance was to exploit another affordance of a human, one that is used by mosquitoes: heat. The problem was partial segmentation of a human because the candidate regions were getting rejected for being too small. Heat would make a good decision criterion. If a candidate region was colocated with a hot region, then it was declared a face. Fortunately, the team was able to transfer a *E²T* digital thermometer used on another robot to Borg Shark. The thermal sensor shown in [figure 11.17](#) was intended for determining the temperature of a person on contact, but it was able to detect an increase in temperature above ambient temperature from a person up to two meters away.

Why was the vision system not simply replaced with the thermal sensor? This situation harkens back to the attributes of sensors and how they fit with the environment. The thermal sensor has a 2° field of view and a measurement latency, making it too slow for quickly scanning a room for humans. Instead, the vision system covered a much wider field of view and could generate a small list of candidate regions. Then, as the camera turned to center on the largest region, the thermal probe readings could determine whether this was really a person or not.

The problem with the food-count estimate was greatly reduced by a simple AND function with the sonars. The system counted a treat as being removed only if there was a close range reading in front of the tray at the same time as the laser detected a hand motion.

The false reading rate dropped considerably as seen below:

Logical sensor	W/O Fusion		Fusion	
	FP	FN	FP	FN
Face-Find	1.7%	27.5%	2.5%	0%
Food-Count	6.7%	76.7%	6.7%	1.7%

At this point it is helpful to step back and examine the sensing for the *Hors d’Oeuvres, Anyone?* entry in terms of the attributes listed in chapter 10. Recall that the attributes for evaluating the suitability of an individual sensor were *field of view and range, accuracy, repeatability, and resolution, responsiveness in target domain, power consumption, reliability, and size*. The field of view and range of the sensors were issues, as seen by the differences in the face-finding behavior with vision and with thermal sensors. The camera had a much better field of view than the thermal sensor, so the camera was used to focus the attention of the heat sensor. Repeatability was clearly a problem for the laser with its high false positive/false negative rate. The sonars could not be used for estimating the location of a face because the resolution was too coarse. Each of the sensors had reasonable responsiveness from a hardware perspective, though the algorithms may not have been able to take advantage of them. Power consumption was not an issue because all sensors were on all the time due to the way the robots were built. Reliability and size of the hardware were not serious considerations since the hardware was already on the robots.

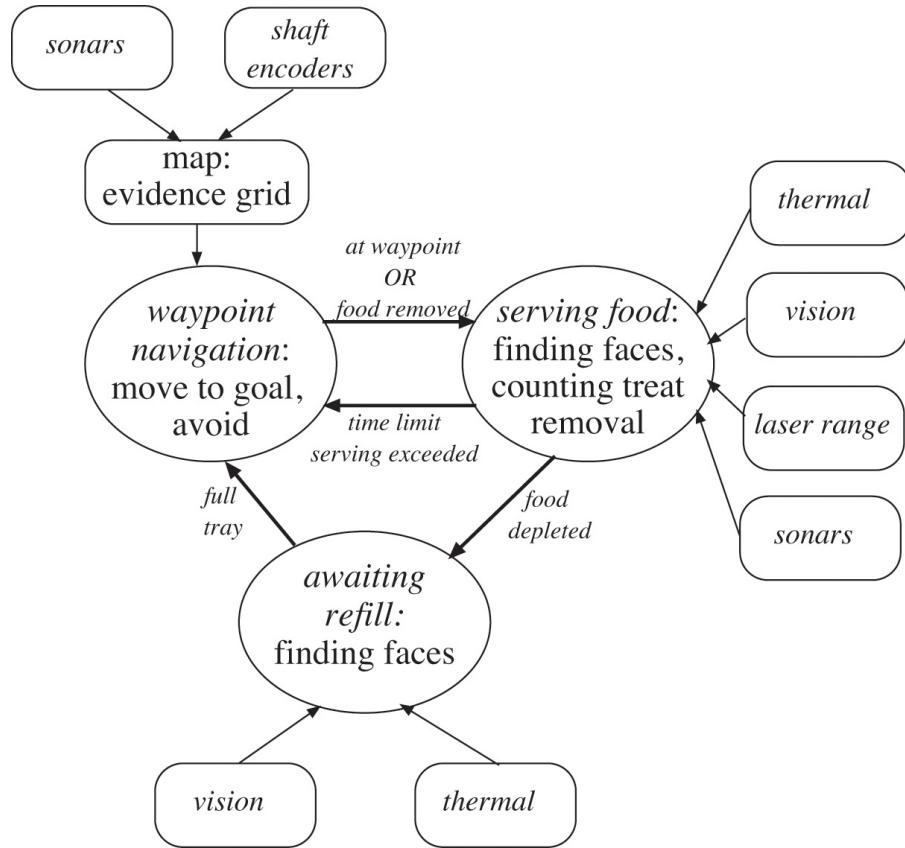


Figure 11.18 State diagram for the Borg Shark, annotated to show sensors being used.

The algorithmic influences on the sensor design were *computational complexity* and *reliability*. Both were definitely a factor in the design of the perceptual schemas for the reactive behaviors. The robots had the hardware to support stereo range (two cameras with dedicated framegrabbers). Stereo ranging could have been used to find faces, but the high computational complexity, even with a Pentium-class processor of the time, could not process the algorithm in real-time. Reliability was also an issue. The vision face-finding algorithm was very unreliable, not because of the camera but because the algorithm was not well-suited for the environment and, as a result, it picked out extraneous blobs.

Finally, the sensing suite can be rated overall in terms of *simplicity*, *modularity*, and *redundancy*. The sensor suites for both robots can be considered simple and modular in that they housed several separate sensors, mostly commercially available, that were able to operate independently of each other. The sensor suite did exhibit a high degree of physical redundancy: one robot had dual sonar rings, the sonars, laser, and a camera pair could have been used for range-finding, ignoring the placement of the shark teeth and the computational complexity of stereo range algorithms. There was also a large amount of logical redundancy, which was exploited through the use of behavioral sensor fusion.

11.6 Summary

This chapter has explored *What is the difference between depth and range?* and *Why is it so hard for robots to sense depth and range?* Navigation and manipulation require exteroception, whereby the robot observes the environment. In particular, the robot generally needs depth perception over large distances for navigation; this is generally referred to as range sensing. Laser rangers, or Lidar, are preferred but have drawbacks including cost, size, and weight. These produce a high density of $\{x, y, z, d\}$ points leading to point cloud representation and associated processing algorithms for registration and reconstruction. RGB-D cameras, such as those pioneered by the Microsoft Kinect, are low cost, low weight, and small but rely on an IR sensor, which has been carefully engineered for indoor game rooms, and rely on specific algorithms built for people standing at a specific distance from the sensor. RGB-D cameras are not a general purpose sensor and have significant problems with outdoor lighting and specular and reflective surfaces. Ultrasonics and computer vision have been explored, with ultrasonics being largely discarded due to low resolution and unreliability. However, the occupancy grid representation and the use of evidential reasoning to handle uncertainty persists in sensing. Computer vision offers many approaches to extracting depth and detecting surfaces, including using stereo pairs and calculating depth from X. Depth is generally over close distances for manipulation. Range and depth sensors and algorithms have drawbacks, such as computational cost, financial expense, inability to work under different lighting conditions, and the lack of algorithms to robustly convert the raw sensor data into accurate 3D representations.

11.7 Exercises

Exercise 11.1

Define point cloud and give at least one example of a sensor and a software algorithm that produces a point cloud.

Exercise 11.2

Compare and contrast how lidar and RGB-D cameras work. Discuss what is similar and what is different, advantages and disadvantages, and computational complexity.

Exercise 11.3

Describe the purpose of each step in the Sense—Register—Reconstruct sequence for processing point clouds.

Exercise 11.4

Ultrasonic sensors have many positive and negative attributes. Name and describe three positive and three negative attributes.

Exercise 11.5

Describe the three major problems of ultrasonic sensing, and define a hypothetical instance in which a robot would encounter each problem (such as a room with windows or glass partitions).

Exercise 11.6

Describe the problems of specular reflection, cross talk, and foreshortening with an ultrasonic transducer.

Exercise 11.7

An alternative to a Denning ring is to mount one or more sonars on a mast that rotates. Turning gives the robot a 360° coverage. Which do you think would be better in general for range sensing, a fixed ring or a panning mast? Would a panning mast reduce problems of foreshortening, cross-talk, and specular reflection?

Exercise 11.8

Define *image function*. What is the image function for

- a. the left-right images in a stereo pair?
- b. the depth map?

Exercise 11.9

Describe the similarities and differences between an image array, occupancy grid, and point cloud.

Exercise 11.10

Consider an obstacle avoidance behavior which consists of a perceptual schema that provides a polar plot of range and motor schema which direct the robot to the most open sector. List all the sensors and algorithms covered in this chapter that can be used interchangeably for the perceptual schema. Which of these are logically redundant? Physically redundant?

Exercise 11.11

Assume you had a mobile robot about 0.5 meters high and a planar laser range finder. At what angle would you mount the laser if the robot was intended to navigate

- a. in a classroom?
- b. in a hallway or reception area where the primary obstacles are people?
- c. outdoors in unknown terrain?

State any assumptions your design is based on. Is there more information needed; if so, what?

11.8 End Notes

Yes, it IS a SICK sensor.

The adoption of the planar laser made by the German surveying company, SICK, resulted in a spate of confusion (SICK is the brand name, not the status) and interesting pronunciations (SEE-ick, SEE-CHEECH, etc.).

Decapitating robots.

One challenge with planar lidar is where to mount it on a robot. A common strategy is to try to mount the sensor as high above the robot as possible and facing slightly down so that the robot can perceive obstacles and terrain. But since the sensor is not pointing forward, it has a blind spot, and, as a result, several robots and autonomous driving cars have decapitated their expensive lidar.

Lidar is not just for robots, it is for music videos as well.

In 2008, Radiohead released a video for its song *House of Cards*; the video was shot with Velodyne lasers, producing a point cloud video of the band.

Undergraduates and sonars.

It is interesting to note that the first serious analysis of the Polaroid sonars was done by an undergraduate at MIT, Michael Drumheller. Drumheller's paper, "Mobile Robot Localization Using Sonar,"⁶³ was eventually published in the *IEEE Transactions on Pattern Analysis and Machine Intelligence* in 1987 and became a classic.

Stereo with a single camera.

Ray Jarvis at Monash University in Australia came up with a clever way of gathering rectified stereo images from a single camera. He used a prism to project two slightly different viewpoints onto the lens of a camera, creating an image which had a different image on each side. The algorithm knew which pixels belong to each image, so there was no problem with processing.

III

Deliberative Functionality

Contents:

- [Chapter 12: Deliberation](#)
- [Chapter 13: Navigation](#)
- [Chapter 14: Metric Path Planning and Motion Planning](#)
- [Chapter 15: Localization, Mapping, and Exploration](#)
- [Chapter 16: Learning](#)

12

Deliberation

Chapter Objectives:

- Solve a simple deliberation problem using *Strips*, given an initial *world model*, *operators*, *difference table*, and *difference evaluator*. Show the state of the world model after each step.
- Describe the *Mission Planner*, *Navigator*, and *Pilot* organization of the *Nested Hierarchical Controller*.
- Define the *physical symbol grounding problem* and *anchoring*; give an example.
- Compare and contrast the advantages and disadvantages of *local perceptual spaces* versus multi-level hierarchical models for creating, maintaining, and using world models.
- Describe the role and operation of the *Sequencer* and *Skill Manager* agents in the 3T architecture and how it incorporates reactivity and deliberation.
- Contrast the *fail upwards* and *model-based reasoning* approach to *fault detection, identification, and recovery* (FDIR), and give an advantage and disadvantage of each.
- Define *skill*, *virtual sensor*, and *cognizant failure*.

12.1 Overview

Chapters 6 to 8 provide the essentials of a behavior-based robot and how to achieve animal-like autonomy. However, what we value as “real” cognitive intelligence is deliberative, not reactive. The desire to understand deliberation leads to the question addressed in this chapter: *How do robots think?* Returning to chapter 4, deliberative intelligence is associated with generating goals and intents, selecting or planning how to best meet those goals and intentions and implementing the specific actions and resources to carry out the plan, and finally monitoring the execution of the plan and replanning if it is not working. The methods for deliberative intelligence involve most aspects of artificial intelligence, especially the disciplines of planning and problem solving, with their subfields of scheduling, resource allocation, and reasoning.

Because it is impossible to cover the entire set of advances in the set of disciplines, this chapter will attempt to introduce the reader to the terminology and the conceptual framework needed for further

reading. Fortunately most of the terminology and conceptual framework stem from the inception of the first AI robot, Shakey (figure 2.6) at SRI in 1967. Shakey reportedly got its name because it was top heavy; as it rolled forward, the TV antenna would wobble and shake.

Shakey was the quintessential hierarchical system: SENSE, then PLAN, then ACT. It investigated the idea of a general problem solver in planning and problem solving, captured within the Strips algorithm, that would enable Shakey to reason and to plan a path. Strips is rarely used in robotics, but familiarity with it is important for three reasons: One, it will serve to motivate the reader to examine the computer challenges inherent in even as simple a task as walking across a room. Two, Strips is interesting historically because it formalized the concepts of preconditions, closed-world assumption, open-world assumption, and the frame problem. Third, and most importantly, the programming design choices of knowledge representation and reasoning algorithms introduced in Strips continues to pervade AI robotics, and thus, understanding Strips is a prerequisite for following advanced research.

Shakey and Strips concentrated on generating the perfect plan for scheduling the actions of the robot, but the methodology glossed over the critical issue of how what a robot sensed could be converted into a world representation. As Shakey could “see” only large, brightly colored objects, there was not a lot to represent anyway. However, for modern robots, sensors and sensing are more advanced, thereby providing a richer set of inputs, and the expectations of intelligence necessitate creating more sophisticated world models. This chapter will review basic concepts.

Strips also largely ignored how the actions were implemented; again, pragmatically, it was a basic robot so there were not really any options. The Nested Hierarchical Controller became a cornerstone of navigational planning through the 1990s. Later, in the mid-1990s, AI roboticists began tackling the general connection between plans and schedules and their implementation. RAPS and the 3T architecture were the initial forays into combining planning and behaviors. 3T is an excellent example of a system that can generate high level plans and propagate those plans down to controlling specific actuators and thus is summarized in this chapter.

Shakey and Strips assumed that the robot would complete the entire SENSE, PLAN, ACT cycle at each update, but unlike the Reactive paradigm, the SENSE step included updating a comprehensive global model. Sensing the change in the world from the action produced de facto monitoring the execution of the plan. If the robot was supposed to move to a door and had not reached it, the new plan should provide the most up-to-date optimum path or solution. Of course, if the robot had a breakdown and did not have internal sensing to notice, it could not recover. Later work in artificial intelligence began tackling monitoring, how to notice when something is going wrong, and then to reason about it. These efforts generally involve planning and problem solving, inference, and search. One major school of thought in monitoring and reasoning to recover from problems uses models of the robot, as in knowledge representations of the components of the robot, how the components are related, and statistical models of performance. Model-based reasoning for fault detection, identification, and recovery is best exemplified by the Deep Space One probe.

Shakey, Strips, and the approach to deliberation that they spawned is a major conceptual departure from traditional control theory. The focus on explicit knowledge representations and the use of predicate logic can be tedious and bewildering for those trained in control theory. Another barrier to understanding and implementing deliberation is the choice of programming languages. AI researchers shift from coding behaviors in procedural programming languages like C++ to coding deliberative

functions in functional languages, such as Lisp, which are better suited for coding and executing classic artificial intelligence algorithms.

12.2 Strips

Shakey, the first AI mobile robot, needed a generalized algorithm for planning how to accomplish goals. For example, it would be useful to have the same program allow a human to type in that the robot is in Office 311 and should go to Office 313 or that the robot is in 313 and should deliver the red box.

GENERAL PROBLEM SOLVER (GPS)

STRIPS

MEANS-ENDS ANALYSIS

The method finally selected was a variant of the *General Problem Solver* method called *Strips*. Strips uses an approach called *means-ends analysis*, where if the robot cannot accomplish the task or reach the goal in one “movement,” it picks an action which will reduce the difference between what state it was in now (e.g., where it was) versus the goal state (e.g., where it wanted to be). This method was inspired by cognitive behavior in humans; if you cannot see how to solve a problem, you try to solve a portion of the problem to see if it gets you closer to the complete solution.

GOAL STATE

INITIAL STATE

OPERATOR

DIFFERENCE

DIFFERENCE EVALUATOR

Consider trying to program a robot to figure out how to get to the Stanford AI Lab (SAIL). Unless the robot is at SAIL (represented in Strips as a variable `goal state`), some sort of transportation will have to be arranged. Suppose the robot is in Tampa, Florida (`initial state`). The robot may represent the decision process of how to get to a location as function called an `operator` which would consider the Euclidean distance (a variable named `difference`) between the `goal state` and `initial state`. The difference between locations could be computed for comparison purposes, formally called “`evaluation`.” The mathematical function that computed the distance is referred to as the `difference evaluator`. For example, using an arbitrary (X, Y) frame of reference that puts Tampa at the center of the world with made-up distances to Stanford:

initial state:	Tampa, Florida (0,0)
goal state:	Stanford, California (1000,2828)
difference:	3,000

DIFFERENCE TABLE

This difference-oriented approach could lead to a data structure called a `difference table` of explicitly how to make decisions:

difference	operator
$d \geq 200$	fly
$100 < d < 200$	ride_train
$d \leq 100$	drive
$d < 1$	walk

Different modes of transportation are appropriate for different distances. A mode of transportation, `fly`, `ride_train`, `drive`, `walk`, in the table is really a function in the robot's program. It is also called an operator because, if applied, it reduces the distance between `initial state` of Tampa and the `goal state`. A robot following this difference table would begin by planning to fly as close as it could to SAIL.

PRECONDITIONS

But suppose the robot flew into the San Francisco airport. It would be within 100 miles of SAIL, so the robot would appear to have made an intelligent decision. But the robot has not reached its goal and thus has a new difference to reduce, that is, to cover those 100 miles between the airport and SAIL. It examines the difference table with a new value of `difference`. The table says the robot should `drive`. Drive what? A car? Oops: if the robot did have a personal car, it would be back in Tampa. The robot needs to be able to distinguish between driving its car and driving a rental car. This distinction is made by listing the preconditions that have to be true in order to execute that particular operator. The preconditions are represented by a column in the difference table; note that a single operator can have multiple preconditions. In practice, the list of preconditions is quite lengthy, but, for the purposes of this example, only `drive_rental`, `drive_personal` will be shown with preconditions.

difference	operator	preconditions
$d \leq 200$	fly	
$100 < d < 200$	ride_train	
$d \leq 100$	drive_rental	at airport
	drive_personal	at home
$d < 1$	walk	

ADD-LIST

DELETE-LIST

The difference table enables the robot to drive a rental car if it is at an airport. But this addition to the difference table introduces a new issue: How does the robot know where it is at? The robot knows where it is by monitoring its state and its world. If the robot took an airplane from Tampa to the San Francisco airport, its state has changed. Its `initial state` is now the San Francisco airport and no longer Tampa. Therefore, whenever the robot executes an operator, there is almost always something that has to be added to the robot's knowledge of the world state (which is entered into an `add-list`) and something that has to be deleted (`delete-list`). These two lists are stored in the difference table so that when the robot selects an operator based on the difference and then executes it, it can easily apply the appropriate modifications to the world. The difference table below is expanded to show the add and delete lists.

difference	operator	pre-conditions	add-list	delete-list
$d \leq 200$	fly		at Y at airport	at X
$100 < d < 200$	train		at Y at station	at X
$d \leq 100$	drive_rental	at airport		
	drive_personal	at home		
$d < 1$	walk			

Of course, the above difference table is fairly incomplete. Driving a rental car should have a precondition that there is a rental car available. (And that the robot has a waiver from the state highway patrol to drive as an experimental vehicle and has a satisfactory method of payment.) The number of facts and preconditions that have to be explicitly represented seem to be growing explosively, which is Very Bad from a programming standpoint.

Ignoring details for now, the main point is that the difference table appears to be a good data structure for representing what a robot needs to plan a trip. A recursive function can be written to examine each entry in the table in order to find the first operator that reduces the difference, then to repeat and reduce that difference and so on until the goal is reached. The resulting list of operators is actually the plan: a list of the steps (operators) that the robot has to perform in order to reach a goal. Note that the robot constructs the entire plan before handing it off to another program to execute, because the planning algorithm may try an operator, then later discover that it will not work and have to backtrack to try another operator.

At this point in time, it is not likely that a robot will get on a plane and then drive. So perhaps any criticisms of Strips are because the example used is too complicated a task to be realistic. Let us explore if Strips becomes more streamlined when applied to a simple task of getting from one room to another.

12.2.1 More Realistic Strips Example

CONSTRUCTING A WORLD MODEL

AXIOMS

PREDICATES

The first step in creating a Strips planner is to construct a Strips-based representation of the world, or *world model*. Everything in the world that is relevant to the problem is represented by facts, or *axioms*, in predicate logic. *Predicates* are functions that evaluate to TRUE or FALSE. By AI programming convention, predicates are written in uppercase.

Consider the problem of a robot named IT in a room, R1, where the robot needs to go to another room, R2, in the house shown in [figure 12.1](#). In order to solve this problem using Strips, the robot has to be given a way of representing the world, which will, in turn, influence the difference table, a difference evaluator, and how the add and delete lists are written. The world model in the previous example was never formally defined.

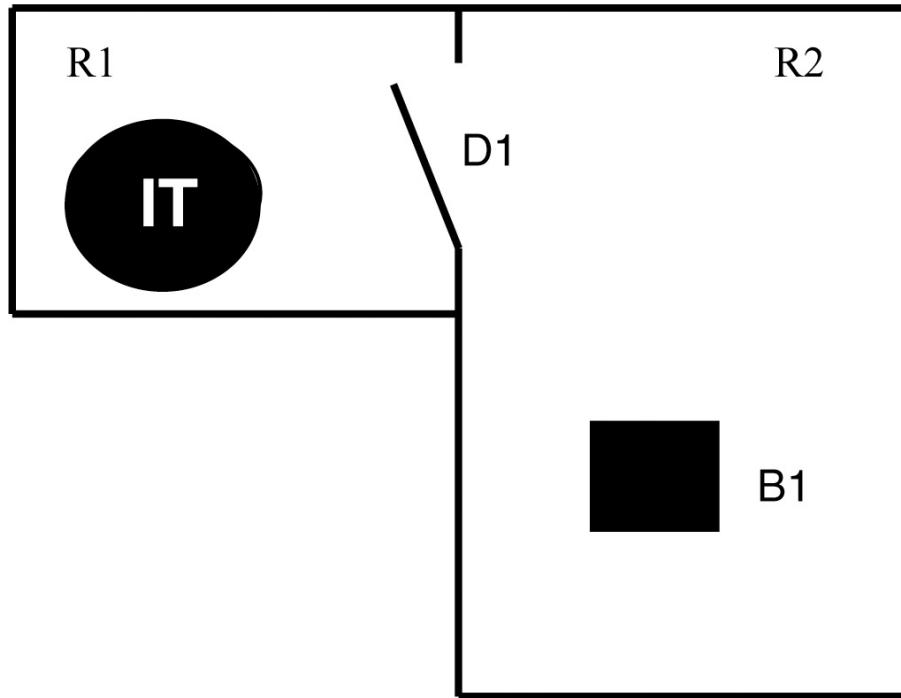


Figure 12.1 An example for Strips of two rooms joined by an open door.

A world model is generally built up from static facts (represented as predicates) from a set of candidates and things in the world like the robot. The robot's name is in capital letters because it exists (formally it is TRUE). Lower-case identifiers indicate that a thing is a variable, that a real thing has not been assigned to that placeholder yet.

Suppose the robot was limited to knowing only whether a movable object, such as a box, was in a room, next to a door or another movable object, and whether a door was open or closed and what rooms the door connected. In a programming sense, there would be only three types of things in the world: `movable_object` (such as the robot or a box), `room`, and `door`. The robot's knowledge could be represented by the following predicates:

<code>INROOM (x, r)</code>	where x is an object of type <code>movable_object</code> , r is type <code>room</code>
<code>NEXTTO (x, t)</code>	where x is a <code>movable_object</code> , t is type <code>door</code> or <code>movable_object</code>
<code>STATUS (d, s)</code>	where d is type <code>door</code> , s is an enumerated type: <code>OPEN</code> or <code>CLOSED</code>
<code>CONNECTS (d, rx, ry)</code>	where d is type <code>door</code> , <code>rx</code> , <code>ry</code> are the room

With the above predicates, the world model for the initial state of the world in [figure 12.1](#) would be represented by:

```

initial state:
INROOM(IT, R1)
INROOM(B1, R2)
CONNECTS(D1, R1, R2)
CONNECTS(D1, R2, R1)
STATUS(D1, OPEN)

```

This world model captures that a specific `movable_object` named `IT` is in a room named `R1`, and `B1` is in another room labeled `R2`. A door `D1` connects `R1` to `R2` and it connects `R2` to `R1`. (Two different `CONNECTS` predicates are used to represent that a robot can go through the door from either room.) A door called `D1` has the enumerated value of being `OPEN`. The `NEXTTO` predicate was not used because it was not true and there would be nothing to bind the variables to.

Under this style of representation, the world model for the goal state would be:

```

goal state:
INROOM(IT, R2)
INROOM(B1, R2)
CONNECTS(D1, R1, R2)
CONNECTS(D1, R2, R1)
STATUS(D1, OPEN)

```

CONSTRUCTING THE DIFFERENCE TABLE

Once the world model is established, it is possible to construct the difference table. The partial difference table is:

operator	preconditions	add-list	delete-list
OP1: GOTODOOR (IT, dx)	INROOM (IT, rk) CONNECT (dx, rk, rm)	NEXTTO (IT, dx)	
OP2: GOTHRUUDOOR (IT, dx)	CONNECT (dx, rk, rm) NEXTTO (IT, dx) STATUS (dx, OPEN) INROOM (IT, rk)	INROOM (IT, rm)	INROOM (IT, rk)

This difference table says the robot is programmed for only two operations: go to a door, and go through a door. The `GOTODOOR` operator can be applied only if the following two preconditions are true:

- `INROOM(IT, rk)` The robot is in a room, which will be assigned to the identifier `rk`.
- `CONNECT(dx, rk, rm)` There is a door, which will be assigned to the identifier `dx`, which connects `rk` to some other room called `rm`.

The label `IT` is used to constrain the predicates. Notice that only the variables `dx` and `rk` get bound when `GOTODOOR` is called. `rm` can be anything. If `GOTODOOR` is executed, the robot is now next to the door called `dx`. Nothing gets deleted from the world state because the robot is still in room `rk`; the door `dx` still connects the two rooms `rk` and `rm`. The only thing that has changed is that the robot is now in a noteworthy position in the room: next to the door.

The difference table specifies that the `GOTHRUUDOOR` operator will work only if the robot is in the room, next to the door, the door is open, and the door connects the room the robot is in to another

room. In this case, predicates must be added and deleted from the world model when the operator executes. When the robot is in room rk and goes through the door, it is now in room rm (which must be added to the world model), and it is no longer in room rk (which must be deleted).

So far, the world model and difference table should seem reasonable, although tedious, to construct. But constructing a difference table is pointless without an evaluation function for differences. (Notice that there was not a column for the difference in the above table.) The difference evaluator in the travel example was Euclidean distance. In this example, the evaluator is predicate calculus, where the initial state is logically subtracted from the goal state. The logical difference between the initial state goal state is simply:

$$\neg \text{INROOM}(IT, R2) \text{ or } \text{INROOM}(IT, R2) = \text{FALSE}$$

REDUCING DIFFERENCES

Reducing differences is a bit like solving a jigsaw puzzle, where Strips tries different substitutions to see if a particular operator will reduce the difference. In order to reduce the difference, Strips looks in the difference table for a match under the add-list column, starting at the top. It looks in the add-list column rather than a separate differences column because the add-list entry for an operator expresses the result of that operator. If Strips finds an operator that produces the goal state, then that operator eliminates the existing difference between the initial and goal states.

The add-list entry for OP2: GOTHRUDOOR has a match on form. If $rm=R2$, then the result of OP2 would be $\text{INROOM}(IT, R2)$, which would eliminate the difference; so OP2 is a candidate operator.

Before the OP2 can be applied, Strips must check the preconditions. To do this, rm must be replaced with $R2$ in every predicate in the preconditions. OP2 has two preconditions, only CONNECTS(dx , rk , rm) is affected. It becomes CONNECTS(dx , rk , $R2$). Until dx and rk are bound, the predicate does not have a true or false value. Essentially dx , rk are wildcards, CONNECTS(*, *, $R2$). To fill in values for these variables, Strips looks at the current state of the world model to find a match. The predicate in the current state of the world CONNECTS($D1$, $R1$, $R2$) matches CONNECTS(*, *, $R2$). $D1$ is now bound to dx and $R1$ is bound to rk .

FAILED PRECONDITIONS

Now Strips propagates the bindings to the next precondition on the list: NEXTTO(IT , dx). NEXTTO(IT , $D1$) is FALSE because the predicate is not in the current world state. NEXTTO(IT , $D1$) is referred to as a *failed precondition*. An informal interpretation is that GOTHRUDOOR(IT , $D1$) will get the robot to the goal state, but, before it can do that, IT has to be next to D1.

RECURSION TO RESOLVE DIFFERENCES

Rather than give up, STRIPS *recurses* (uses the programming technique of recursion) to repeat the entire procedure. It marks the original goal state as $G0$, pushes it on a stack, and then it creates a new subgoal state, $G1$, of NEXTTO(IT , $D1$).

The difference between NEXTTO(IT , $D1$) and the current world state is:

$$\neg \text{NEXTTO}(IT, D1)$$

Strips once again searches through the add-list column in the difference table to find an operator that

will negate this. Indeed, OP1: `GOTODOOR(IT, dx)` has a match in the add-list of `NEXTTO(IT, dx)`. Strips has to start over with reassigning values to the identifiers because the program has entered a new programming scope, so `dx=D1`.

Again, Strips examines the preconditions. This time `rk=R1` and `rm=R2` can be matched with `CONNECTS(dx, rk, rm)`, and all preconditions are satisfied (that is, they are evaluated as true). Strips puts the operator `OP1` on the plan stack and applies the operator to the world model, changing the state. (Note that this is the equivalent of a “mental operation.” The robot does not actually go to the door; it just changes the state to imagine what would happen if it did.)

Recall that the initial state of the world model was:

```
initial state:  
INROOM(IT, R1)  
INROOM(B1,R2)  
CONNECTS(D1, R1, R2)  
CONNECTS(D1, R2, R1)  
STATUS(D1,OPEN)
```

Applying the operator `OP1` means making the changes on the add-list and delete-list. There is a predicate only on the add-list to change, and none on the delete-list. After adding `NEXTTO(IT, D1)`, the state of the world is:

```
state after OP1:  
INROOM(IT, R1)  
INROOM(B1,R2)  
CONNECTS(D1, R1, R2)  
CONNECTS(D1, R2, R1)  
STATUS(D1,OPEN)  
NEXTTO(IT, D1)
```

Strips then returns control to the previous call. It resumes where it left off evaluating the preconditions for `OP2` with `dx=D1`, `rm=R2`, and `rk=R1`. Only now the world model has changed. Both `STATUS(D1, OPEN)` and `INROOM(IT, R1)` are true, so all the preconditions for `OP2` are satisfied. Strips puts `OP2` on its plan stack and changes the world model by applying the add-list and delete-list predicates. This is what the state of the world will be when the plan is executed:

```
state after OP2:  
INROOM(IT, R2)  
INROOM(B1,R2)  
CONNECTS(D1, R1, R2)  
CONNECTS(D1, R2, R1)  
STATUS(D1,OPEN)  
NEXTTO(IT, D1)
```

Strips exits, and the plan for the robot to execute (in reverse order on the stack) is: `GOTODOOR(IT, D1)`, `GOTHRUDDOR(IT, D1)`.

12.2.2 Strips Summary

Strips works recursively; if it cannot reach goal directly, it identifies the problem (a failed precondition), and then makes the failed precondition a subgoal. Once the subgoal is reached, Strips puts the operator for reaching the subgoal on a list and then backs up (pops the stack) and resumes trying to reach the previous goal. Strips plans rather than executes: it creates a list of operators to apply, but it does not apply the operator as it goes. Strips' implementations requires the designer to set up a:

- world model representation,
- difference table with operators, preconditions, add and delete lists, and
- difference evaluator.

The steps in executing Strips are:

1. Compute the difference between the goal state and the initial state using the difference evaluation function. If there is no difference, terminate.
2. If there is a difference, reduce the difference by selecting the first operator from the difference table whose add-list has a predicate which negates the difference.
3. Next, examine the preconditions to see if a set of bindings for the variables can be obtained which are all true. If not, take the first FALSE precondition, make it the new goal, and store the original goal by pushing it on a stack. Recursively reduce that difference by repeating steps 2 and 3.
4. When all preconditions for an operator match, push the operator onto the plan stack and update a copy of the world model. Then return to the operator with the failed precondition so it can apply its operator or recurse on another failed precondition.

12.2.3 Revisiting the Closed-World Assumption and the Frame Problem

CLOSED-WORLD ASSUMPTION

FRAME PROBLEM

Strips sensitized the robotics community to two pervasive issues: the *closed-world assumption* and the *frame problem*. As defined earlier, the closed-world assumption says that the world model contains everything the robot needs to know: there can be no surprises. If the closed-world assumption is violated, the robot may not be able to function correctly. But, on the other hand, it is very easy to forget to put all the necessary details into the world model. As a result, the success of the robot depends on how well the human programmer can think of everything.

OPEN-WORLD ASSUMPTION

But even assuming that the programmer did come up with all the cases, the resulting world model is likely to be huge. Consider how big and cumbersome the world model was just for moving between two rooms. And there were no obstacles! People began to realize that the number of facts (or axioms) that the program would have to sort through for each pass through the difference table was going to become intractable for any realistic application. The problem of representing a real-world situation in a way that was computationally tractable became known as the frame problem. The opposite of the closed-world assumption is known as the *open-world assumption*. When roboticists say that “a robot

must function in the open world,” they are saying the closed-world assumption cannot be realistically applied to that particular domain.

The above example, although trivial, shows how tedious Strips is (though computers are good at tedious algorithms). In particular, the need to represent the world formally and then maintain every change about it is nonintuitive. Strips also illustrates the advantage of a closed-world assumption: imagine how difficult it would be to modify the planning algorithm if the world model could suddenly change. The algorithm could get lost between recursions. This example should also bring home the meaning of the frame problem: imagine what happens to the size of the world model if a third room is added with boxes for the robot to move to and pick up! And this is only for a world of rooms and boxes. Clearly the axioms which frame the world will become too numerous to use to describe any realistic domain.

One early solution was ABStrips, which tried to divide the problem into multiple layers of abstraction, that is, to solve the problem on a coarse level first. That had its drawbacks, and soon many people who had started out in robotics found themselves working in an area of AI called planning. The two fields became distinct, and by the 1980s, the planning and robotics researchers had separate conferences and publications. Many roboticists during the 1970s and 1980s worked either on computer vision related issues, trying to get the robots to be able to sense the world better, or on path planning, computing the most efficient route around obstacles, and so forth, to a goal location.

12.3 Symbol Grounding Problem

PHYSICAL SYMBOL GROUNDING

Strips relies on a world model. The world model is the knowledge structure where regions of the sensed world are labeled with symbols, such as hallway, door, my office, and so forth. The process of assigning symbols to objects or concepts is called *symbol grounding* in cognitive science. Robotics concentrates on *physical symbol grounding*, where data acquired about the physical world from sensors are assigned to symbols.⁵² The symbols represent objects or conditions of the world that have some permanence and thus persist. For example, the robot may pass a room that has an open door and then return later when the door is shut—How does the robot know it passed a room with a door or that it is the same room but the door is now shut or that someone must have shut the door?

General symbol grounding becomes even more important if a robot system is taking commands from natural language interfaces (“deliver this to John”) and then operationalizing that into navigation tasks and perceptual recognition routines. Now the robot has to translate “this” symbol into a concept and then into a tangible physical presence in the world; “this” means the package on the table in front of the human, not the table or the coffee cup. Such translations of symbols into physical objects and understanding what to do with those physical objects are often addressed through some form of deliberation about the human’s *beliefs, desires, and intentions*.

Understanding physical symbol grounding is a priority with the symbol grounding community, but it has remained elusive for decades. It is a priority because other types of symbol grounding, such as in natural language with smart phones, have more opportunities to resolve ambiguities (“I did not understand you”) and can structure the interaction. There are relatively fewer options of disambiguation in symbol grounding for navigating and comprehending the world. Another reason why physical

symbol grounding is challenging arises because of a variant of the frame problem: how to build and describe world models with the right amount of detail.

ANCHORING

Physical symbol grounding involves *anchoring*: assigning labels to perceptual inputs and thus anchoring perception to a common frame of reference that can be used by the deliberative functions of the robot.

Anchoring is usually accomplished through one of two general strategies:

- *Top-down object recognition* that matches existing symbols in the robot's knowledge representations. For example, the robot is programmed to find doors, rooms, hallways, dead-ends, "my coffee cup," and so forth, and has descriptions of how to recognize these from perceptual inputs. An advantage of this top-down strategy to symbol grounding is that it filters unnecessary or irrelevant perceptual information and identifies only what is important, that is, it imposes a de facto solution to the frame problem. It is a "find the physical manifestation of the symbol in the perceptual inputs" approach. A disadvantage of a top-down strategy in the open world is that it may miss something important because that something is not being searched for or is not in the knowledge base. Another disadvantage is that a concept or feature on a map may be difficult for a robot to reliably perceive. Thus there is a disconnect between internal representation and the external world.
- *Bottom-up object recognition*, where the robot senses an unknown object, determines it is of interest, declares it to be an object, and assigns it a label, Object -23. The label may later be revised to `CoffeeCup(My)`. One intriguing mechanism for revising the label is having the robot run a search on the World Wide Web to see if it can find any labeled image that matches what it is seeing. It is a "What is in the perceptual inputs?" approach. The advantage of the bottom-up approach is that it does not exclude unexpected objects. The disadvantage is determining how to perceptually group features into a coherent object—for example, understanding where a coffee cup ends and a table begins.

Although anchoring is necessary for symbol grounding, it is not sufficient. It may identify what the object is and where it is located but it does not provide the semantics of the object. For example, consider trying to determine that the perceptual input is "Donna's coffee cup" or that "It was left here after a meeting and belongs back in her office." There is not sufficient perceptual information to label the "coffee cup" with those pieces of information; deliberative capabilities, such as memory, reasoning, and inference, are required. As a result, a command to a robot to "Pick up Donna's coffee cup and put it back" may be stymied if there are two coffee cups in view unless there is more deliberative work being done to resolve the ambiguity.

12.4 Global World Models

If symbol grounding is hard for a single object such as `CoffeeCup(My)`, then how are the global world models used for deliberative capabilities constructed? One method is to have a global world model consisting of a collection of limited global world models called *local perceptual spaces*. Another, more popular, method is to create a world model with layers of abstraction about the physical world called

hierarchical global world models. The Nested Hierarchical Controller is the most pervasive of hierarchical global world models and thus will be covered in a separate section. Both types of world models can be used by the reactive layer as *virtual sensors*. However, local perceptual spaces and hierarchical global models focus on anchoring, determining what objects are where, while generally ignoring the incorporation of other knowledge ideally included in a global world model.

12.4.1 Local Perceptual Spaces

LOCAL PERCEPTUAL SPACE

Local perceptual spaces were introduced by Konolige and Myers¹⁰⁶ as part of Erratic, the grandchild of Shakey, shown in [figure 12.2](#). The idea is that the robot creates and maintains a model of the region it is operating in. For example, a robot might roll into a room and create a 3D scan of the room and detect and anchor any objects of interest within that room. It may then roll into another room and create a model of that room, leading to a global world model of two very rich local models of specific rooms that are connected by a model of a hallway. Anchoring may be limited to whatever is of relevance to the robot's task, thus addressing the frame problem. What was very innovative about local perceptual spaces is that they could be created and shared by a team of robots. One robot could sense one room while another robot simultaneously sensed another room, and then the two could share their perceptions and form a fused, single global world model. The architectural layout for Erratic, called the Saphira architecture, is shown in [figure 12.3](#).



Figure 12.2 Erratic, a Pioneer robot. (Photograph courtesy of Kurt Konolige.)

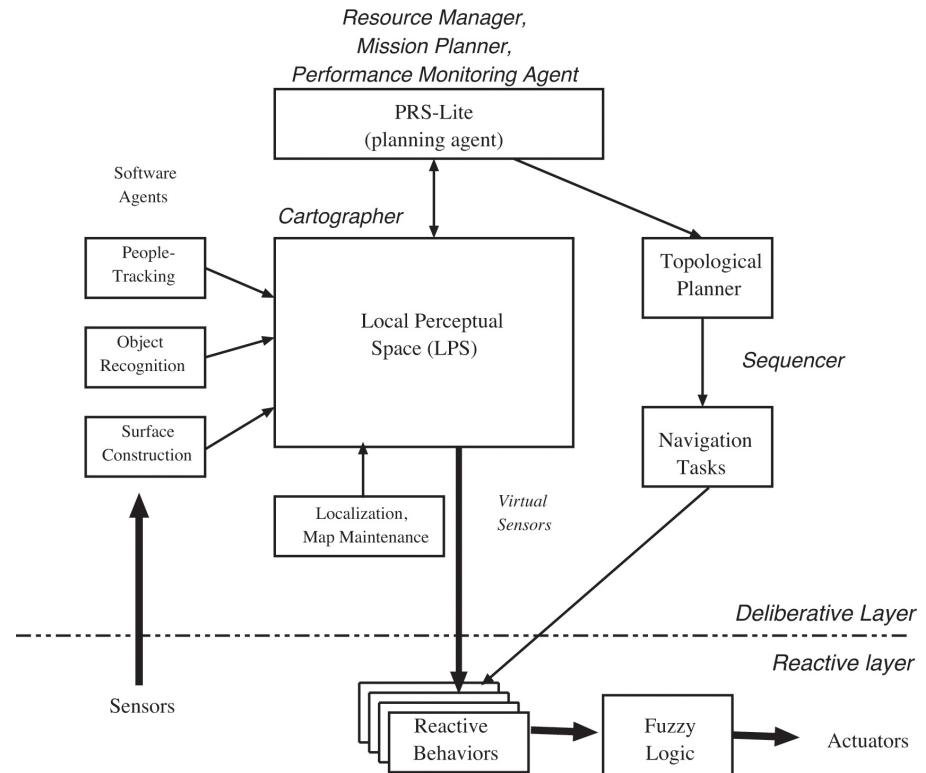


Figure 12.3 Simplified view of Saphira architecture.

Local perceptual spaces are the opposite of most simultaneously localization and mapping algorithms described in chapter 15, which try to create a single 3D representation of everything in the world without anchoring specific objects. A 3D map of the environment is an impoverished global world model because there is no symbol grounding.

12.4.2 Multi-level or Hierarchical World Models

Multi-level or hierarchical world models create a world model that consists of layers of abstraction. The lowest level might be a 2D or 3D spatial representation of the world. The next higher, or more abstract, layer might be a topological spatial representation that labels portions of the world as “rooms,” “doors,” and “hallways.” The next layer might label the objects within a room or describe the color of the walls.

The layers are spatially registered, so that if a deliberative function searches the world model for coffee cups and finds that a large collection of them exist, it can tie that finding to “the kitchen” and know where the kitchen is. This methodology is similar to that of perceptual hierarchies or perceptual pyramids in computer vision where an image of an office is abstracted into layers and labels that represent the region until there is one labeled “Murphy’s office.”

Layering, as a strict hierarchy of abstractions, from spatial representation to symbol, proved impractical because of the many types of deliberation functions that were needed to store and use

information about the world. Indeed, the term “global” is used almost synonymously with “deliberative” and “local” with “reactive.” This synonymous terminology can lead to significant confusion because “global” is not always truly global. The deliberative portion of a hybrid architecture contains modules and functions for things which are not easy to represent in reactive behaviors. Some of these functions clearly require a global world model; path planning and map making are probably the best examples.

But other activities require global knowledge of a different sort. Behavioral management (planning which behaviors to use) requires knowing something about the current mission and the current (and projected) state of the environment. This is global knowledge in that it requires the module to know something outside of itself as compared to a reactive behavior which can function without any knowledge of whether there are other behaviors actively executing. Likewise, performance monitoring, to see if the robot is actually making progress to its goal, and problem solving are global activities. Consider writing a program to diagnose whether the robot is not moving forward because there is a problem with the terrain (it is stuck in the mud) or with a sensor (the shaft encoders do not report wheel turns correctly). In order to perform the diagnostics, the program has to know what the behaviors were trying to accomplish, if there are any other sensors or knowledge sources to corroborate any hypotheses, and so forth. Therefore, a deliberative function may not need a global world model, but it may need to know about the internal workings of the robot on a global scale, if only to know what other modules or deliberative capabilities the program should interact with.

12.4.3 Virtual Sensors

VIRTUAL SENSOR

The global world model can also serve to supply perception to the behaviors in which case, the global world model serves as a *virtual sensor*. Instead of perceptual schemas operating on sensor input, the perceptual schema can extract percepts from the world model. A logical sensor treats the world model as if it were sensor input.

As an example of logical sensors, consider a robot trying to navigate in a cluttered difficult-to-sense environment. The robot has a `avoid-obstacle` behavioral schema that uses a `extract-nearest-obstacle` perceptual schema connected to a 3D scanner. The robot may also be building a very accurate 3D representation of the world which has better depth information than a single 3D scan of the world. The global world model can be converted to a virtual 3D scan to be used as the input to perceptual schema. The advantage is that sensor errors and uncertainty can be filtered by the world model using sensor fusion over time. This can dramatically improve the performance of the robot without changing any of the behavioral schemas. This method maintains the modularity and design philosophy of building on behaviors, not replacing them.

12.4.4 Global World Model and Deliberation

The use of a single global world model for sensing appears to be a throwback to the Hierarchical Paradigm, and conceptually it is, but it is essential for deliberation. It is not always the case that the deliberator generates the set of behaviors, turns them on, and lets them execute until they either complete the subtask or fail. In the case of sensing, it might be desirable for the deliberator to make

available some of the global models being constructed to the behaviors. For example, consider a robot vehicle following a road. Now suppose that it is looking for the large tree, since it is supposed to turn right at the intersection after the tree. In order to maintain the correct sequence of behaviors, the deliberator has a global world model where trees are noted. The reactive behaviors of following a road and avoiding an obstacle do not need the explicit representation of a tree. But, what if the tree casts a shadow which might confuse the follow-path behavior and cause the behavior to give the robot incorrect steering commands? In this case, it would be useful if the information about the presence of a tree could be absorbed by the behavior. One way to do this is to permit methods on the world model to act as virtual sensors or perceptual schema. Then the follow-road behavior could use the virtual sensor, which tells the behavior when the road boundaries extracted by the vision sensor are probably distorted and to ignore affected regions in the image that would normally be observed by the vision sensor. This is an example of how the deliberative layer can aid with selective attention, or filtering of perception for a context.

12.5 Nested Hierarchical Controller

NESTED HIERARCHICAL CONTROLLER

The *Nested Hierarchical Controller* (NHC) has influenced the design of global world models since its creation by Alex Meystel¹²⁹ in 1987. It uses a hierarchical global world model, where each layer supports a particular deliberative capability or agent. The organization is intuitive and easy to implement; however, it only applies to navigation.

As shown in [figure 12.4](#), the Nested Hierarchical Controller architecture is a classic hierarchical system. It has components that are easily identified as either SENSE, PLAN, or ACT, uses them in a sequence, and relies on a global world model. The robot begins by gathering observations from its sensors and combining those observations to form the World Model data structure through the SENSE activity. The World Model may also contain a priori knowledge about the world—for example, maps of a building, rules saying to stay away from the foyer during the start and finish of business hours, and so forth. After the World Model has been created or updated, then the robot can PLAN what actions it should take. Planning for navigation has a local procedure consisting of three steps executed by the Mission Planner, Navigator, and Pilot. Each of these modules has access to the World Model in order to compute their portion of planning. The last step in planning is for the Pilot module to generate specific actions for the robot to do (for example, turn left, turn right, move straight at a velocity of 0.6 meters per second). These actions are translated into actuator control signals (for example, velocity profile for a smooth turn) by the Low-Level Controller. Together, the Low-Level Controller and actuators form the ACT portion of the architecture.

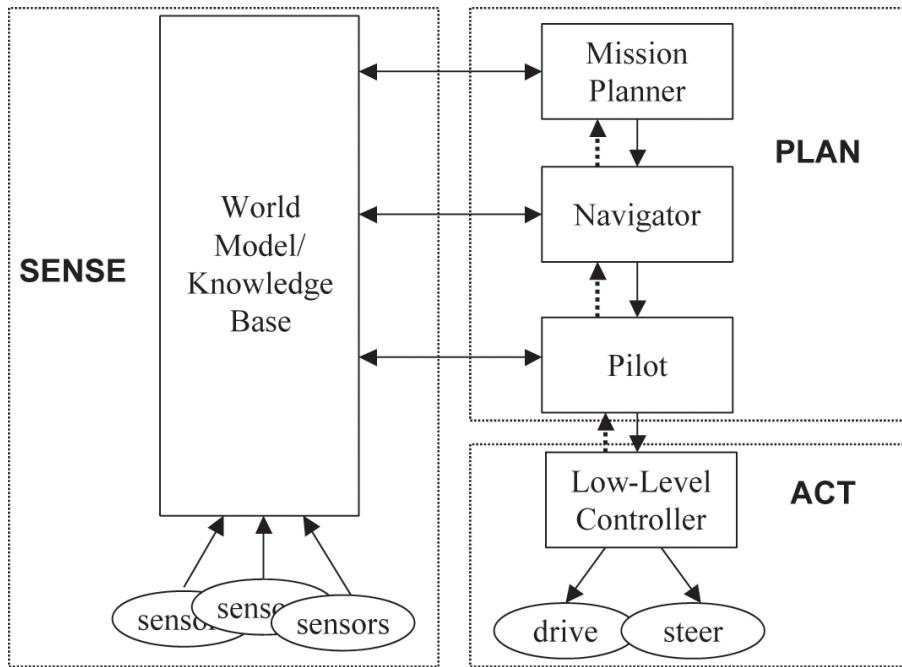


Figure 12.4 Nested Hierarchical Controller.

MISSION PLANNER

NAVIGATOR

PILOT

The major contribution of NHC was its clever decomposition of planning into three different functions or subsystems aimed at supporting navigation: the *Mission Planner*, the *Navigator*, and the *Pilot*. As shown in [figure 12.5](#), the Mission Planner either receives a mission from a human or generates a mission for itself—for example, pick up the box in the next room. The Mission Planner is responsible for operationalizing, or translating, this mission into terms that other functions can understand: box=B1; rm=ROOM311. The Mission Planner then accesses a map of the building and locates where the robot is and where the goal is. The Navigator takes this information and generates a path from the current location to the goal. It generates a set of waypoints, or straight lines for the robot to follow. The path is passed to the Pilot. The Pilot takes the first straight line or path segment and determines what actions the robot has to do to follow the path segment. For instance, the robot may need to turn around to face the waypoint before it can start driving forward.

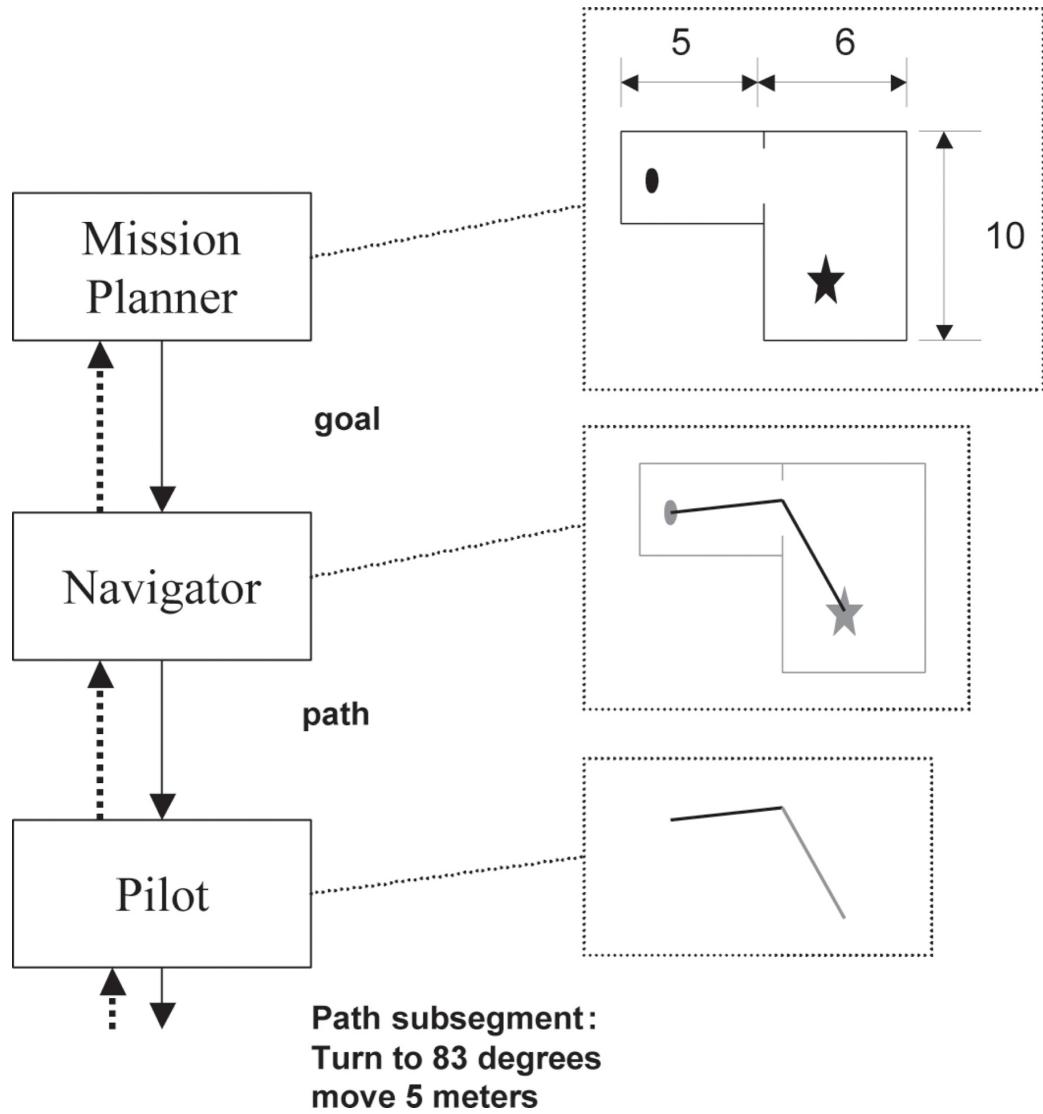


Figure 12.5 Examination of planning components in the NHC architecture.

What happens if the Pilot gives directions for a long path segment (say 50 meters) or if a person suddenly walks in front of the robot? Unlike Shakey, under NHC, the robot is not necessarily walking around with its eyes closed. After the Pilot gives the Low-Level Controller commands and the controller sends actuator signals, the robot polls its sensors again. The World Model is updated. However, the entire planning cycle does not repeat. Since the robot has a plan, it does not need to rerun the Mission Planner or the Navigator. Instead, the Pilot checks the World Model to see if the robot has drifted off the path subsegment (in which case it generates a new control signal), has reached the waypoint, or if an obstacle has appeared. If the robot has reached its waypoint, the Pilot informs the Navigator. If the waypoint is not the goal, then there is another path subsegment for the robot to follow,

and so the Navigator passes the new subsegment to the Pilot. If the waypoint is the goal, then the Navigator informs the Mission Planner that the robot has reached the goal. The Mission Planner may then issue a new goal, for example, Return to the starting place. If the robot has encountered an obstacle to its path, the Pilot passes control back to the Navigator. The Navigator must compute a new path and subsegments, based on the updated World Model. Then the Navigator gives the updated path subsegment to the Pilot to carry out.

NHC has several advantages. It differs from Strips in that it interleaves planning and acting. The robot comes up with a plan, starts executing it, and then changes it if the world is different than it expected. Notice that the decomposition is inherently hierarchical in intelligence and scope. The Mission Planner is “smarter” than the Navigator, which is smarter than the Pilot. The Mission Planner is responsible for a higher level of abstraction than the Navigator, and so forth. We will see that other architectures, both in the Hierarchical and Hybrid paradigms, will make use of the NHC organization.

One disadvantage of the NHC decomposition of the planning function is that it is appropriate only for navigation tasks. The division of responsibilities seems less helpful, or clear, for tasks such as picking up a box, rather than just moving over to it. The role of a Pilot in controlling end-effectors is not clear. At the time of its initial development, NHC was never implemented and tested on a real mobile robot; hardware costs during the Hierarchical period forced most roboticists to work in simulation.

12.6 RAPS and 3T

REACTIVE PLANNING

Shakey considered deliberation in general, offering the General Problem Solving algorithm for all forms of deliberation. It was planning heavy. The Nested Hierarchical Controller simplified planning to focus on navigational planning, ignoring *why* the robot needed to go to that location or *what* it was to do when it got there. It was too navigation heavy. Fortunately during the 1990s, members of the general AI community had become exposed to the principles of reactive robots. The concept of considering an intelligent system, or agent, as being situated in its environment, combined with the existence proof that detailed, Shakey-like world representations are not always necessary, led to a new style of planning. This change in planning was called *reactive planning*. Many researchers who had worked in traditional AI became involved in robotics. One type of reactive planner for robots, Jim Firby’s *reactive-action packages (RAPs)*,⁷⁶ was integrated as a layer within the 3T architecture,²⁵ where aspects of Slack’s NaT system,¹⁹⁶ Gat’s subsumption-style ATLANTIS architecture,⁸⁰ and Firby’s RAPs system⁷⁶ were merged at NASA’s Jet Propulsion Laboratory under the initial direction of David Miller, and later refined by Pete Bonasso and Dave Kortenkamp at NASA’s Johnson Space Center. The idea of reactive planning for practical robot tasking, described below, is foundational.

As the name suggests, 3T divides itself into three layers, one clearly reactive, one clearly deliberative, and one which serves as the interface between the two. Figure 12.6 shows the layers actually running on three different computers at NASA Johnson Space Center controlling the Extra-Vehicular Activity Helper-Retriever (EVAHR) robot simulator. 3T has been used primarily for planetary rovers, underwater vehicles, and robot assistants for astronauts.

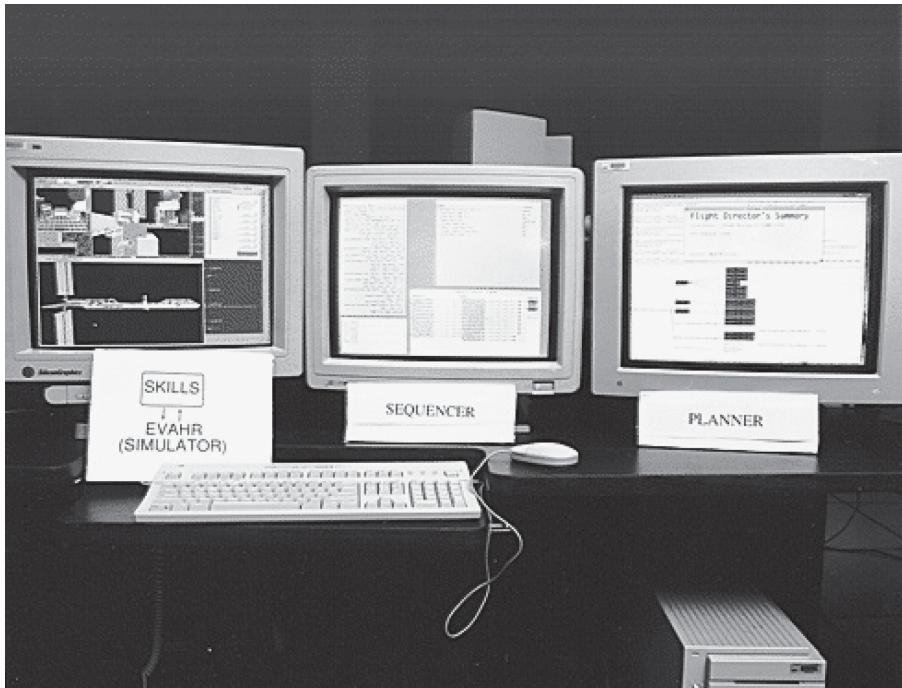


Figure 12.6 Each of the 3T layers running on a separate computer, controlling the EVAHR robot. (Photo courtesy of NASA Johnson Space Center.)

SEQUENCER

SKILLS

The top layer of 3T is Planner (see [figure 12.7](#)). It fulfills the duties of the Mission Planner and Cartographer modules by setting goals and generating strategic plans. These goals are passed to the middle layer, called the *Sequencer*. The Sequencer uses a reactive planning technique called RAPS to select a set of primitive behaviors from a library and develops a task network specifying the sequence of execution for the behaviors for the particular subgoal. The Sequencer is responsible for the sequencing and performance monitoring functions of a generic Hybrid architecture (recall chapter 4). The Sequencer layer instantiates a set of behaviors (called *skills*) to execute the plan. These behaviors form the bottom layer, called the Controller or Skill Manager. In order to avoid confusion with the connotations of purely reflexive behaviors left over from the Reactive Paradigm, 3T does not call its behaviors “behaviors.” It calls them “skills” to distinguish them from the connotations of behaviors popularized by the subsumption architecture. A skill is often an assemblage of primitive skills; indeed, one of the interesting aspects of 3T is its foundation as a tool for learning assemblages.

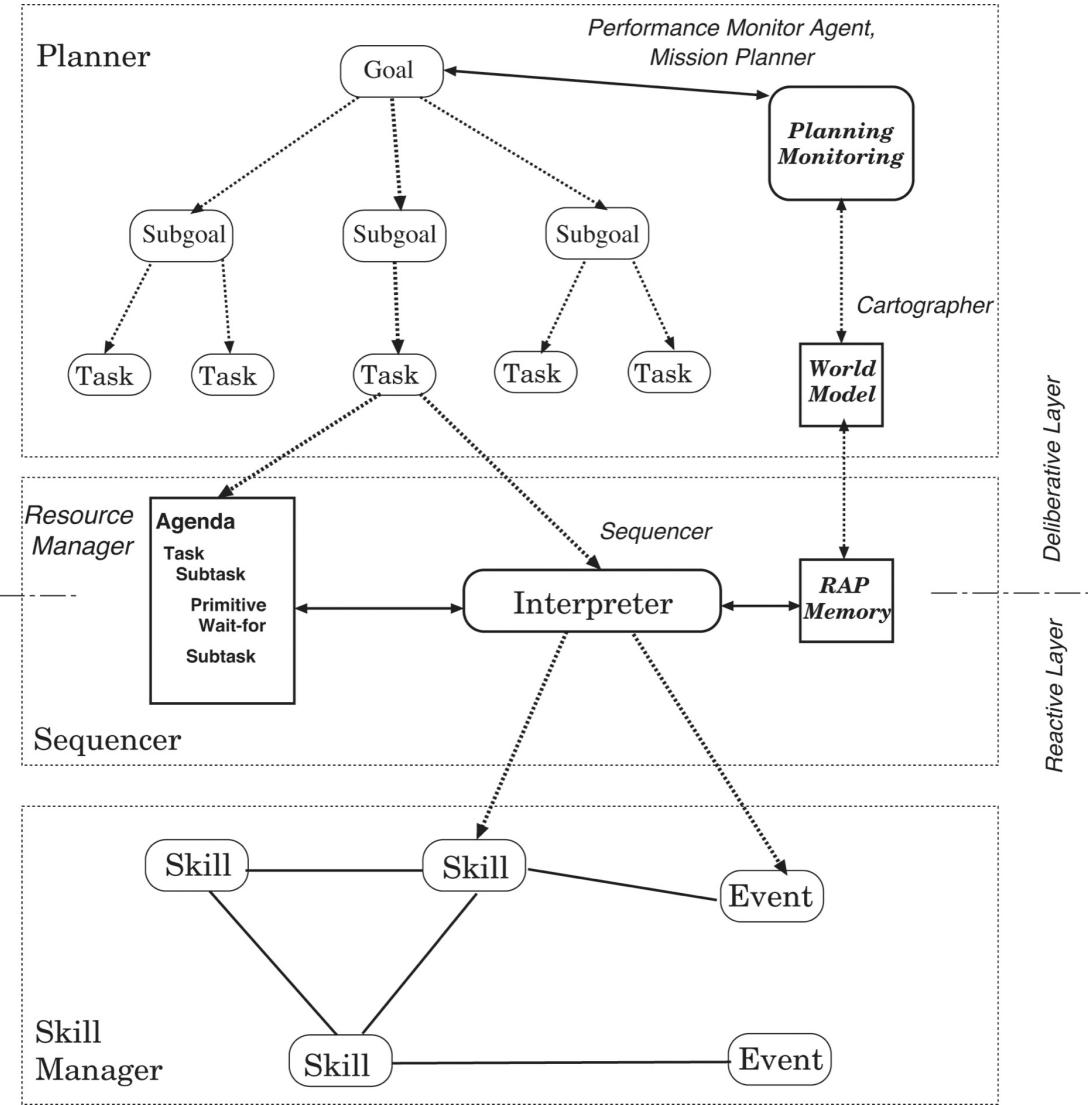


Figure 12.7 3T architecture.

EVENTS

A powerful attribute of the lower level is that the skills have associated *events*, which serve as checkpoints to explicitly verify that an action has had the correct effect. In some regards, events are equivalent to the innate releasing mechanisms; both let the world be its own best representation.

SKILL MANAGER

The three layers represent true deliberation, reactive planning, and reactive control. They also represent a philosophy organized by scope of state, rather than scope of responsibility. The *Skill Manager* layer is composed of skills that operate only in the Present (although with some allowances to

permit the persistence of some behaviors when the stimulus temporarily disappears). The components in the Sequencer layer operate on state information reflecting memories about the Past, as well as the Present. Therefore, sequences of behaviors can be managed by remembering what the robot has already done and whether that was successful or not. This adds a great deal of robustness and supports performance monitoring. The Planner layer works with state information to predict the Future. It can also use information from the Past (what the robot has done or tried) and the Present (what the robot is doing right now). In order to plan a mission, the planner needs to project what the environment will be and other factors.

UPDATE RATE

In practice, 3T does not strictly organize its functions into layers by state (Past, Present, Future). Instead it often uses *update rate*. Algorithms that update slowly are put in the Planner, while fast updating algorithms go into the Skill Manager layer. This appears to be a situation where the pragmatic considerations of computation influenced the design rules; in the early 1990s, behaviors were very fast, reactive planning (especially RAPs and Universal Plans) was fast, and mission planning was very slow. However, many sensor algorithms involving computer vision were also slow, so they were placed in the Planner, despite their low-level sensing function.

While robotics systems architectures typically have five common subsystems Planning, Cartographer, Navigation, Motor Schema, and Perception (recall chapter 4), 3T added a Sequencer to connect generating plans with selecting and implementing specific schemas and to aid in monitoring progress of the plan, called the *Agenda*. 3T was also influential in grouping behaviors into skills, or meta-behaviors, and skills into task networks.

12.7 Fault Detection Identification and Recovery

FAULT DETECTION IDENTIFICATION AND RECOVERY (FDIR)

The previous sections have presented deliberation as a classic planning problem of how the robot can accomplish a particular goal and, as seen with 3T, how to monitor progress. Deliberation is also necessary for problem solving, especially what to do if the robot is not accomplishing a particular goal. In engineering, this type of problem is often referred to as *fault detection identification and recovery (FDIR)*. Initial work in FDIR assumed that a system had to detect the fault, identify or diagnose what the fault was, and then apply the appropriate recovery methods, reminiscent of the SENSE-PLAN-ACT cycle. More recent work in FDIR is more reactive; first, detect that a fault is occurring, apply general recovery methods to preserve the system operation (e.g., not crash) while trying to identify precisely what the fault is, and then apply specialized recovery methods.

FAIL UPWARDS

In AI robotics, there have been two general approaches to handling problems. One is a “*fail upwards*” or *exception handling strategy* exemplified by the Nested Hierarchical Controller. In the NHC, the assumption was that a failure to make navigational progress would be discovered on the lowest level by the Pilot, and, if it could not be handled by the Pilot, then it would fail upwards to the Navigator. If the Navigator could not generate a detour, the system would fail upwards to the Mission Planner. Another system, the SFX architecture,¹³⁸ also uses this exception handling approach. Suppose

that a sensor breaks and the perceptual schema cannot produce a percept. The schema cannot diagnose itself. Therefore, its failure would cause a deliberative function to be triggered. If the schema manager cannot find a replacement perceptual schema or equivalent behavior, then it can no longer meet the constraints imposed by the Mission Planner. The Mission Planner presumably has the intelligence to relax constraints on the mission or, in practice, to alert the human supervisor.

Exception-handling approaches distribute and push monitoring to instantiated behaviors and functions. This creates a local perspective which can put the robot at risk of missing more subtle problems, or solutions, emerging from the larger interaction of subsystems. Exception handling approaches also tend to defer identification of the problem in favor of rapid recovery, which can be a benefit or cause even more problems if a new problem triggers the wrong response.

COGNIZANT FAILURE

Failing upwards or exception handling leads to the challenge of creating a *cognizant failure*, meaning that the failure analysis requires some awareness of the context to resolve the problem. It is not that the cognition itself has failed. Erann Gat identified the need for cognizant failures, calling it the “Wesson Oil” problem. In an old Wesson Oil TV commercial, a mother would be cooking chicken in oil when she would have to stop to rush her son to the emergency room for treatment of a broken arm. In an optimal world, the mother would not only turn off the stove but also remove the chicken from the oil. In the commercial, the mother neglects to remove the chicken. Since she is using Wesson Oil, the chicken isn’t ruined when they return, but it would have been with any other oil. The point of the Wesson Oil problem for robotics is *what happens when a set of reactive behaviors are turned off in the middle of a sequence?* We cannot assume that the robot will get lucky and not ruin the mission. In many cases, the solution will not be a simple act of de-instantiating the behaviors and activating new ones. There may be behaviors in the sequence that have to execute (turn off the stove) or disaster will occur as well behaviors which must run to prevent merely undesirable side effects (chicken gets greasy if it is not removed from the oil). This means that a planner or sequencer must know why a failure has occurred or why there is need to suddenly change behaviors and what the intent of the current sequence is. Deliberation is definitely not trivial!

MODEL-BASED REASONING

The second approach to handling problems is *model-based reasoning*, which was used by Deep Space One as discussed in chapter 3. Deep Space One, the NASA probe, used a global world model and a complete causal model of how all the systems worked together.²¹⁸ When a thruster valve became stuck in the closed position, the model-based reasoning system recovered by switching to a secondary control mode. The reasoning system also replanned to work around a stuck camera and repair an instrument by resetting it. The use of models simplified identification and recovery because the system has a model of the possible explanations for a failure. However, there could be thousands of explanations. Therefore to speed up recovery, many model-based reasoners also use probabilities to help prioritize searching for an explanation. The disadvantage of model-based FDIR is that it requires complete models of both the system and how it reacts to the environment; as apparent with reactive robots, predicting environments is difficult.

12.8 Programming Considerations

It is interesting to note that the use of predicate logic and recursion by Strips favors languages like Lisp and PROLOG. These languages were developed by AI researchers specifically for expressing logical operations but they do not necessarily have good real-time control properties like C or C++. However, during the 1960s, the dominant scientific and engineering language was FORTRAN IV, which did not support recursion. Therefore, researchers in AI robotics often chose the lesser of two evils (poor suitability for control versus lack of recursion) and program in Lisp. The use of special AI languages like Lisp for robotics may have aided the split between the engineering and AI approaches to robotics, as well as slowed the infusion of ideas between the two communities. It certainly discouraged non-AI researchers from becoming involved in AI robotics.

However, the software engineering reason for the partition remains: things which operate on symbols and global information should be in the deliberative layer; things which operate directly on sensors and actuators should be in the reactive layer. Symbol processing favors languages like Lisp while control favors languages like C++.

12.9 Summary

The chapter answers the question of *How does a robot think?* by providing an overview of the Deliberative Layer. In the Deliberative Layer, a robot engages six of the seven areas of artificial intelligence introduced in chapter 1. As seen with Shakey, *knowledge representation* is a core challenge. Shakey chose a knowledge representation of the world that supports the use of logic and the General Problem Solver style of planning and problem solving contained by Strips. The knowledge presentation consisted of a global world model with an associated knowledge base to contain a priori maps or knowledge relevant to a task. Strips relies on representing procedural knowledge about the world with some spatial information (e.g., CONNECTS). In addition, planning and problem solving methods assumed that the robot was able to perform *physical symbol grounding* in order to build those knowledge representations and to connect the execution of a plan to actual actions in the world and anchor changes in what is perceived. The reliance on a global world model introduces the frame problem.

Physical symbol grounding remains an open challenge in robotics and is explored by the *vision* and *natural language* communities. Flat or “everything-in-one-representation” models of the world are rare in practice, and, instead, some variant of local perceptual spaces or hierarchies of abstraction is common. Spatial representations of the world are popularly called “world models,” but they are impoverished and often of minimal use in advanced deliberation. They serve only as “shells” and do not contain the procedural knowledge about how the robot is situated in the world or what are the symbolic labels of objects. These shells can serve as virtual sensors to the behaviors and for navigation.

Planning and problem solving algorithms have been a major effort in creating deliberative capabilities for robots. Planning has often been restricted to planning navigational paths, with many systems mimicking a Nested Hierarchical Controller style of Mission Planner, Navigator, and Pilot partitioning of responsibilities and scope. Planning and problem solving often make extensive use of *search* algorithms, where Strips was essentially a recursive search engine relying on explicit preconditions for candidate operations.

Inference is used less frequently in practice, possibly because of the limited applications of robots,

but it is important to fault detection, identification, and recovery. The robot may not have time to explicitly confirm the cause of a problem and instead needs to guess close enough to rapidly generate a recovery strategy to avert a catastrophe.

Learning is the one area in AI that does not appear frequently in work on deliberation. This is most likely due to the ubiquity of learning which extends across the SENSE-PLAN-ACT cycle.

The next chapters will focus on deliberation for navigation, surveying the range of spatial knowledge representations and algorithms for planning the movement of a robot.

12.10 Exercises

Exercise 12.1

Define and give an example of

- a. cognizant failure
- b. physical symbol grounding problem
- c. anchoring.

Exercise 12.2

Give an example of how each of the following arises in a Strips-like robotic system:

- a. precondition
- b. closed-world assumption
- c. open-world assumption
- d. frame problem.

Exercise 12.3

Consider the frame problem. Suppose the World Model for a Strips-based robot consisted of 100 facts. Each fact requires 1KB of memory storage. Every time a new object is added to the world model, it increases the model by 100 facts (a linear increase). One object, 100 facts, 100KB of storage; two objects, 200 facts, 200KB. How many objects would fill 64KB of memory?

Exercise 12.4

Redo the above exercise if the number of facts in the world model doubles every time a new object is added. One object, 100 facts, 1KB; two objects, 200 facts, 200KB; three objects, 400 facts, 400KB. Would the rate of increase of the world model be linear or exponential?

Exercise 12.5

Describe the different purpose of the Mission Planner, Navigator, and Pilot in the *Nested Hierarchical Controller*.

Exercise 12.6

Describe how the Mission Planner, Navigator, and Pilot organization of the Nested Hierarchical Controller would handle the problem in section 12.2.1.

Exercise 12.7

Solve the 1994 AAAI Pick Up the Trash problem using Strips. Be sure to include a world model and a difference table.

Exercise 12.8

Solve the following navigation problem using Strips. Return to the world in section 12.2.1. The robot will move to the box B1 and pick it up.

- a. Add a new operator pickup to the difference table.
- b. Use the world model, difference table, and difference evaluator to construct a plan. Show failed preconditions and new subgoals after each step.
- c. Show the changes in the world model after *each* operator is applied.

Exercise 12.9

What is the difference between the fail upwards and model-based reasoning approach to FDIR. Give an advantage and disadvantage of each.

Exercise 12.10

Compare and contrast the advantages and disadvantages of local perceptual spaces versus multi-level hierarchical models for creating, maintaining, and using world models.

Exercise 12.11

Describe the role and operation of the Sequencer and Skill Manager agents in the 3T architecture and discuss how they incorporate reactivity and deliberation.

Exercise 12.12

What would be needed to convert a 3D scan of a room by a Kinect sensor into a global world model sufficient for picking up objects?

Exercise 12.13

Give three examples of reactive behaviors that might be performed better using a virtual sensor acting on a global world model.

Exercise 12.14

Return to the generating, monitoring, selecting, and implementing partitioning of deliberation. Do you think these can be implemented as separate modules or are they intertwined? Justify your answer.

[*World Wide Web*]

Exercise 12.15

Search the Web for interactive versions of Strips and experiment with them.

12.11 End Notes

GPS and GPS.

The General Problem Solver, pervasive in early AI, is abbreviated as GPS. When AI roboticists began exploiting Global Positioning Satellite systems, also abbreviated as GPS, for outdoor navigation, confusion surfaced. I and other colleagues had papers submitted to AI conferences rejected because we clearly did not understand how the General Problem Solver worked in path planning.

Shakey.

It can be debated whether Shakey is really the first mobile robot. There was a tortoise robot built by Grey Walter, but it was never really on the main branch of AI research. See *Behavior-Based Robots* ¹¹ for details.

Robot name trivia.

Regardless of how Shakey got its name, SRI continued the tradition with Shakey's successor called Flakey, followed by Flakey's successor, Erratic.

Strips.

The description of Strips and the robot examples are adapted from *The Handbook of Artificial Intelligence*.¹⁹

Science fiction.

While in practice, intelligent robots, such as the Mars Exploratory Rovers, do not autonomously explore new worlds without guidance from humans on earth, Vernor Vinge's classic short story, "Long Shot," describes a fully autonomous robot space ship that must explore and decide which planet would best support a human colony.

13

Navigation

Chapter Objectives:

- Name the four questions of navigation, the associated robotic functions, and the areas of artificial intelligence those functions draw upon.
- Explain the role of *spatial memory* in navigation and the four ways in which it is maintained.
- Contrast route, or *topological navigation*, with layout, or *metric*, navigation.
- Define the difference between a *natural* and *artificial landmark* and give one example of each.
- Define *gateway*, *perceptual stability*, and *perceptual distinguishability*.
- Given a description of an indoor office environment and a set of behaviors, build a relational graph representation labeling the *distinctive places* and *local control strategies* using gateways.
- Compare and contrast *relational* and *associative* methods of topological navigation.

13.1 Overview

As the term “mobile” implies, mobile robots need intelligence to be able to navigate in the open world. Mobility poses several questions, the most obvious being: *What is navigation?* Given the success of duplicating biological intelligence, another obvious question is: *How do animals navigate?* From a computing perspective, it is important to answer the questions: *Are there different types of navigation?* and *Which one is best?* As will be seen in this chapter, navigation in AI robotics connotes moving the entire robot from one location to another, sometimes called *destination planning*.

As will be seen from studies of navigation in animals, navigation is actually much broader than path planning. Destination path planning assumes that the navigational goals are known, that a map exists, and that the agent can execute the path and keep up with its location. Navigation also implies that the agent has some form of spatial memory to maintain the world model, though most animals rely on simple topological models. Biological intelligence indicates that path planning is just the tip of the iceberg of understanding navigation. Thus artificial intelligence methods for these four different aspects of navigation (mission planning, path planning, localization, and mapping) will be spread across multiple chapters. Mission planning has already been discussed in chapter 12.

This chapter and the next will concentrate on path planners. A large body of work in AI robotics has been expended creating path planning algorithms that treat the robot as a holonomic vehicle that travels between recognizable and distinct locations on a map, essentially the same abstraction and algorithms used by navigation smart phone applications to generate directions; this work is formally known as *path planning*. The directions abstract away details, such as the shape of the car, bus, or person, and operate over symbolic labels which are expected to be perceptually grounded in the world. For example, a mapper might produce directions to “turn left at Bizzell Street” regardless of whether a person is walking or driving a bus or car, and the mapper assumes that the agent will be able to detect the possibility of a turn and that the cross street is labeled with a sign that says Bizzell Street.

Smart phone mapping applications produce routes which can be executed without precisely knowing the absolute coordinates of the agent. For example “turn left at Bizzell Street” does not require a person to know the GPS location of Bizzell Street, just the topological feature of an intersection with Bizzell Street. While it is helpful to know how far the left turn is from the last navigational point, it is not strictly necessary. Thus this route can be executed without quantitative information about locations and distances in a coordinate frame. Path planners which do not produce quantitative directions are called *qualitative path planners*. Qualitative path planners are often called topological path planners or instances of *topological navigation* to emphasize that navigation depends on perceptually distinctive features in the world. Note that topological path planning is different from considering terrain features and the impact on navigation, as terrain is topographical, not topological, information.

This chapter will focus on qualitative or topological navigation, which relies on research in the search and planning fields within artificial intelligence. Search and planning are both about finding action sequences that achieve the agent’s goals. Recall that search is analogous to finding a needle in a haystack, where you know there is a definitely needle in the haystack, have good models of what a needle is and what hay looks like, and understand the haystack has boundaries. In this case, searching and planning are identical; search enables the robot to find the optimal path from the starting point to the goal. The chapter defers the topics of metric path planning and motion planning for situations pose and vehicle dynamics matter to chapter 14 and simultaneous localization and mapmaking (SLAM) and incorporating terrain sensitivities into navigation to chapter 15.

13.2 The Four Questions of Navigation

Navigation has been extensively explored in animals, and scientists see navigation as answering four questions necessary for survival. These questions are:

Where am I going? Where am I going (and why?) is addressed by *mission planning* in AI, where the agent sets goals through deliberation as discussed in the previous chapter. A planetary rover may be directed to move to a far away crater and look for a specific rock in the caldera. Roboticists generally do not include high-level mission planning as part of navigation, assuming that the robot has been directed to, or engineered for, a particular goal.

What is the best way there? This is the problem of *path planning*, and is the area of navigation which has received the most attention. As noted in the introduction, path planning methods fall into two broad categories: qualitative (or route) and quantitative (or metric).

Where have I been? This is the purview of *map making*, a rich area in AI robotics research due to

the advent of new sensors and algorithms for perceiving range. As a robot explores new environments, it may be part of its mission to map the environment. Mapping the environment requires knowing where the robot is, or localization. The category of algorithms are now known as *simultaneous localization and mapping (SLAM)*. The acronym SLAM is also an inside joke on being slammed by the significant computational challenges involved in localization and mapping.

But SLAM is not the only reason for asking: *Where have I been?* Even if the robot is operating in the same environment (e.g., delivering mail in an office building), it may improve its performance by noting changes. A new wall may be erected, furniture rearranged, paintings moved or replaced. Or, as discovered by Xavier, one of the indoor robots at the Robotics Institute at Carnegie Mellon University, certain hallways and foyers are too crowded at specific times of the day (end of class, lunchtime) and should be avoided. Applications, such as Waze, exploit the information about where other drivers have been in order to substitute for direct experience, thus improving navigational performance by planning around traffic jams or road closures.

Where am I? In order to follow a path or build a map, the robot has to know where it is; this is referred to as *localization*. If the robot is concerned only with a rough topological map, localization is fairly straight forward using relational methods such as visual homing or QualNav. Localization can be relative to a local environment (e.g., the robot is in the center of the room), in topological coordinates (e.g., in Room 311), or in absolute coordinates (e.g., latitude, longitude, altitude). Localization is more difficult if the location needs to be accurate in an absolute coordinate frame, that is, for building a map to scale. Unfortunately, as has already been seen, a robot is very poor at dead reckoning. Imagine what happens to map making when a robot travels around the perimeter of a room but cannot record its footsteps. Range sensors do not obviate the problem of localization, they actually make the problem more computationally complex even though they provide a richer set of data to be exploited in localization. Both map making and localization are covered in chapter 15.

Before the recent interest in autonomous cars, AI researchers were demonstrating reliable indoor navigation in complex environments as far back as 1998. The Rhino and Minerva Tour Guide Robots (shown in [figure 13.1](#)) provide an excellent example of how these four questions arise naturally in applications.²⁰⁶ Rhino and Minerva were designed by researchers from CMU and the University of Bonn to perform all the functions of a tour guide in a museum, including leading groups of people to exhibits upon request and answering questions. Rhino hosted tours of the Deutsches Museum in Bonn, Germany, while Minerva was used in the Smithsonian's National Museum of History in Washington, DC.



a.



b.

Figure 13.1 Two Tour Guide Robots: a.) Rhino in the Deutsches Museum in Bonn (in the center), and b.) a close up of the more emotive Minerva. (Photographs courtesy of Sebastian Thrun and Wolfram Burgard.)

The tour guide robots had to know where they were at any given time (localization) in order to answer questions about an exhibit or to provide directions to another exhibit. Minerva also could create a custom tour of requested exhibits, which require the robot to know how to get to each of the exhibits (path planning). The tour path had to be reasonably efficient, otherwise the robot might have the group pass a targeted exhibit repeatedly before stopping at it. This also meant that the robot had to remember where it had been. Rhino operated using a map of the museum, while Minerva created her own. An important aspect of the map was that it included the time needed to navigate between exhibits, not just the distance. This type of information allows a planner to factor in areas to avoid at certain times of day. For example, Xavier at CMU has learned to avoid cutting through a particular foyer when classes

are changing, since the density of people moving about slows its progress.

13.3 Spatial Memory

Spatial Memory

The answer to *What's the best way there?* depends on the representation of the world that the robot is using, again harkening back to the saying that good algorithms are enabled by good data structures. A robot's world representation and how it is maintained over time is its *spatial memory*.⁸⁴

Spatial memory is the heart of a Cartographer module, which should provide methods and data structures for processing and storing output from current sensory inputs. For example, suppose a robot is directed to "go down the hall to the third red door on the right." Even for the coordination and control of reactive behaviors, the robot needs to operationalize concepts, such as "hall," "red," "door," into features to look for with a perceptual schema. It also needs to remember how many red doors it has gone past (and not count the same door twice!). It would also be advantageous if the robot sensed a barrier or dead-end and updated its map of the world.

Spatial memory should also be organized to support methods which can extract the relevant expectations about a navigational task. Suppose a robot is directed this time to "go down the hall to the third door on the right." It could consult its spatial memory and notice that odd numbered doors are red and even numbered are yellow. By looking for "red" and "yellow," in addition to other perceptual features of a door, the robot can more reliably identify doors, either by focus of attention (the robot runs door detection only on images with red and yellow areas, not on every image) or by sensor fusion (more sources of data mean a more certain percept).

Spatial memory supports four basic functions:

ATTENTION

1. *Attention.* What features or landmarks should I look for next?

REASONING

2. *Reasoning.* Can that surface support my weight?

PATH PLANNING

3. *Path planning.* What is the best way through this space?

INFORMATION COLLECTION

4. *Information collection.* What does this place look like? Have I ever seen it before? What has changed since the last time I was here?

13.4 Types of Path Planning

Spatial memory takes two forms: *route*, or *qualitative*, and *layout*, or *metric* or *quantitative*, representations. These result in two distinct styles of robotics path planning, shown in [figure 13.2](#). Here the two styles of path planning manifest as different types of algorithms for searching through space. In general, path planning algorithms abstract the robot as a holonomic vehicle and abstract the world as a graph.

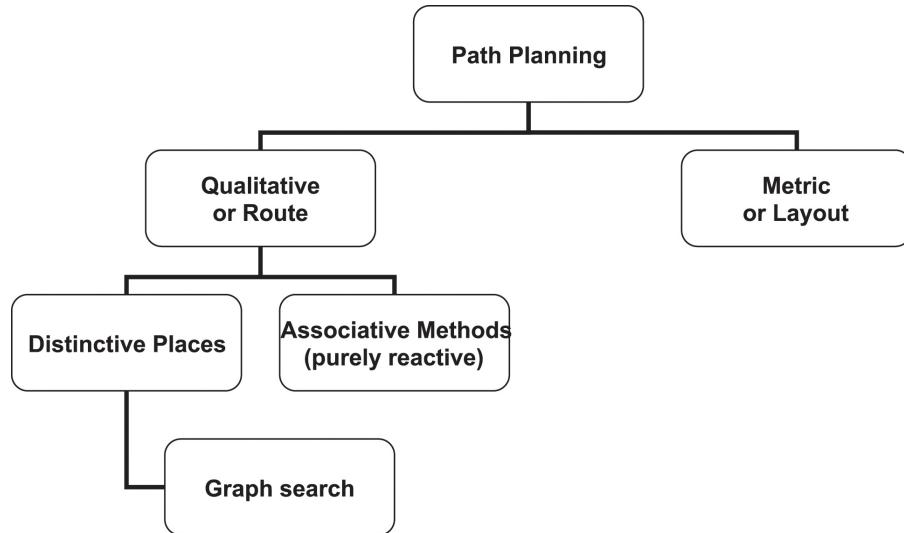


Figure 13.2 Types of path planning algorithms.

Topological path planning applies classic computer science algorithms, such as Dijkstra's single source shortest path algorithm, for optimally searching for routes through a graph. The number of possible places to turn (nodes) are fairly small given the overall space. Metric path planning applies more specialized path planning algorithms, often a variant of the A* algorithm, that can efficiently handle planning through large empty spaces where each block of empty space may be a node on a graph.

TOPOLOGICAL NAVIGATION

TOPOLOGICAL NAVIGATION

Route, topological, or qualitative representations express space in terms of the connections between landmarks. An example of a route representation is when a person gives directions propositionally (as a list): “go out of the parking lot, and turn left onto Fowler Drive. Look for the museum on the right, and turn left at next traffic light.” Notice that route navigation is perspective dependent; landmarks that are easy for a human to see may not be visible to a small robot operating close to the floor. Route representations also tend to supply orientation cues: “out of the parking lot” (versus being contained in it), “turn left,” “on the right.” These orientation cues are egocentric, in that they assume the agent is following the directions at each step. Route representations also assume that there is physical symbol grounding. Navigation using routes is commonly referred to as *topological navigation* because it is using topology to relate landmarks spatially.

METRIC NAVIGATION

Layout, metric, or quantitative representations are the opposite of route representations. When a person gives directions by drawing a map, the map is a layout representation. Layout representations are often called metric representations because most maps have some approximate scale to estimate distances. The major differences between layout and route representations are viewpoint and utility. A layout representation is essentially a bird’s-eye view of the world. It is not dependent of the perspective

of the agent; the agent is assumed to be able to translate the layout into features to be sensed. The layout is orientation- and position-independent. Layout representations can be used to generate a route representation, but this does not necessarily work the other way. (Consider how easy it is to read a map and give verbal directions to a driver, versus drawing an accurate map of a road you have been on only once.) Most maps contain extra information, such as cross streets. An agent can use this information to generate alternative routes if the desired route is blocked. Navigation using metric layouts of the world is sometimes referred to as quantitative navigation, but most often as metric navigation.

Route or topological navigation is the most common form of navigation in animals. Roboticists favor creating layout styles of spatial memory because having a metric layout, or map, can be used to generate either topological routes or metric paths.

The difference between topological and metric navigation is illustrated by the difference between navigating with an autonomous car and an unmanned aerial vehicle. It is natural to think of cars navigating by route because regardless of the exact distance between segments—say, between the four-way stop on Parkway and the traffic light at Parkway and Avenue—the car can turn only at those externally recognizable locations. In contrast, it is natural to think of a UAV navigating based solely on GPS waypoints.

In practice, a car uses a combination of topological and metric navigation to overcome uncertainty and to cue perception. For example, knowing that the distance between the four-way stop on Parkway and the traffic light at Parkway and Avenue is 0.8km is helpful. If the car goes more than 1km, it is clear that it passed the traffic light, while, if it finds a traffic light at 0.75km, that is most likely the correct traffic light, and there was error in measuring the distance. Likewise, the perceptual system may not need to devote CPU cycles to looking for the traffic light until it has traveled 0.5km beyond the four-way stop.

13.5 Landmarks and Gateways

LANDMARK

Topological navigation depends on the presence of landmarks. A *landmark* is one or more perceptually distinctive feature of interest on an object or locale. Note that a landmark is not necessarily a single, self-contained object like “red door.” A landmark can be a grouping of objects; for example, to many people the phrase “McDonald’s” evokes a tall sign, a bright building of a certain shape, and a parking lot with a lot of activity. Another outdoor landmark might be a “stand of aspen trees.”

Landmarks are used in most aspects of navigation. If a robot finds a landmark in the world and that landmark appears on a map, then the robot is localized with respect to the map. If the robot plans a path consisting of segments, landmarks are needed so the robot can tell when it has completed a segment and another should begin. If a robot finds new landmarks, they can be added to its spatial memory, creating or extending a map.

GATEWAY

Dave Kortenkamp popularized a particularly interesting special case of landmarks called a *gateway*.¹⁰⁷ A gateway is an opportunity for a robot to change its overall direction of navigation. For example, an intersection of two hallways is a gateway; the robot can choose to go straight or turn left or right. Because gateways are navigational opportunities, recognizing gateways is critical for

localization, path planning, and map making.

ARTIFICIAL LANDMARKS

NATURAL LANDMARKS

Landmarks may be either *artificial* or *natural*. The terms “artificial” and “natural” should not be confused with “man-made” and “organic.” An artificial landmark is a set of features added to an existing object or locale in order to support either recognition of the landmark or some other perceptual activity. An interstate highway exit sign is an example of an artificial landmark. It is put there for the purpose of being easy to see (retro-reflective), and the white-on-green font is sized for optimal visibility (perceptual activity is reading the sign). A natural landmark is a configuration of existing features selected for recognition which were not expressly designed for the perceptual activity. If someone gives directions to her house, such as “take the second right after McDonald’s,” McDonald’s is being used as an orientation cue for navigation to the house. Clearly, McDonald’s was not built for the purpose of being a navigational cue to a private home. The fact that it was used for another purpose means it is a natural landmark.

CRITERIA FOR LANDMARKS

Regardless of whether a landmark is artificial or natural, it must satisfy three criteria:

1. *Be readily recognizable.* If the robot cannot find the landmark, it is not useful.
2. *Support the task dependent activity.* If the task dependent activity is simply an orientation cue (“take the second right after the McDonald’s”), then being recognized is enough. Suppose a landmark is intended to provide position estimation to guide docking of the space shuttle to a space station. In that case, the landmark should make it easy to extract the relative distance to the point of contact.
3. *Be perceivable from many different viewpoints.* If the landmark is widely visible, the robot may never find it.

[Figure 13.3](#) shows artificial landmarks constructed for use in the 1992 AAAI Mobile Robot Competition.⁵⁶ Each robot was supposed to follow a route between any sequence of waypoints in an arena. The teams were allowed to mark the waypoints with artificial landmarks. Each waypoint was readily recognizable by the checkerboard pattern. The task dependent activity of going to the correct waypoint was facilitated by the cylindrical barcodes which are unique for each station. Notice that the use of a cylinder guaranteed that landmarks would be perceivable from any viewpoint in the arena.



Figure 13.3 Artificial landmarks used in the 1992 AAAI Mobile Robot Competition. (Photograph courtesy of AAAI.)

When the RoboCup soccer small league started, considerable effort was put into deciding on landmarks to aid with navigation. The goals and corners of the ring around the playing field were festooned with bright colors that were easy to extract with computer vision algorithms. The relative position of the colors indicated the location (green on top meant on the defender side of the field, green on bottom meant the opponent side).

A good landmark has many other desirable characteristics. It should be passive, that is, not emit energy, in order to be available despite a power failure. It should be perceivable over the entire range where the robot might need to see it. It should have distinctive features, and, if possible, unique features. Distinctive features are those which are locally unique; they appear only as part of the

landmark from every viewpoint of the robot in that region of the world (e.g., there is only one McDonald's on Busch Boulevard). If the feature occurs only once in the entire region of operations (e.g., there is only one McDonald's in Tampa), then the feature would be considered globally unique. In addition to being perceivable for recognition purposes, it must be perceivable for the task. If the robot needs to position itself to within 0.5 meters of the landmark, then the landmark must be designed to support extraction of that degree of accuracy.

13.6 Relational Methods

Relational methods represent the world as a graph or network of nodes and edges (see figure 13.4). Nodes represent gateways, landmarks, or goals. Edges represent a navigable path between two nodes, in effect, representing that two nodes have a spatial relationship. Additional information may be attached to edges, such as direction (north, south, east, west), approximate distance, terrain type, or the behaviors needed to navigate that path. Paths can be computed between two points using standard graph algorithms, such as Dijkstra's single source shortest path algorithm. (See any algorithm textbook for details.)

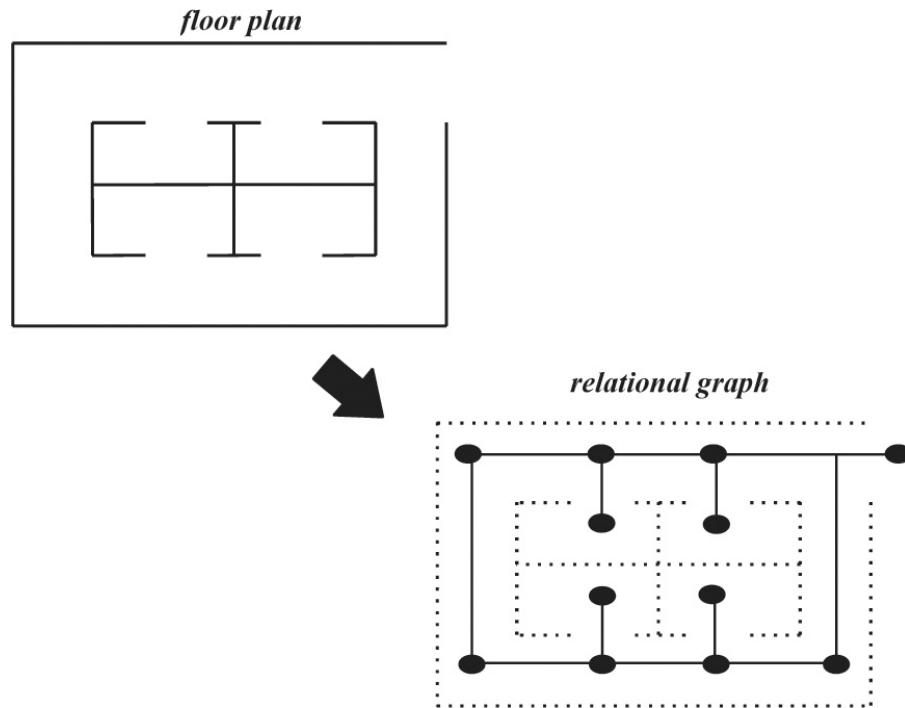


Figure 13.4 Representation of a floor plan as a relational graph.

One of the earliest investigations of relational graphs for navigation was conducted by Smith and Cheeseman.¹⁹⁷ They represented the world as a relational graph, where the edges represented the direction and distance between nodes and simulated what would happen if a robot used dead-reckoning

to navigate. As would be expected from the section on proprioception in chapter 10, they found that the error would continually increase, and soon the robot would be unable to reach any of the nodes ([figure 13.5](#)).

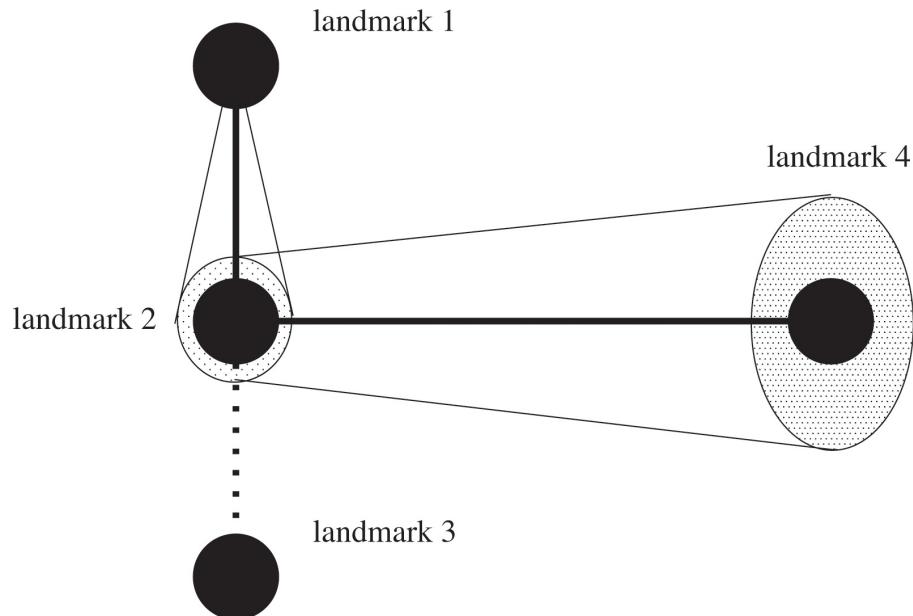


Figure 13.5 Propagation of error in a relational graph.

13.6.1 Distinctive Places

DISTINCTIVE PLACE

Kuipers and Byun tied relational graphs to sensing in their seminal work with distinctive places.¹⁰⁹ A *distinctive place* is a landmark that the robot could detect from a nearby region called a neighborhood. Their work was motivated by research in cognitive science indicating that spatial representation in the animal kingdom forms a multi-level hierarchy ([figure 13.6](#)). (More recent studies suggest this hierarchy is not as clearly partitioned as previously thought.) The lowest level, or most primitive way of representing space, was by identifying landmarks (doors, hallways) and having the procedural knowledge to travel between them (follow-hall, move-thru-door). The next layer up was topological. It represented the landmarks and procedural knowledge in a relational graph which supported planning and reasoning. The uppermost level was metric, where the agent had learned the distances and orientation between the landmarks and could place them in a fixed coordinate system. Higher layers represented increasing intelligence.

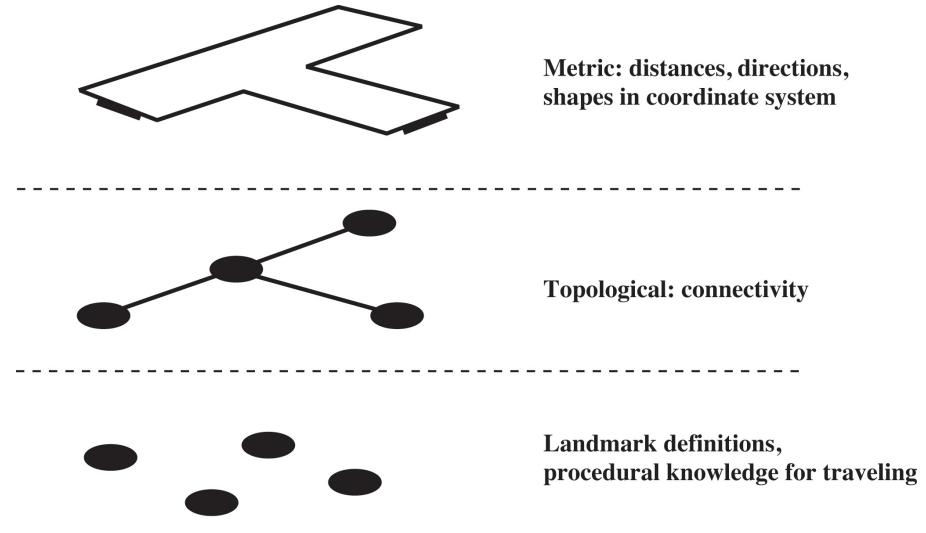


Figure 13.6 Multi-level spatial hierarchy, after Byun and Kuipers.¹⁰⁹

Kuipers and Byun's topological representation is of particular interest. Each node represents a distinctive place. Once in the neighborhood, the robot can position itself, using sensor readings, in a known spot relative to the landmark. One example of a distinctive place was a corner (figure 13.7). (Kuipers and Byun worked in simulation; a corner as a distinctive place did not turn out to be realistic with sonars.) The idea of a neighborhood was that the robot would move around in the neighborhood until it achieved a specified relative location to the landmark; for the corner example, the robot might position itself 1.0 meter from each wall. Then the robot would be localized on the map.

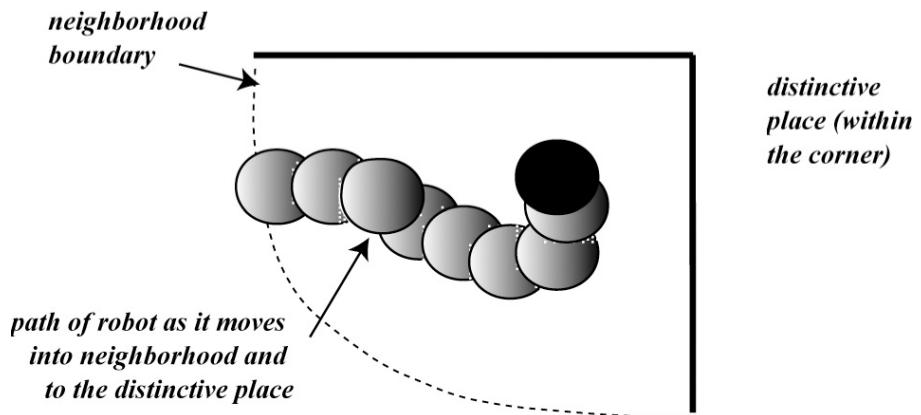


Figure 13.7 A robot reacting to a corner and moving to a distinctive place within the neighborhood.

LOCAL CONTROL STRATEGY

HILL-CLIMBING ALGORITHM

An arc or edge in the relational graph was called a *local control strategy*, or *lcs*. The local control strategy is the procedure for getting from the current node to the next node. When the robot senses the landmark, it fills in values for a set of features. The robot uses a *hill-climbing algorithm*. The hill-climbing algorithm directs the robot around in the neighborhood until a measurement function (e.g., how far away the walls are) indicates when the robot is at a position where the feature values are maximized (e.g., both are 1.0 meter away). The point where the feature values are maximized is the distinctive place. The hill-climbing algorithm is a very simple approach. The idea is that for most hills, if you always choose the next step that is the highest (you cannot look ahead), then you will get to the top quickly. Therefore, the robot always moves in the direction which appears to cause the most positive change in the measurement function.

Although researchers discovered reactive behaviors and distinctive places independently of each other, reactive behaviors map nicely onto distinctive places and local control strategies as shown in figure 13.8. Consider a robot navigating down a wide hall to a dead-end. The local control strategy is a behavior, such as `follow-hall`, which operates in conjunction with a releaser, `look-for-corner`. The releaser is an exteroceptive cue. When it is triggered, the robot is in the neighborhood of the distinctive place, and the released behavior, `hillclimb-to-corner`, directs the robot to within 1.0 meter from each wall.

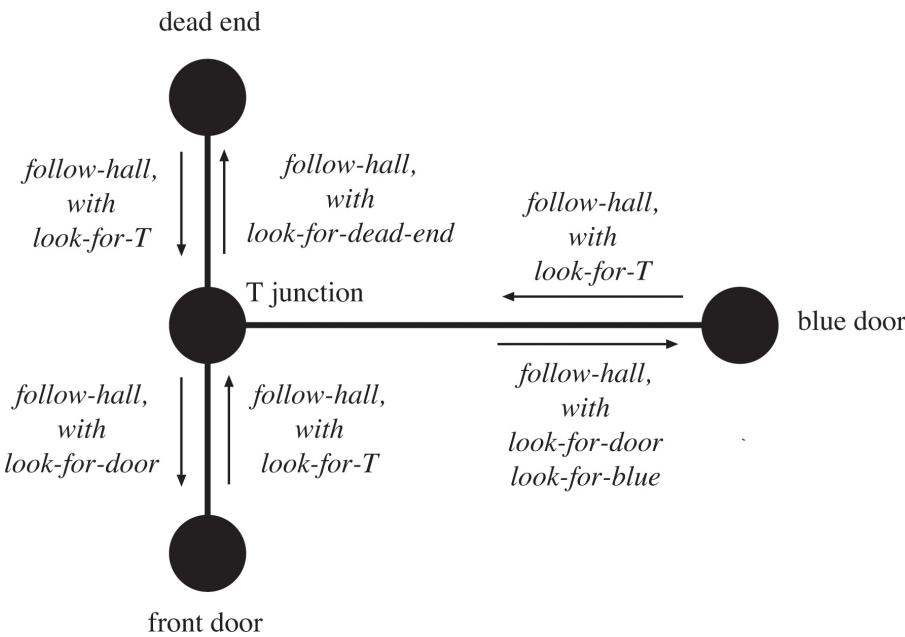


Figure 13.8 Behaviors serving as local control strategies and releasers serving as means of signaling the entrance to a neighborhood.

13.6.2 Advantages and Disadvantages

The distinctive place concept eliminates any navigational errors at each node. The robot may drift off course because the hall is wide, but as soon as it reaches the neighborhood, it self-corrects and localizes

itself. Kuipers and Byun were able to show in simulation how a robot with dead reckoning errors could use multiple trips between nodes to build up a reasonable metric map, since most of the errors would average out. Another attractive aspect of the distinctive place approach is that it supports discovery of new landmarks as the robot explores an unknown environment. As long as the robot could find something distinctive that it could reliably localize itself to, that place could be put on a topological map. Then as the robot repeatedly moved to that place, the robot could construct a metric map.

Returning to the discussion of landmarks, it should be noticed that a landmark must be unique to a node pair. There cannot be any corners in the real world that are not on the graph between the nodes or else the robot will localize itself incorrectly.

The distinctive place approach, as originally formulated, encountered some problems when behavior-based robotocists began to apply it to real robots. One of the most challenging problems was perception. Good distinctive places are hard to come by; configurations that seemed useful to humans, like corners, proved difficult to sense reliably and localize against. Features that were easy to sense, such as corners, doors, and horizontal lines, often were too numerous in the world, and so were not locally unique. Another challenge was learning the local control strategy. As the robot explored an unknown environment, it was easy to imagine that it could find distinctive places. But how did it learn the appropriate local control strategy? In an indoor environment, the robot might resort to always using wall-following, even though other behaviors would be better. How would the robot ever try something different? Another open issue is the problem of indistinguishable locations. The issue of indistinguishable locations has been tackled to some degree by work with probabilistic methods, which will be covered in chapter 15.

13.7 Associative Methods

ASSOCIATIVE METHODS

Associative methods for topological navigation essentially create a behavior which converts sensor observations into the direction to go to reach a particular landmark. Associative methods are infrequently used in robot systems, but it is important to be aware of them. There are basic approaches: *visual homing*,^{161;72} where the robot recognizes visual patterns associated with locations and routes (the approach is similar to the way bees navigate), and *QualNav*,¹¹⁵ where the robot infers its location based on how the relative positions of landmarks change (this approach is similar to navigating by constellations).

The underlying assumption is that a location or landmark of interest for navigation usually has two attributes:

PERCEPTUAL STABILITY

1. *perceptual stability*: views of the location that are close together should look similar

PERCEPTUAL DISTINGUISHABILITY

2. *perceptual distinguishability*: views far away should look different.

These principles are implicit in the idea of a neighborhood around a distinctive place. If the robot is in the neighborhood, the views of the landmark look similar. The main difference between distinctive

places and associative methods is that associative methods use very coarse computer vision. Associative methods are very good for retracing steps or routes.

13.8 Case Study of Topological Navigation with a Hybrid Architecture

This section presents a case study of topological navigation using the SFX architecture in the 1994 AAAI Mobile Robot Competition by a team of undergraduates from the Colorado School of Mines. The 1994 competition had an office navigation event.¹⁸ Each robot was placed in a random room and then had to navigate out of the room and go to another room within 15 minutes (see [figure 13.9](#)). Entrants were given a topological map but were not allowed to measure the layout of the rooms and halls. This case study shows how the topological map was entered, the activities of the Cartographer, how scripts (discussed in chapter 19) were used to simplify behavioral management, and the lessons learned.

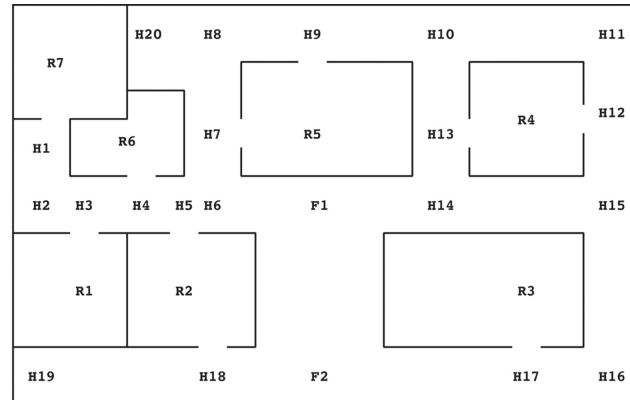


Figure 13.9 A student testing an unidentified entry in the competition arena at the 1994 AAAI Mobile Robot Competition “Office Delivery” event. (Photograph courtesy of AAAI.)

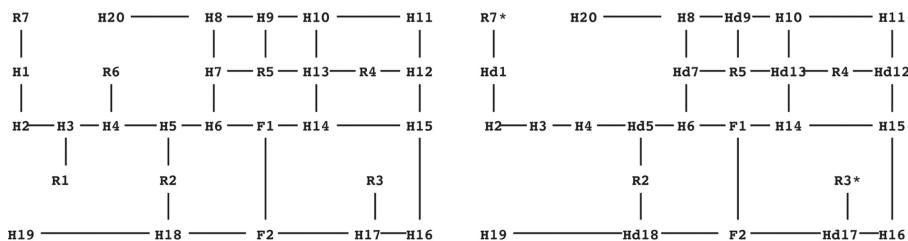
13.8.1 Topological Path Planning

The topological map was entered as an ASCII file in Backus-Naur Form. A sample map is shown in figure 13.10. The input map consists of three node types: room (R), hall (H), and foyer (F). The world is assumed to be orthogonal. Since edges between nodes can exist in only four directions, it is convenient to refer to them as north (N), south (S), east (E), and west (W), where N is set arbitrarily on the map. The robot is given its starting node but, as an extra challenge, the robot is not given the direction it is initially facing relative to the map. The topological map is structurally correct, but it does not necessarily represent if a corridor or door is blocked. Such blockages may occur, or be moved, at any

time. An additional assumption is that the outside of each door is marked with a landmark such as a room number or room name.



a.



b.

c.

Figure 13.10 a.) Metric map of an office layout, b.) graphical topological representation, and c.) refined graph for traveling from R3 to R7.

The Cartographer in SFX is responsible for constructing the route. It takes, as input, a gateway type of topological map and a start node and goal node and produces a list of nodes representing the best path between the start and goal. The cartographer operates in two steps: preprocessing the map to support path planning and path planning itself. The preprocessing step begins by building a database of the nodes in the input map, reclassifying the corridor nodes which represent a hall-to-door connection as Hd.

Once the start and goal nodes are known, the Cartographer eliminates extraneous gateways. A Hd node may be connected to a room which is not visitable, that is, it is neither the goal room, the start room, or a room with more than one door. If a room is not visitable, then both the R and Hd node entries are eliminated from the database. A sample input graphical topological representation for a metric map is shown in [figure 13.10](#). If R3 was selected as the start node and R7, the goal, the refined graphical representation is shown in [figure 13.10](#). Note that Hd3-R1 and Hd4-R6 were eliminated because they were not associated with the start or goal rooms and could not be used as a shortcut since they each had only one entrance. Gateway nodes, such as H10, remain in the path because they may be useful if a

blocked path occurs. For example, if the robot is going down the hallway from H11 to H8 and the path is blocked, the robot will need to return to a known location in order to reorient itself and replan. However, if the H10 is eliminated, then the robot must return to the very beginning of the corridor because it does not know where it is with respect to H10. To solve this dilemma of traveling the same part of the corridor possibly three times, the cartographer maintains nodes which represent possible alternative strategies. It should be noted that different types of gates, tasks, and robot capabilities will lead to different preprocessing strategies.

An optimal path is computed using Dijkstra's single source shortest path algorithm. The algorithm generates the shortest path by considering the costs between each pair of connected nodes; the costs are expressed as the length or weight of an edge in the graph. Since the topological representation is not metric, connections between the nodes reflect *preferences* in constructing routes. These preferences are expressed as edge weights. In this implementation, navigating through foyers was considerably more computationally expensive and unreliable; therefore the edge-weighting for any path subsegment originating from a foyer was set at 3. The edge weight for going from a room to another room was set at 2 to discourage the robot from finding solutions which were "impolite" (e.g., cutting through people's offices). All other connections were set to 1.

TRANSITION TABLE

The Task Manager in SFX uses the path computed by the cartographer to select the appropriate *abstract navigation behavior* (chapter 19) for traveling between a particular pair of nodes. It maintains a pointer to the current node and to the next intended node from the path. The current- and next-node type determine the appropriate behavior according to the *transition table*, shown below:

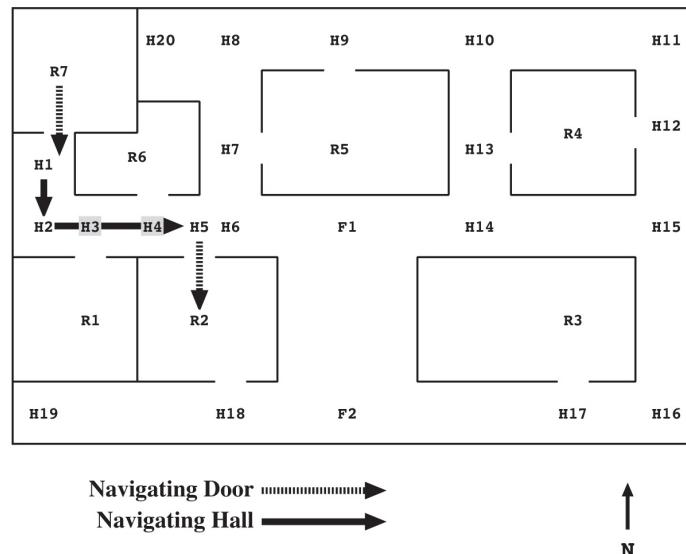
From	To			
	H	F	R	Hd
H	Navigate-hall	Navigate-hall	Undefined	Navigate-hall
F	Navigate-hall	Navigate-foyer	Navigate-door	Navigate-hall
R	Undefined	Navigate-door	Navigate-door	Navigate-door
Hd	Navigate-hall	Navigate-hall	Navigate-door	Navigate-hall

The transition table shows that not all combinations of nodes are permitted; by definition, the robot cannot move from a hall node H to a room node R without going through a Hd node. Also, the table is not necessarily symmetrical. In the case of rooms, navigate-door must be employed either to enter or exit, but the case of moving to a foyer will use different strategies depending on whether the robot is traversing a hall or a foyer. The ANB itself uses the information in the database entries for the nodes as parameters for instantiating the script to the current waypoint pair.

One novel aspect of this implementation is the way the Task Manager handles a blocked path; it does not reverse the currently instantiated abstract navigation behavior (ANB). If the obstacle avoidance behavior posts to the whiteboard structure that a BLOCKED condition has occurred, the Task Manager terminates the currently active abstract navigation behavior. Because the robot is between nodes, the Task Manager directs the robot to return to the current node. But it triggers a simple move-to-goal behavior, which allows the robot to reorient itself more quickly than it would be able to if it had to reinstantiate the abstract navigation behavior with new parameters.

Once the robot has returned approximately to the last known location, the Task Manager requests a new path from the Cartographer. The Cartographer removes the connection between the current and intended nodes and then recomputes a new path with the current node as the start. Once the new path is completed, the Task Manager resumes control.

[Figure 13.11](#) demonstrates the robot moving through the map handed out at the actual AAAI Mobile Robot Competition. The start node is R7 and the goal is R2. The cartographer computes the path as R7 - H1 - H2 - H5 - R2. Note that H3 and H4 are not considered relevant gateways for this task and so do not appear in the path.



```

R7 -> R2
R7 - H1 - H2 - H5 - R2
Moving from R7 to H1, going SOUTH
In navigating door behavior
    ultra looking for door towards the: SOUTH
    MOVE_AHEAD MOTOR ACTIVE
    Found door - Initialization terminated
    MOVE_THROUGH DOOR MOTOR ACTIVE
Moved through door - Nominal Behavior terminated

Moving from H1 to H2, going SOUTH
In navigating hall behavior
    turning towards the: SOUTH
    Turned towards hall - Initialization terminated
    looking for hall towards the: EAST
    HALL FOLLOW MOTOR ACTIVE
Found hall - Nominal Behavior terminated

Moving from H2 to H5, going EAST
In navigating hall behavior
    turning towards the: EAST
    Turned towards hall - Initialization terminated
    vision looking for door relative: 90 (right side)
    HALL FOLLOW MOTOR ACTIVE
Found door (vision) - Nominal Behavior terminated

Moving from H5 to R2, going SOUTH
In navigating door behavior
    ultra looking for door towards the: SOUTH
    following wall on left (right ground truth)
    WALL FOLLOW MOTOR ACTIVE
    Found door - Initialization terminated
    MOVE_THROUGH DOOR MOTOR ACTIVE
Moved through door - Nominal Behavior terminated
Goal location reached!

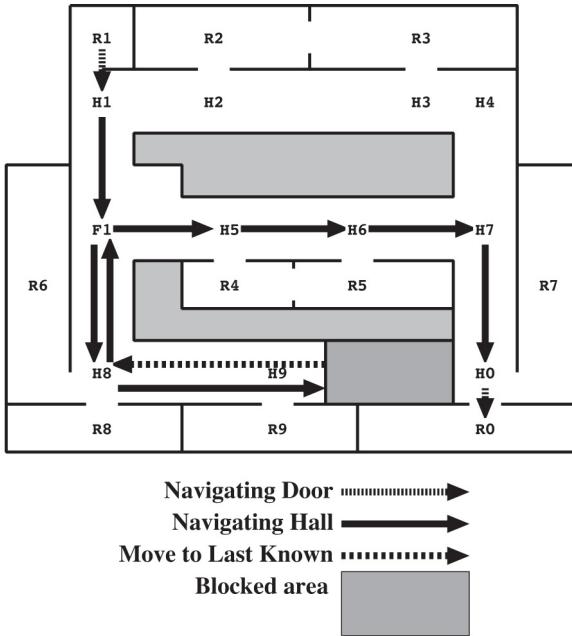
```

[Figure 13.11](#) Scenario for moving from R7 to R2. Shaded gateways are extraneous and discarded by the planner.

As shown by the output, the Cartographer computes the path shown in the first line. The second line shows that the current navigational task is to move from R7 to H1. The Task Manager selects `navigate-door`, and the robot begins by looking for the door since the user did not specify where it was. By examining the database, the door is known to be to the south. Therefore, the behavior initiates the `move-ahead` behavior to the south. When the door is found, the initialization phase of the abstract behavior is over and the nominal activity of moving through the door is triggered. Once the robot is through the door, the nominal behavior is terminated, terminating the entire abstract behavior as well.

The next navigational task is to move from H1 to H2, again in the south direction. The task manager selects `navigate-hall`. The task of moving from H2 to H5 is interesting because it shows that the termination condition for hall following is different from traveling from H1 to H2. Since H5 is a gateway to a room of interest, a different perceptual process is used to identify the room visually. Once the door has been identified visually, the robot is considered to be at H5. The H5-R2 connection instantiates `navigate-door`. However, in this case, the ultrasonics have not yet identified the door opening, and so the initialization phase consists of wall following until the door opening is recognized. Then the nominal behavior is activated, and the robot moves into the room, successfully completing the task.

The second simulation shown in [figure 13.12](#) uses the sample topological map provided prior to the competition. The intended path was R1-H1-F1- H8-H0-R0, but H8-H0 was blocked. As shown in the figure, the robot detects that the path is blocked and uses `move-to-goal` to return to H8. The Cartographer updates the topological map, and computes a new route and the robot resumes execution, in this case, H8-F1-H5-H6-H7-H0-R0. The output of the script below shows the robot confirming the nodes H5 and H6 even though the plan did not have the robot enter the associated rooms. These nodes were kept in the path because, if another blocked hall was encountered, the robot would then be able to return either to R5 or R6 and attempt to use a route through the door in between.



```

Moving from H8 to H0, going EAST
In navigating hall behavior
    turning towards the: EAST
    Turned towards hall - Initialization terminated
    HALL FOLLOW MOTOR ACTIVE
User stopped or BLOCKED - behavior terminated
    In last known location loop
    MOVE TO GOAL MOTOR ACTIVE

Moving from H8 to F1, going NORTH
In navigating hall behavior
    turning towards the: NORTH
    Turned towards hall - Initialization terminated
    HALL FOLLOW MOTOR ACTIVE
    will follow wall on right (left ground truth) to stay in foyer
    Found foyer - Nominal Behavior terminated

Moving from F1 to H5, going EAST
In navigating hall behavior
    following wall on left (right ground truth)
    trying to exit foyer
    looking for hall towards the: EAST
    WALL FOLLOW MOTOR ACTIVE
    Found hall - Initialization terminated
    HALL FOLLOW MOTOR ACTIVE
    Found door (vision) - Nominal Behavior terminated

```

Figure 13.12 Scenario for moving from R1 to R0 with blocked path.

13.8.2 Navigation Scripts

The path planning and execution components are clearly deliberative. The Cartographer is maintaining a map in the form of a graph and is monitoring progress. The transition table takes the role of a high-level sequencer. Scripts are used to specify and carry out the implied details of the plan in a modular and reusable way.

The implementation consisted of three scripts for navigating doors, halls, and foyers. The pseudocode for `navigate-door` is shown below in C++ style and takes advantage of the indexing properties of the switch statement:

```

switch(door)
case door-not-found:
    //initialization phase
    //follow wall until find door
    if wall is found
        wallfollow to door
    else
        move-ahead to find a wall
case door-found:
    //nominal activity phase
    move-thru-door(door-location)

```

The nominal behavior, move-thru-door, is self-terminating, so there is no separate termination condition. The perception of the door is the key determinant in what the robot does.

The navigate-hall behavior is used for traveling between corridors, foyers, and corridors, and from corridors to corridor/door nodes. The behavior has two different starting conditions. One condition is that the robot is in a foyer and detects a hall. Since the hall may be somewhat ahead and the hallfollow behavior assumes the robot is in the hall, the script has the robot employ wallfollow to find the hall and then begin following the hall. The task manager uses the directional information stored in the database to determine which wall of the foyer to follow. In the other condition, the robot is already in a hall. Since the robot is not guaranteed to be facing the centerline of the hall (ANBs have no knowledge of what was done prior to their instantiation except via global variables), the sub-script turns it to line itself up with the desired hall.

```

switch(hall)
case not-facing-hall:
    //initialization phase
    if starting in a FOYER
        if hall-not-found
            wallfollow until find the hall
    else
        if not facing hall
            turn to face hall
    else starting in a HALL
        if not facing hall
            turn to face hall
case facing-hall:
    //nominal activity phase
    hallfollow until next gateway

```

The navigate-hall ANB terminates when the next expected gateway (the next node in the path) is found. There are three behaviors which look for gateways. hallwatch looks for the ultrasonic signature of a hall in the expected direction; foyerwatch similarly looks for foyers, and confirm-door uses vision to detect the landmark associated with the room. These behaviors run concurrently with the nominal behaviors.

The navigate-foyer ANB is used to move the robot between two foyers. It assumes that two

connected foyer nodes are a convenient representation of a single large foyer with entries from different directions (i.e., multiple gateways into the foyer). The script moves the robot n feet into the first foyer in the direction of the second foyer. This gets the robot away from potentially confusing ultrasonic signatures. Then the task manager determines which side of the foyer to wall-follow until the next expected gateway is detected. There is no case statement in the pseudocode since the sequence of activities is fixed.

```
//step 1  
move-to-goal(n, dir) in direction of next foyer  
//step 2  
wallfollow until next gateway is detected
```

13.8.3 Lessons Learned

The CSM team did not place at the AAAI competition; the robot suffered a series of hardware failures, causing both the power supply and the sonars to fail. The robot was eventually fixed, and the software worked fine. The tests before and after the competition offered several practical lessons.

First, it is critical to build the abstract navigation behaviors out of robust primitives. The biggest problem with the implementation has not been with the scripts per se but rather with the quality of the primitive behaviors. If the robot cannot reliably detect an open doorway while it is following a wall around a room, it will never exit no matter what the script says. Second, contention for sensing resources is an emerging issue for robotic control schemes. The video camera on the robot did not have a pan mechanism; essentially, Clementine's directional control is the camera effector. In one instantiation of the navigate-hall behavior between an H node and a Hd node, the nominal behavior of hallfollow frequently pointed Clementine and her camera away from the anticipated location of the door, interfering with the confirm-door perceptual behavior used to terminate navigate-hall. Even at slow speeds, Clementine frequently missed the door. One alternative is to let the two active behaviors take turns controlling the robot. This leads to a move, stop, sense scenario which slowed the robot's progress and still allowed the robot to roll past the door.

It should be noted that the representation and path planning algorithm employed support the addition of metric information. The metric distance between every node-pair that has been visited could be stored in the database. The node-class stored a reference index to the nodes connected to the N, E, S, W; the distance could be attached to these indices as well. One problem with simply attaching the metric distance is deciding what distance value to use. If wallfollow takes the robot around the perimeter of a foyer in order to exit, the distance traveled will not reflect the straight line distance between nodes, nor will the measure be bidirectional; going N-S through the foyer could be significantly longer than S-N if the foyer is asymmetric and the robot follows different walls each time. On the other hand, if the width of the foyer were known, the robot might possibly be able to use dead reckoning to move directly across it to the desired exit. The impact of obstacles lengthening the distance traveled can also be a problem, especially if the obstacles are people moving down a hall. A short hall can take a long time to navigate if it is crowded with people during one visit, but empty during another. Another difficulty with adding metric distances is determining how to use distances effectively. Metric distances may not be known for all pairs of nodes, making it difficult to apply Dijkstra's algorithm. Likewise, distance may be only one factor in selecting a route; our current implementation prefers going through halls versus

taking short cuts between rooms. Utility theory is one mechanism for quantifying the impact of these competing concerns.

13.9 Discussion of Opportunities for AI

AI robotics approaches to navigation have been focused on path planning and simultaneous localization and mapping. These topics rely heavily on the AI areas of knowledge representation, search, and planning. Topological methods, especially associative methods, are often a subset of computer vision research. Topological methods are essentially behavioral methods and suffer from the same issue of lack of overall robustness; if the method fails, it is difficult to detect and apply problem solving techniques. The other areas of AI are typically not incorporated into topological navigation implementations.

The key knowledge representation used in topological path planning is some form of spatial memory. The Reactive Paradigm inspires the topological “less is more” approach which minimizes spatial memory and the semantic labels applied to gateways and landmarks. The amount of spatial memory an agent needs to navigate is a gray area and depends on many factors. How accurately and efficiently does the robot have to navigate? Is time critical, or can it take a slightly suboptimal route? Navigational tasks which require optimality tend to require more dense and complex world representations. What are the characteristics of the environment? Are there landmarks to provide orientation cues? Are distances between landmarks known accurately? What are the sources of information about that environment that specify terrains, surface properties, obstacles, and so forth? What are the properties of the available sensors in that environment?

Topological navigation embeds generating the path with selection and implementation of the procedures to execute navigation along the path. In relational methods, the world model consists of distinctive places connected by local control strategies, which essentially are behaviors. Historically it has not required sophisticated search algorithms to find paths as the network of distinctive places is sufficiently sparse that it can use standard graph algorithms. In associative methods, the world model consists of viewframes, which have an associated direction to move in, or combinations of landmarks, from which the associated direction can be remembered for previously encountered paths or explorations of the environment or inferred for new goals.

Topological navigation assumes that perception, especially computer vision, is sufficient to identify and learn landmarks and gateways, although the methods try to skirt the symbol grounding problem. Distinctive places “recognize” a place but that recognition is closer to an affordance than actual recognition, returning to Neisser’s two types of perception. Associative methods are interesting because they were computer vision oriented. In all cases, the underlying theme was that the agent would build a world model by exploring the environment and learning what was unique (e.g., distinctive places, landmarks, viewpoints) and the behaviors (e.g., local control strategies, direction of movement). In practice, the practical challenges of reliable perception and behavioral execution tended to overwhelm topological implementations and thus they started with an a priori map.

While topological methods use planning, there is less emphasis on the problem solving aspect of planning. One reason may be that without metric information, for example, how long it takes to get from one place to another, it is difficult to monitor progress. In the AAAI competitions and other

events set indoors, the environment has been rich enough in gateways for the robot to detect quickly that it has gone too far and missed a cue. On rural highways, a robot could drive for long distances before realizing it was not on the right path if it did not have GPS or another form of metric localization.

13.10 Summary

This chapter has attempted to address: *What is navigation?* in terms of artificial intelligence. Navigation can be divided into four functions or questions, each of which requires intelligence. One is: *Where am I going?* addressed by *mission planning*. A second is: *What is the best way there?* or the problem of *path planning*, which is typically synonymous with navigation. *Where have I been?* and *Where am I?* form the basis for the field of exploration and *simultaneous localization and mapping*.

Returning to the questions that motivated this chapter, *How do animals navigate?* can be broadly answered as: They navigate topologically. It should be noted, though, that a few animals use dead reckoning to retrace their path home. However, navigation is an area where people radically diverge from animals. We engineer our environment with landmarks, such as signs, and additional information, such as mile markers, and use GPS to help simplify navigation. The chapter partially answers: *Are there different types of navigation?* There are two broad classes of navigation, topological and metric, but, as seen by the functional questions of navigation, navigation can also be thought of as having four foci. One focus is on path planning, which is based on a priori maps or spatial memory, and another significant focus of work is on exploration and map making to generate the maps. The mission planning aspect of navigation, or setting goals of where to go, described in the previous chapter, has not been a major focus of work in robotics. The assumption has been that a person (or some other agent) will set the goals, either programmed in, or communicated by, natural language or by pointing (deictic gestures). The answer to *Which type of navigation is best?* depends on the application. Topological navigation is well suited for situations where reactive styles of spatial memory and behavioral implementations are sufficient.

The chapter delved into methods for topological navigation. Topological navigation, which is also called *route* or *qualitative navigation*, is often viewed as being more simple and natural for a behavior-based robot. Certainly people frequently give other people routes as directives; therefore it seems natural to expect a robot to be able to parse commands such as “go down the hall, turn to the left at the dead end, and enter the second room on the right.” Even without a map of where everything is, there is enough information for navigation as long as the robot knows what a “hall,” “dead-end,” and “room” are.

Topological navigation in robots relies on landmarks. Landmarks simplify the *Where am I?* problem by providing orientation cues. Gateways are a special case of landmarks which reflect a potential for the robot to change directions (turn down a different road or hall, enter a room, etc.). There are two categories of qualitative navigation methods: *relational* and *associative*. Relational methods relate *distinctive places* (nodes) to each other by the *local control strategies* (edges), or behaviors, needed to travel between nodes forming a graph. The robot can use the graph to plan a path using techniques such as the single source shortest path algorithm. It executes the path by employing the behavior associated with the edge it is traversing. When it sees the landmark of the location, it is in the neighborhood, and

then it can use another behavior, such as hill-climbing, to localize itself relative to the landmark. Associative methods relate perception to location either by remembering the visual pattern of changes that allow a robot to home in on a location or by inferring the location from the arrangement of landmarks. Associative methods are better than other route methods for retracing known paths.

To make a bad pun, topological navigation only scratches the surface of navigation and artificial intelligence. The next chapter will address metric path planning and motion planning, followed by simultaneous map making and localization and finally how terrain and clutter impact navigation in chapter 15.

13.11 Exercises

Exercise 13.1

What is spatial memory, and why is it important for navigation?

Exercise 13.2

What are the four ways in which an agent maintains spatial memory?

Exercise 13.3

Define:

- a. gateway
- b. perceptual stability
- c. perceptual distinguishability.

Exercise 13.4

List the four questions associated with navigation. For each question, describe the associated robotic functions and the relevant areas of AI.

Exercise 13.5

Describe the difference between route and metric representations of space.

Exercise 13.6

Define the difference between natural and artificial landmarks and give one example of each.

Exercise 13.7

Define a gateway. What gateways would be in a museum? What sensors would be needed to detect them reliably?

Exercise 13.8

Define a gateway. What gateways would be on a road? What sensors would be needed to detect them reliably?

Exercise 13.9

Build a relational graph representation labeling the distinctive places and local control strategies using gateways for a floor of a building on campus.

Exercise 13.10

Consider the spatial hierarchy of Kuipers and Byun. Do these three levels fit naturally within a Hybrid architecture? How would the levels be implemented in a state-hierarchy style? A model-oriented style?

Exercise 13.11

Did Minerva use topological navigation? What did it use for landmarks?

Exercise 13.12

Why would associative methods be very good for retracing steps or routes?

Exercise 13.13

[Advanced Reading]

Read the scientific papers at the Minerva website. Describe:

- a. The path planner used. Evaluate it using the criteria presented in this overview.
- b. Discuss the impact of sensor uncertainty on Minerva's navigation and describe how it was addressed.
- c. List which of the four functions of spatial memory were used.

[Programming]

Exercise 13.14

Look up the following algorithms and describe how they work. Can they be used interchangeably?

- a. Dijkstra's single source shortest path
- b. hill-climbing.

[Programming]

Exercise 13.15

Design a visual landmark and implement a hill-climbing algorithm to localize the robot relative to the landmark.

[Programming]

Exercise 13.16

Design four unique landmarks.

- a. Program the robot to visit each landmark in any order specified by a user.
- b. Place the landmarks at different locations. Implement Dijkstra's single source shortest path algorithm to compute the shortest path between two points specified by a user.
- c. Implement a *minimal spanning tree* algorithm to allow the robot to visit all waypoints efficiently.

13.12 End Notes

Of batteries and topological navigation.

The CSM entry did not place due to massive hardware failures with the power supply on Clementine. At one point, the team was using the car battery from the school van parked outside the Seattle Convention Center. We would drive up, park, and I would take the battery out of the van in the open. We would walk into the convention center with the battery and later that evening return it to the van and then drive to our housing in the wee hours of the morning. No security guard or policeman ever said a word.

The advantage of driving in circles.

One amusing aspect of QualNav is that the algorithm assumed the robot had cameras literally in the back of its head. Unfortunately the DARPA autonomous land vehicle being used to test the QualNav theory did not; all the sensors faced forward. Daryl Lawton tells the story of trying to convince the union driver of the vehicle to stop every 10 feet and do a 360° turn so the algorithm could record the viewframes of all the visible landmarks. After much pleading, the driver finally honored the request, though it was not clear if he ever understood why the crazy scientist wanted a high-tech truck capable of autonomous navigation to go in circles every 10 feet!

Topological navigation examples.

The figures and printouts of the topological navigation system used by the CSM team were prepared by Paul Wiebe.

14

Metric Path Planning and Motion Planning

Chapter Objectives:

- Define *Cspace*, *path relaxation*, *digitization bias*, *subgoal obsession*, and *termination condition*.
- Represent an indoor environment with a *generalized Voronoi graph*, a *regular grid*, or a *quadtree*, and create a graph suitable for path planning.
- Apply the *A* search algorithm* to a graph to find the optimal path between two locations.
- Explain the differences between *continuous* and *event-driven replanning*.
- Explain how the *D* search algorithm* accomplishes continuous replanning.
- Describe the *piano mover's problem* and why it is different than route planning.
- Explain the similarities and differences between *rapidly exploring random tree (RRT)* and *A** types of algorithms.
- List the criteria for evaluating a path planner, and, if given a description of a path or motion planner, use the criteria to rate the planner's utility.

14.1 Overview

Path planning attempts to answer the navigational question: *What is the best way there?* assuming a goal is known and a map exists. The previous chapter introduced topological navigation as one of two forms of navigation. Topological navigation relied on spatial memory organized as routes between landmarks, while the other form of navigation, metric navigation, expected layouts of the environment to reside in spatial memory. At this point, the reader may want a more detailed answer to *What is the difference between topological navigation and metric navigation/path planning?* From a practical implementation viewpoint, another question is *What is commonly used or works well enough?* A more subtle question is *How much path planning do you need?* The Nested Hierarchical Controller partitioning of deliberation described in chapter 12 suggests that planning is infrequent. Yet with advanced computing resources, is there any downside to continuous deliberative replanning?

This chapter will attempt to answer all of these questions. It will first try to distinguish topological and metric methods by presenting four different situations for which topological navigation is insufficient. As metric path and motion planners have two components—the *representation* (data structure) and the *algorithm*—the chapter will describe them separately. It will first cover how path planners partition the world into a structure, or map, amenable for path planning, called a configuration space. The intent of any representation is to represent only the salient features, or the relevant configuration of navigationally relevant objects in the space of interest; therefore, the term configuration space. Next the chapter will delve into the common algorithms for path planning, focusing on A^* and the popular D^* variant of search. Search algorithms can generally search through any configuration space representation, although, as with any algorithm, some methods work better when paired with certain data structures. Figure 14.1 offers three categories of metric path and motion planning. The path, or route, algorithms fall into two broad categories: those which treat path planning as a graph search problem applying a variant of the A^* algorithm, and wavefront propagation algorithms which treat path planning as a graphics coloring problem. The chapter will focus on the A^* and its popular D^* variant. The chapter will then turn to motion planning for manipulation and other pose-sensitive situations; these typically use a variant of the Rapidly evolving Random Tree (RRT) algorithm. Finally, the chapter covers executing a path and the issue of interleaving reaction and planning. Regardless of what algorithm is used, there is always the issue of when to plan, especially about how to interleave reaction and planning. Should a planner rerun after every sensor update or should the plan be modified only when the world changes or performance significantly degrades?

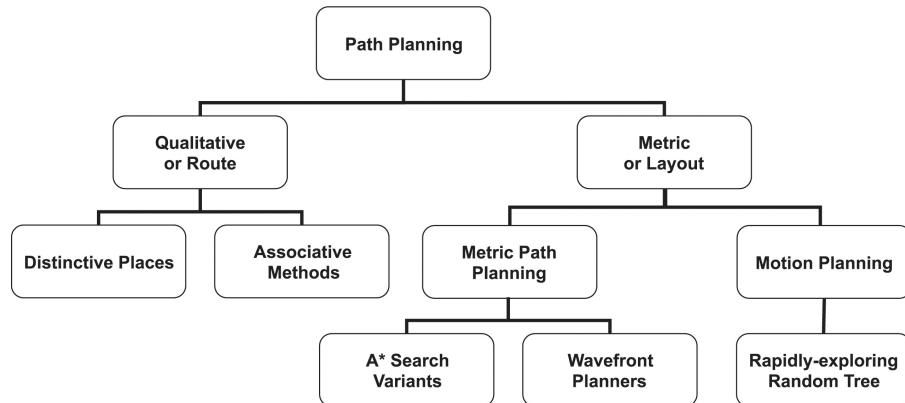


Figure 14.1 Taxonomy of path planning algorithms.

14.2 Four Situations Where Topological Navigation Is Not Sufficient

There are at least four common situations where topological navigation is not suitable. One situation occurs when the space between the starting point and the destination is not easily abstracted into labeled or perceptually distinct regions. For example, an unmanned aerial vehicle in the sky may have very few perceptual landmarks.

A second situation arises when the choice of route impacts the control or energy costs of the vehicle. For example, a fast moving robot may not be able to make a tight turn at high speed but could make

that turn if it were going slower. The robot might be able to make a wider turn without sacrificing speed and thus cover more distance but arrive at the final destination later. In theory, compensating for turning at high speeds could be incorporated into a local control strategy, but that would require having experimented and learned the right vehicle dynamics, risking crashing the platform. Ideally, the robot could generate the safe choice of routes directly from an *a priori* map. Another example is a ground robot that may prefer routes with gentle slopes or which do not require energy-intensive climbing or aerial vehicles that choose flights that minimize crosswinds.

The third case can occur when the purpose of the path is to allow sensor coverage of an area. For example, an unmanned marine vehicle may need to conduct a lawnmower scan of an area of the ocean to search for a sunken ship or airplane wreck. Another example might involve a UAV surveying a field or construction site. In these cases, the route would be specified as a sequence of locations, or waypoints, in a coordinate frame.

A fourth situation can arise when the pose of the robot is important, either while reaching a destination or during coverage of an area. Consider moving a piano down a flight of stairs, where the piano has to be flipped on one side and rotated to get through tight spaces; this is aptly named *the piano mover's problem*. Since real robots are not holonomic, the piano mover's problem quickly emerges in confined or cluttered spaces. The piano problem also emerges in moving a robot arm from one location to another in between clutter. Robot arm or leg movement is generally called motion planning to distinguish itself from the macro movements of the entire platform.

The pose can also be important in coverage applications. In aerial surveys, the overall quality of the survey is improved if the images of the area are taken from the same altitude and with the camera perpendicular to the ground. However, in a lawnmower scan, a fixed-wing UAV has to turn. That means the camera is no longer facing straight down and the altitude may change slightly. Therefore, fixed-wing UAVs incorporate overshooting to give enough distance for the UAV to turn and resume the correct pose before it returned to surveying.

WAYPOINTS

These four situations lead to a different flavor of path planning, called metric path planning and motion planning. For the purpose of providing an overview, metric path planning and motion planning will be referred to generically as *metric or layout oriented* methods. In metric methods, the robot has an *a priori* map of the environment and generates a plan of appropriate movements to reach the goal or degree of coverage. Metric methods generally favor techniques which produce an optimal path or sequence of motions, according to some measure of best, whereas the qualitative methods in chapter 13 were content to produce a route with identifiable landmarks or gateways. Another difference is that metric paths usually decompose the path into subgoals consisting of *waypoints*. These waypoints are most often fixed locations or coordinates (x,y) or locations and poses. These locations may have mathematical meaning, but they may not have a sensible correspondence to objects or landmarks in the world, for example, “go 15 meters, turn – 90°” requires localization and mapping as compared to “go about 15 meters to landmark L, turn left,” which requires sensing landmarks.

The terms “optimal” and “best” have serious ramifications for robotics. To say a path is optimal implies comparison. As will be seen, some metric methods are able to produce an optimal path because they consider all possible paths between points. Surprisingly, an optimal path may not appear optimal to the human eye; for example, a mathematically optimal path of a world divided into tiles or grids may

be very jagged and irregular rather than straight. The ability to produce and compare all possible paths also assumes that the planning has access to a pre-existing (or a priori) map of the world. Equally as important, path planning with an a priori map assumes that the map is accurate and up to date. As such, metric methods are compatible with deliberation, while qualitative methods work well with more reactive systems. As a deliberative function, metric methods tend to be plagued by challenges in world representation, handling dynamic changes and surprises, and computation complexity.

14.3 Configuration Space

CONFIGURATION SPACE
(CSPACE)

The physical space robots and obstacles exist in can be thought of as the world space. The *configuration space*, or *Cspace* for short, is a data structure which allows the robot to specify the position (location and orientation) of itself and any other objects and the robot.

DEGREES OF FREEDOM

A good Cspace representation reduces the number of dimensions that a planner has to contend with. Consider that it takes six dimensions (also called *degrees of freedom* or DOF) to represent precisely where an object is. A person may specify the location of the object as a set of (x, y, z) coordinates in some frame of reference. But an object is three-dimensional; it has a front and back, top and bottom. Three more degrees are needed to represent where the front of the chair is facing, whether it is tilted or not or even upside down. These are the Euler (pronounced “oiler”) angles, ϕ, θ, γ , also known as pitch, yaw, and roll.

Six degrees of freedom are more than are needed for a mobile ground robot, in most cases, for planning a path. The z (height) coordinate can be eliminated if the robot and all objects sit on the floor. However, the z coordinate will be of interest if the robot is an aerial or underwater vehicle. Likewise, the Euler angles may be unnecessary. Who cares which way the robot is facing if all the robot wants to do is to plan a path around an obstacle? But the pitch of a planetary rover or slope of an upcoming hill may be critical to a mission over rocky terrain. [Figure 14.2](#) shows a transformation of an object into Cspace. In general, metric path planning algorithms for mobile robots have assumed only two DOF, translation and rotation. The world is described in two dimensions (x, y) and the robot can only move forward or backward, and turn within the (x, y) plane.

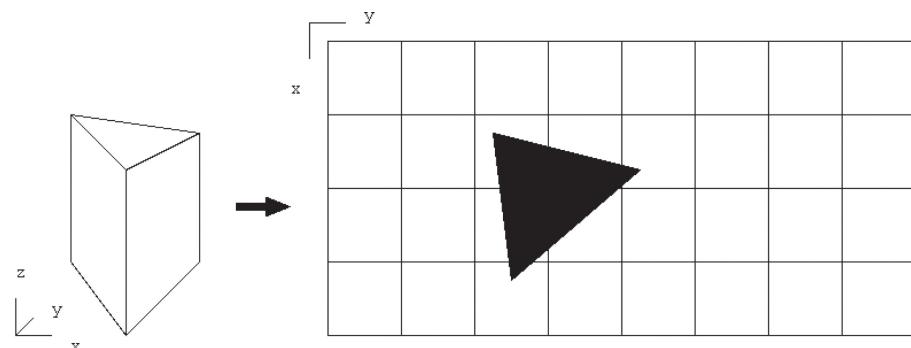


Figure 14.2 Reduction of a 6DOF world space to a 2DOF configuration space.

The number of different Cspace representations is too large to include more than a coarse sampling here. The subsections below provide details on three important Cspace representations: meadow maps, GVG, and grids. However, it is useful to start with an overview of the three representations. One group of methods divides the world into irregular polygonal regions of empty spaces, such as meadow maps and generalized Voronoi graphs (GVG). Meadow maps have not been used frequently in the past few years, but they are intuitive and illustrate the best practices of growing obstacles and path relaxation and how a path can be obtained by searching through the geometry. Meadow maps also illustrate the major problems with region oriented methods. One is that the polygons generated by the algorithm are not necessarily unique; therefore a different algorithm or starting point will produce a different map. Another problem is the boundaries are not guaranteed to be easy for the robot to perceive reliably and thus to know where it is. However, GVGs and other region-oriented methods are experiencing a resurgence because range sensing and dense point clouds make extraction of volumes more reliable and consistent. Rather than try to find polygons of open space, an alternative is a brute-force type of representation where the space is divided in some regular, reproducible way and then each division is marked as empty or occupied. Essentially these methods superimpose a grid over the world. A regular grid provides a simple partitioning. Unfortunately the smaller the grid element size, the higher the resolution of the configuration space but the more elements that have to be searched, increasing the computational time and space needed for planning. Thus, recursive grid partitioning methods, most notably quad- and oct-trees, have emerged.

14.3.1 Meadow Maps

MEADOW MAP

Many early path planning algorithms developed for mobile robots assumed that the robot would have a highly accurate map of the world in advance. This map could be digitized in some manner, and then the robot could apply various algorithms to convert the map to a suitable Cspace representation. An example of a Cspace representation that might be used with an a priori map is the *meadow map* or hybrid vertex-graph free-space model (see [figure 14.3](#)).

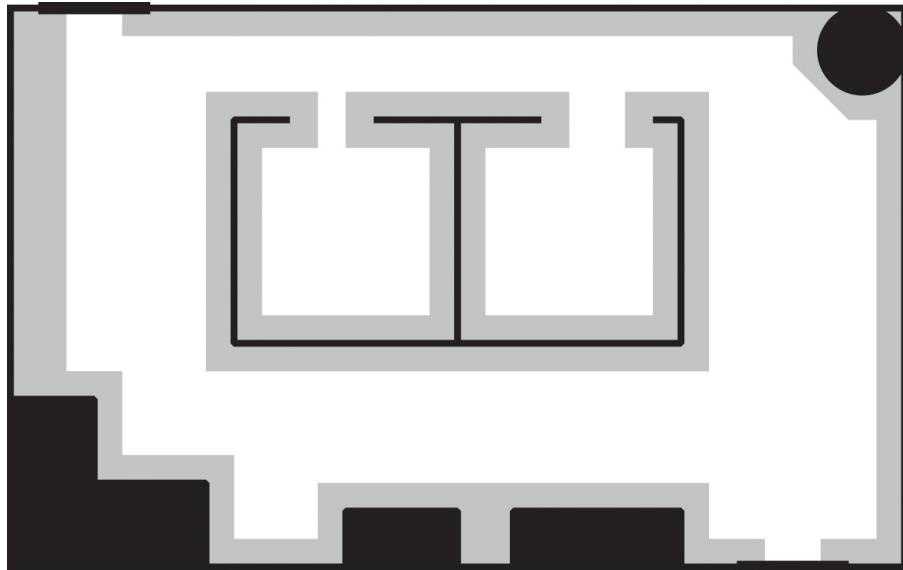


Figure 14.3 An a priori map, with object boundaries “grown” to the width of the robot (shown in gray). The robot is not shown on the map.

Meadow maps transform free space into convex polygons. Convex polygons have an important property: if the robot starts on the perimeter and goes in a straight line to any other point on the perimeter, it will not go out of the polygon. The polygon represents a safe region for the robot to traverse. The path planning problem becomes a matter of picking the best series of polygons to transit through. Meadow maps are not that common in robotics, but they serve to illustrate the principles of effecting a configuration space and then planning a path over it.

The programming steps are straightforward. First, the planner begins with a metric layout of the world space. Most planners will increase the size of every object by adding the size of the robot to the object boundary; this allows the planner to treat the robot as if it were a point, not a 2D object. Notice that path planners implicitly assume holonomic vehicles from the very first step.

The next step in the path planning algorithm is to construct convex polygons by considering the line segments between pairs of interesting features. In the case of indoor maps, these are usually corners, doorways, boundaries of objects, and so forth. The algorithm can then determine which combination of line segments partition the free space in convex polygons.

The meadow map is now technically complete, but it is not in a format that supports path planning. Each convex polygon represents a safe passage for the robot. But there is still some work to be done. Some of the line segments forming the perimeter are not connected to another polygon (i.e., they are part of a wall), so they should be off limits to the planning algorithm. Also, as can be seen in the above figure, some of the line segments are quite long. It would make a difference to the overall path length where the robot cuts across the polygon. It is hard for the planner to discretize a continuous line segment. So the issue becomes how to specify candidate points on the polygon. One solution is to find the middle of each line segment that borders another polygon. Note that each of these midpoints becomes a node, and if edges are drawn between them, an undirected graph emerges. A path planning

algorithm would determine the best path to take.

PATH RELAXATION

One disadvantage of a meadow map, or, indeed of any Cspace representation, is evident on inspection of [figure 14.4](#). Any path which is chosen will be somewhat jagged. Each of the inflection points is essentially a waypoint. One outcome of the partitioning process is that the free space is divided in a way that makes sense geometrically, but it does not necessarily work for a robot that must travel through that space. Why go halfway down the hall and then angle off? The partitioning may be mathematically optimal on paper, but in practice, it seems downright silly. Chuck Thorpe, while at Carnegie Mellon University, devised a solution to paths generated from any kind of discretization of free space (see [figure 14.5](#)).²⁰⁵ Imagine that the path is a string. Now imagine pulling on both ends to tighten the string (the technical name for this is *path relaxation*). This string-tightening algorithm would remove most of the kinks from the path without violating the safe-zone property of convex polygons.

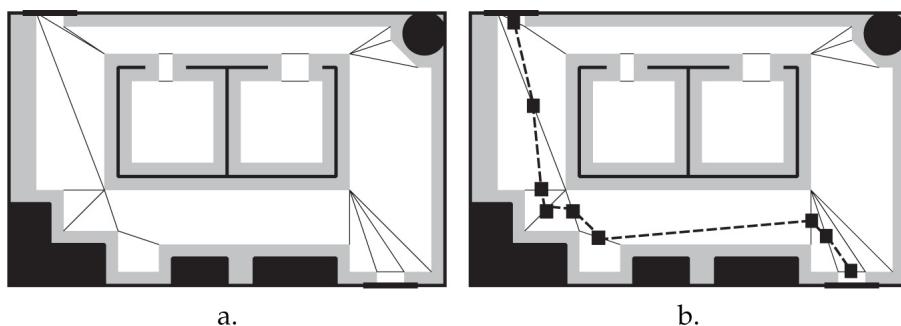


Figure 14.4 Meadow maps: a.) partitioning of free space into convex polygons and b.) generating a graph using the midpoints.

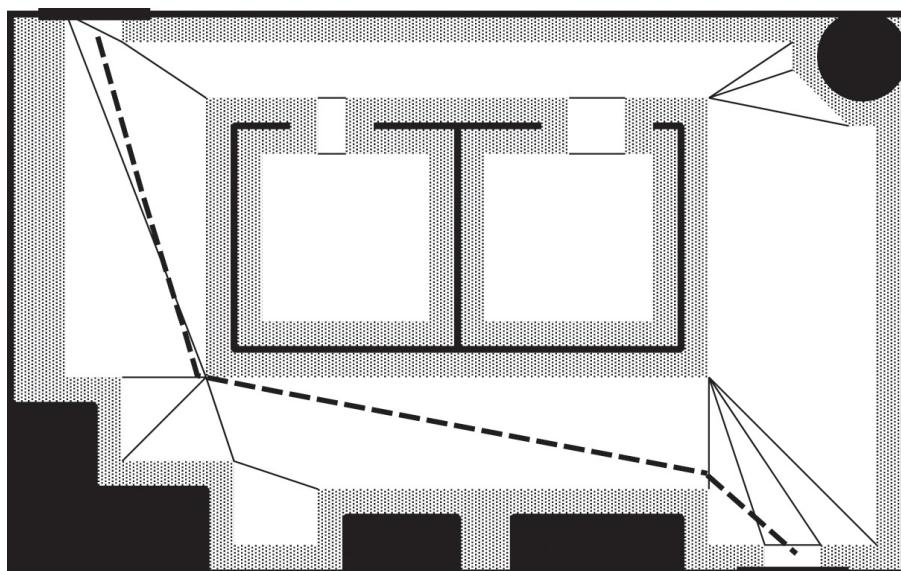


Figure 14.5 String tightening as a relaxation of an initial path.

Meadow maps have three problems which limit their usefulness. One, the technique to generate the polygons is computationally complex. But more importantly, it uses artifacts of the map to determine the polygon boundaries rather than things which can be sensed. Unless the robot has accurate localization, how does it know it is halfway down a long, featureless hall and should turn 30°? The third major disadvantage is that it is unclear how to update or repair the diagrams as the robot discovers discrepancies between the a priori map and the real world.

14.3.2 Generalized Voronoi Graphs

GENERALIZED VORONOI GRAPH

(GVG)

A *generalized Voronoi graph*, or *GVG*, is a popular mechanism for representing Cspace and generating a graph. Unlike a meadow map, a GVG can be constructed as the robot enters a new environment, thereby creating a topological map as shown by Howie Choset at Carnegie Mellon University.⁴⁵

VORONOI EDGE

The basic idea of a GVG is to generate a line, called a *Voronoi edge*, equidistant from all points. As seen in figure 14.6, this line goes down the middle of hallways and openings. The point where many Voronoi edges meet is known as a *Voronoi vertex*. Notice the vertices often have a physical correspondence to configurations that can be sensed in the environment. This makes it much easier for a robot to follow a path generated from a GVG, since there is an implicit local control strategy for staying equidistant from all obstacles.

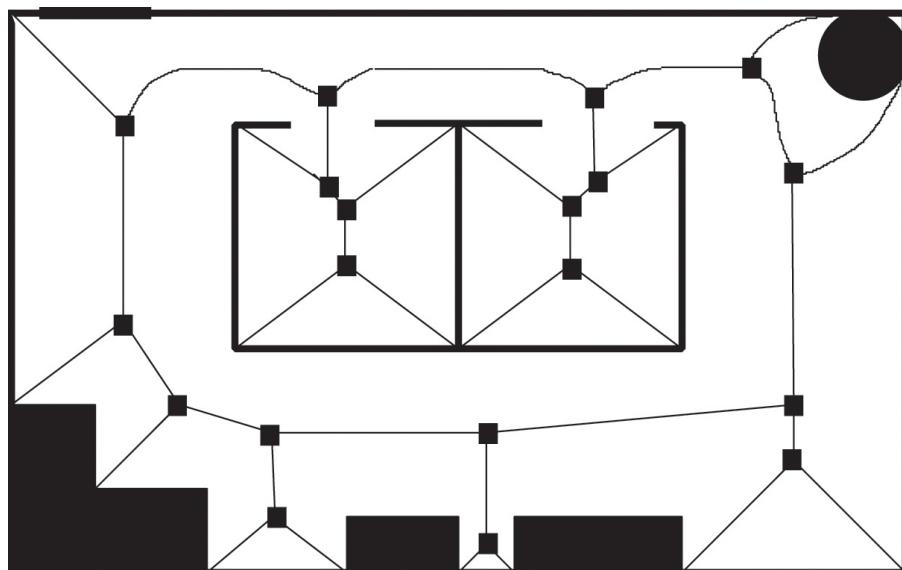


Figure 14.6 Graph from a generalized Voronoi graph (GVG).

If the robot follows the Voronoi edge, it will not collide with any modeled obstacles because it is staying “in the middle.” This obviates the need to grow the obstacle boundaries, as with the meadow map. Edges serve as freeways or major thoroughfares. It should also be noted that the curved edges in a GVG do not matter to graph theory or graph algorithms. It is only the length, not the physical reality, of the edges that make any difference.

14.3.3 Regular Grids

REGULAR GRID

Another method of partitioning the world space is a regular grid. The *regular grid* method superimposes a 2D Cartesian grid on the world space, as shown in [figure 14.7](#). If there is any object in the area contained by a grid element, that element is marked “occupied.” Therefore, regular grids are often referred to as occupancy grids. Occupancy grids will be detailed in chapter 15.

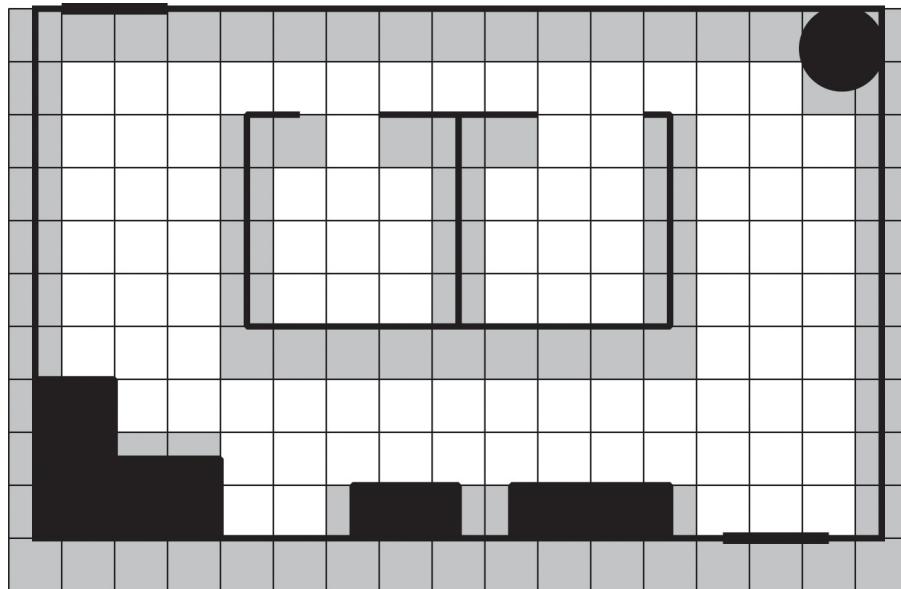


Figure 14.7 Regular grid.

4-CONNECTED NEIGHBORS

8-CONNECTED NEIGHBORS

Regular grids are straightforward to apply. The center of each element in the grid can become a node, transforming the grid into a highly connected graph. Grids are either considered *4-connected* or *8-connected*, depending on whether they permit an arc to be drawn diagonally between nodes.

DIGITIZATION BIAS

Unfortunately, regular grids are not without problems. First, they introduce *digitization bias*, which means that if an object falls into even the smallest portion of a grid element, the whole element is marked “occupied.” This leads to wasted space and to very jagged objects. To reduce the wasted space, regular grids for an indoor room are often finely grained, on the order of 4- to 6-inches square. This fine granularity means a high storage cost and a high number of nodes for a path planning algorithm to consider.

14.3.4 Quadtrees

QUADTREES

Quadtrees are a variant on regular grids that attempt to avoid wasted space (see [figure 14.8](#)). A quadtree is a recursive grid. The representation starts out with grid elements representing a large area, perhaps 64 inches square (8 by 8 inches). If an object falls into part of the grid but not all of it, the Cspace algorithm divides the element into four (thus the name “quad”) smaller grids, each 16 inches square. If the object does not fill a particular subelement, the algorithm does another recursive division of that element into four more subelements, represented a 4 inches square region. A three-dimensional quadtree is called an *octree*.

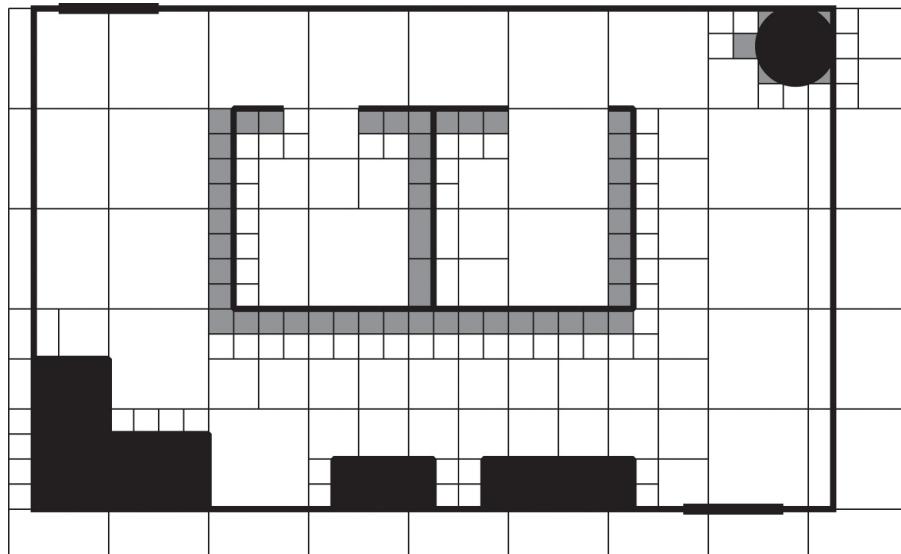


Figure 14.8 Quadtree Cspace representation.

14.4 Metric Path Planning

Metric methods for destination or route planning rely on an a priori map. Graph methods transform the a priori map into a graph-like Cspace, then use a graph search algorithm to find the shortest path in the graph between the start and goal nodes. A^* is a classic AI graph search algorithm, with the D^* variant tailored for robots. An alternative is to exploit graph coloring algorithms from graphics.

14.4.1 A* and Graph-Based Planners

INITIAL NODE

GOAL NODE

As seen earlier in this chapter, most Cspace representations can be converted to graphs. This means that the path between the *initial node* and the *goal node* can be computed using graph search algorithms. Graph search algorithms appear in networks and routing problems, so it is a class of algorithms which is well understood by computer scientists. However, many of these algorithms require the program to visit each node on the graph to determine the shortest path between the initial and goal nodes. Visiting every node may be computationally tractable for a sparsely connected graph, such as derived from a Voronoi diagram, but it is computationally expensive for a highly connected graph such as one generated from a regular grid. Therefore, there has been a great deal of interest in applying path planners that do a “branch and bound” style of search; that is, planners that prune off paths which are not optimal. Of course, the trick is knowing when to prune!

A* SEARCH ALGORITHM

The *A* search algorithm* is the classic method for computing optimal paths for holonomic robots. It is derived from the *A search algorithm*. *A search* will be first presented with an example, then *A** (as it is commonly called) will be introduced by building on that example. Both assume a metric map where the location of each node is known in absolute coordinates, and the graph edges represent whether it is possible to traverse between those nodes.

The *A search* algorithm produces an optimal path by starting at the initial node and then working through the graph to the goal node. It generates the optimal path incrementally; at each update, it considers the nodes that could be added to the path and picks the best one. It picks the “right” node to add to the path every time it expands the path (the “right node” is more formally known as the plausible move). The heart of the method is the formula (or evaluation function) for measuring the plausibility of a node:

$$f(n) = g(n) + h(n)$$

where:

- $f(n)$ measures how good the move to node n is
- $g(n)$ measures the cost of getting to node n from the initial node. When *A* expands from the initial node outward, this is the distance of the path generated so far plus the distance of the edge to node n
- $h(n)$ represents the lowest cost of getting from n to goal

Consider how the formula is used in the example below. Assume that a Cspace representation yielded the graph in [figure 14.9](#).

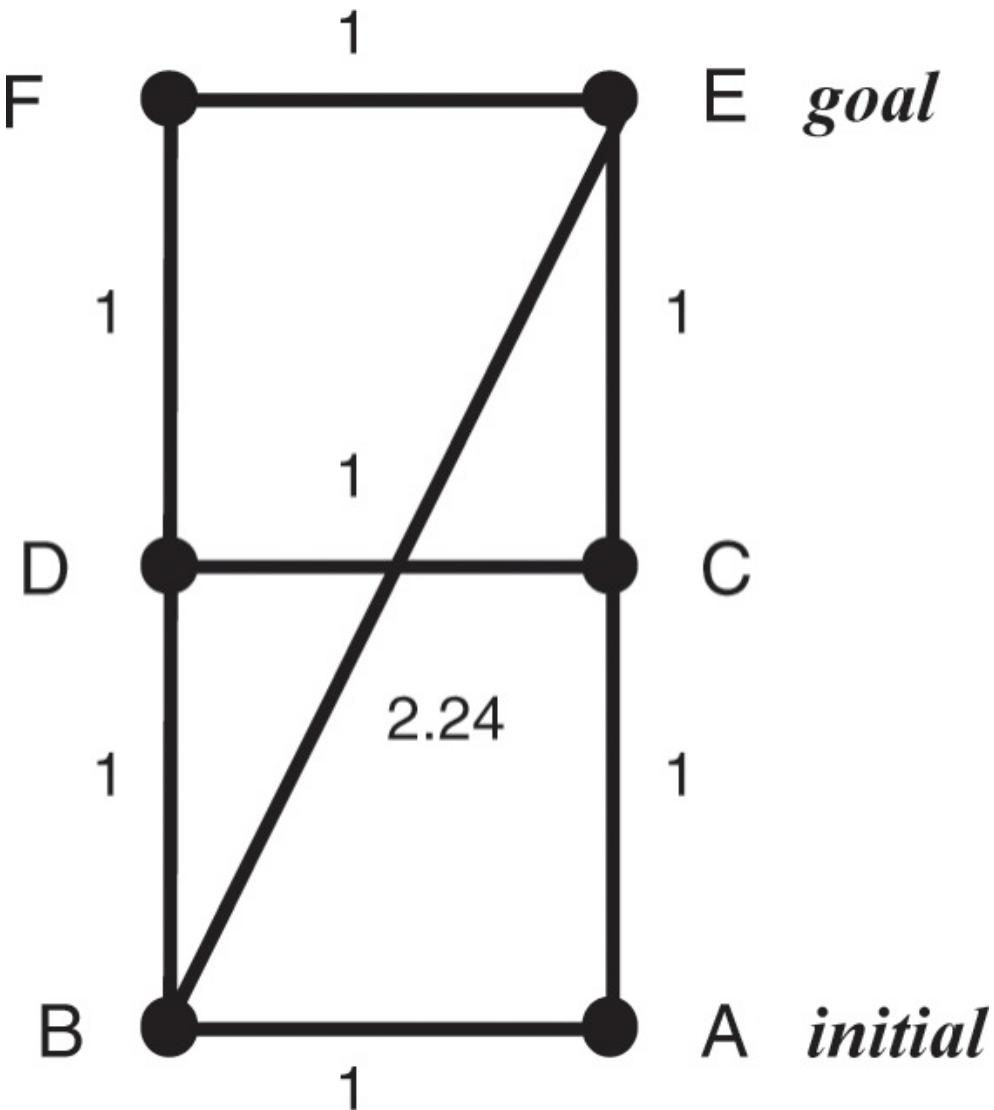


Figure 14.9 Graph for an *A search* algorithm example.

The *A search* algorithm begins at node *A* and creates a decision tree-like structure to determine which are the possible nodes it can consider adding to its path. There are only two nodes to choose from: *B* and *C*.

In order to determine which node is the right node to add, the *A search* algorithm evaluates the plausibility of adding *B* or *C* by looking at the edges. The plausibility of adding *B* as the next move is:

$$f(B) = g(B) + h(B) = 1 + 2.24 = 3.24,$$

where $g(B)$ is the cost of going from *A* to *B*, and $h(B)$ is the cost from *B* to the goal *E*.

The plausibility of adding *C* is:

$$f(C) = g(C) + h(C) = 1 + 1 = 2.0,$$

where $g(C)$ is the cost of going from A to C , and $h(C)$ is the cost of getting from C to E . Since $f(C) > f(B)$, the path should go from A to C .

But this assumes that $h(n)$ is known at every node. This means that the algorithm has to recurse in order to find the correct value of $h(n)$. This means the algorithm has to visit all the nodes.

A search is guaranteed to produce the optimal path because it compares all possible paths to each other. *A* search* takes an interesting approach to reducing the number of paths that have to be generated and compared: the algorithm compares the possible paths to the best possible path, even if there is not a path in the real world that goes that way. The algorithm estimates h rather than checks to see if there is actually a path segment that can get to the goal in that distance. The estimate can then be used to determine which nodes are the most promising and when certain paths have no chance of reaching the goal with a shorter path than other candidates and thus should be pruned from the search.

Under *A** the evaluation function becomes:

$$f^*(n) = g^*(n) + h^*(n),$$

ADMISSIBILITY CONDITION

HEURISTIC FUNCTION

where the * means that the functions are estimates of the values that would have been plugged into the *A search* evaluation. In path planning, $g^*(n)$ is the same as $g(n)$: the cost of getting from the initial node to n , which is known through the incremental build of the path. $h^*(n)$ is the real difference. So what is a way to estimate the cost of going from n to the goal? Furthermore, how can we be sure that the estimate will be accurate enough so that we do not end up choosing a path which is not truly optimal? This can be done by making sure that $h^*(n)$ will never be smaller than $h(n)$. The restriction that $h^*(n) \leq h(n)$ is called the *admissibility condition*. Since $h^*(n)$ is an estimate, it is also called a *heuristic function*, since it uses a rule of thumb to decide which is the best node to consider.

Fortunately, there is a natural heuristic function for estimating the cost from n to the goal: the Euclidean (straight line) distance. Recall that the locations of each node are known independent of the edges. Therefore, it is straightforward to compute the straight-line distance between two nodes. The straight-line distance is always the shortest path between two points, barring curvature of the earth, and so forth. Since the real path can never be shorter than that, the admissibility condition is satisfied.

To see how *A** uses this to actually eliminate visiting nodes, consider [figure 14.10](#). As with *A search*, the first step in *A** is to consider the choices from the initial node.

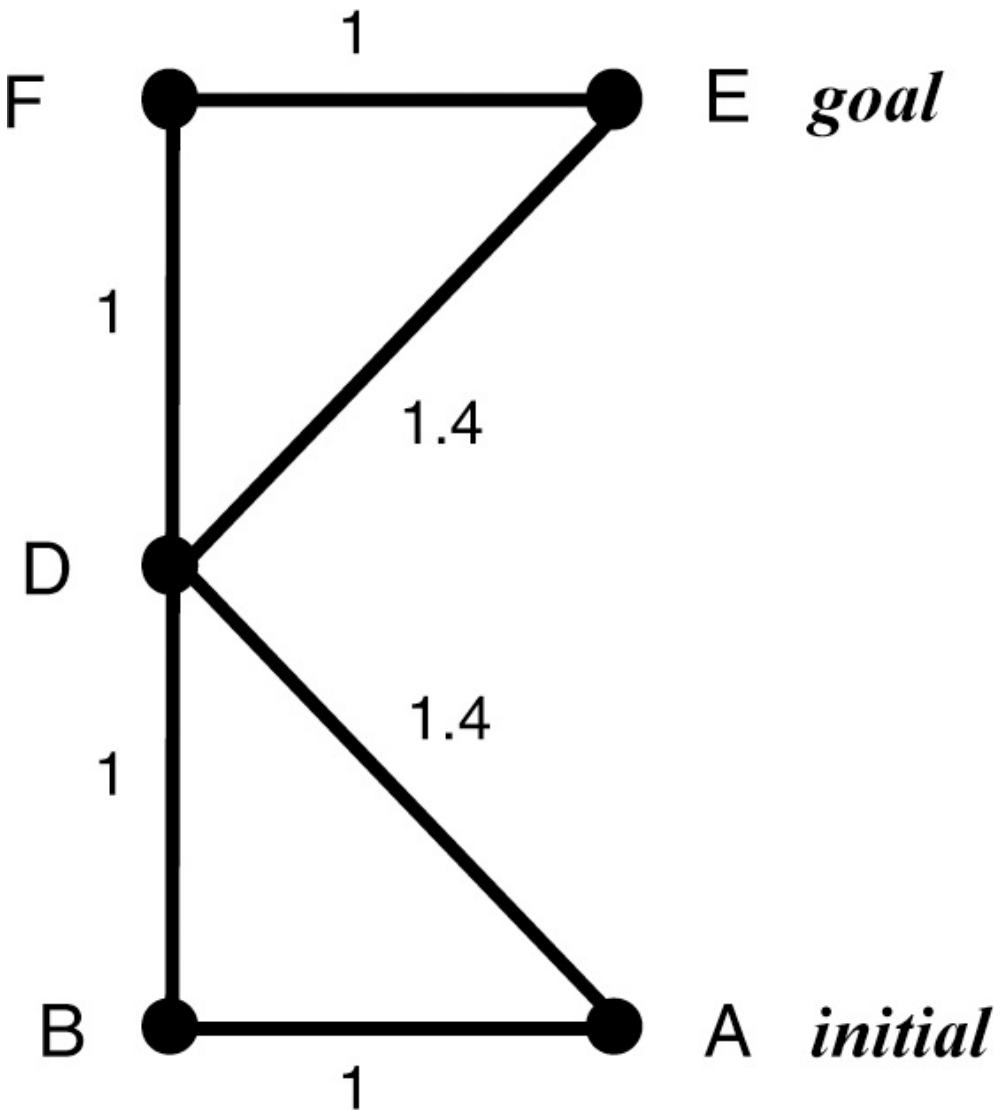
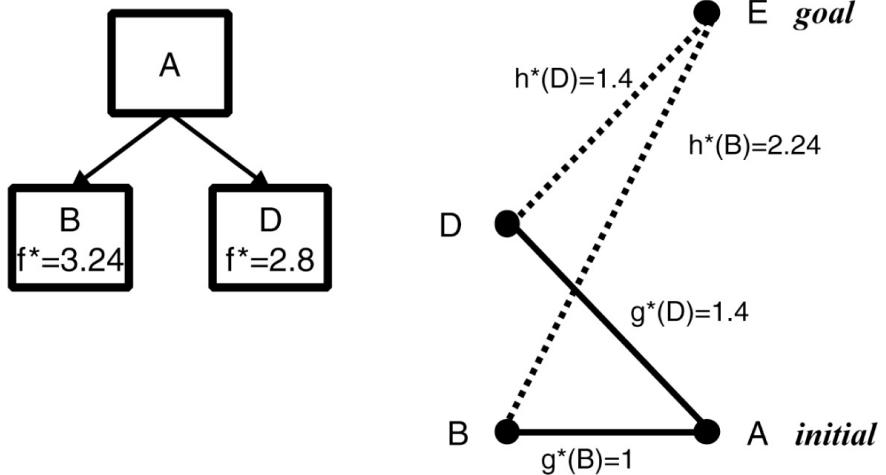


Figure 14.10 An A* example.

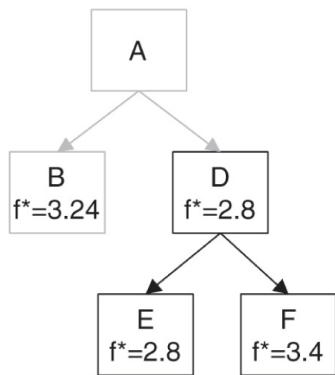
The choices are B and D , which can be thought of as a search tree as shown in [figure 14.11a](#) or as a subset of the original graph (see [figure 14.11b](#)). Regardless of how it is visualized, each node is evaluated to determine if it is the most plausible move. The above figure shows what the algorithm “sees” at this point in the execution. The choices to be evaluated are:

$$f^*(B) = g^*(B) + h^*(B) = 1 + 2.24 = 3.24$$

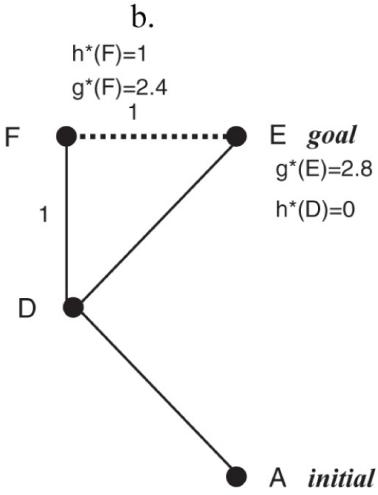
$$f^*(D) = g^*(D) + h^*(D) = 1.4 + 1.4 = 2.8$$



a.



c.



d.

Figure 14.11 a.) What A^* “sees” as it considers a path $A-? - E$, and b.) the original graph. c.) What A^* “sees” from considering a path $A-D-? - E$, and d.) the associated graph.

A path going from $A - D - ? - E$ has the potential to be shorter than a path going from $A - B - ? - E$. So, D is the most plausible node. Notice that A^* cannot eliminate a path through B because the algorithm cannot “see” a path that actually goes from D to E and determine if it is indeed as short as possible.

As step 2, A^* recurses (repeats the evaluation) from D , since D is the most plausible, as shown in figure 14.11.

The two options from D are E and F , which are evaluated next are

$$f^*(E) = g^*(E) + h^*(E) = 2.8 + 0 = 2.8$$

$$f^*(F) = g^*(F) + h^*(F) = 2.4 + 1.0 = 3.4$$

Now the algorithm sees that E is the best choice among the leaves of the search tree, including the branch through B . (If B was the best choice, then the algorithm would have changed branches.) E is a better choice than F and B . When the algorithm goes to expand E , it notices that E is the goal, so the algorithm is completed. The optimal path is $A - D - E$, and we did not have to consider $A - B - F - E$ explicitly. There are other ways to improve the procedure described so far. $f^*(F)$ did not need to be computed if the algorithm looks at its choices and sees that one of them is the goal. Any other path choice has to be longer because edges are not allowed to be negative, so $D - F - E$ has to be longer than $D - E$.

Another important insight is that any path through A to E has to go through D , so the B branch of the search tree could have been pruned. Of course, in the above example the algorithm never had an opportunity to notice this because B was never expanded. For a larger graph, it is easy to imagine that there might be a case where, after several expansions through D , the leaf at B in the search tree might come up as the most plausible. Then the algorithm would have expanded A and seen that the choice was D . Since D had already occurred in another branch, with a cheaper $g^*(D)$, the B branch could be pruned safely. This is particularly useful when A^* is applied to the highly connected graph created from a regular grid versus the sparse graphs that occur in non-robotic planning applications.

One very attractive feature of A^* path planners is that it can be used with any Cspace representation that can be transformed into a graph. The major impact that Cspace has on the A^* planner is how many computations it takes to find the path.

A limitation of A^* is that it is very hard to use for path planning where there are factors other than distance to consider in generating the path. For example, the straight-line distance may cover rocky terrain or sand that poses a risk to the robot. Likewise, the robot may wish to avoid going over hills in order to conserve energy but, at the same time, might wish to go down hills whenever possible for the same reason. In order to factor in the impact of terrain on path costs, the heuristic function $h^*(n)$ has to be changed. But recall that the new heuristic function must continue to satisfy the admissibility condition: $h^* \leq h$. If the new h^* just takes the worst-case energy cost or safety cost, it will be admissible but not particularly useful in pruning paths. Also, gaining energy going downhill is essentially having an edge in the graph with a negative weight, which A^* cannot handle. (Negative weights pose an interesting problem in that the robot can get into a loop of rolling down a hill repeatedly because it is an energy efficient solution; however, this solution does not allow any forward progress! Bellman-Ford types of algorithms deal with this situation.)

14.4.2 Wavefront-Based Planners

Wavefront propagation styles of planners are well-suited for grid types of representations but they never gained popularity in robotics. The basic principle is that a wavefront considers the Cspace to be a conductive material with heat radiating from the initial node to the goal node. Eventually the heat will spread and reach the goal if there is a way, as shown in the sequence in [figure 14.12](#). Another analogy for wavefront planners is region coloring in graphics, where the color spreads out to neighboring pixels. Two interesting aspects of wavefront propagation are i) the optimal path from all grid elements

to the goal can be computed as a side effect, and ii) any costs of navigating through a terrain can be incorporated as a different conductivity. Wavefront propagation results in a map which looks like a potential field. One of the many wavefront types of path planners is the Trulla algorithm developed by Ken Hughes.¹⁴⁶ It exploits similarities with a potential field that let the path itself represent what the robot should do as if the path were a sensor observation.

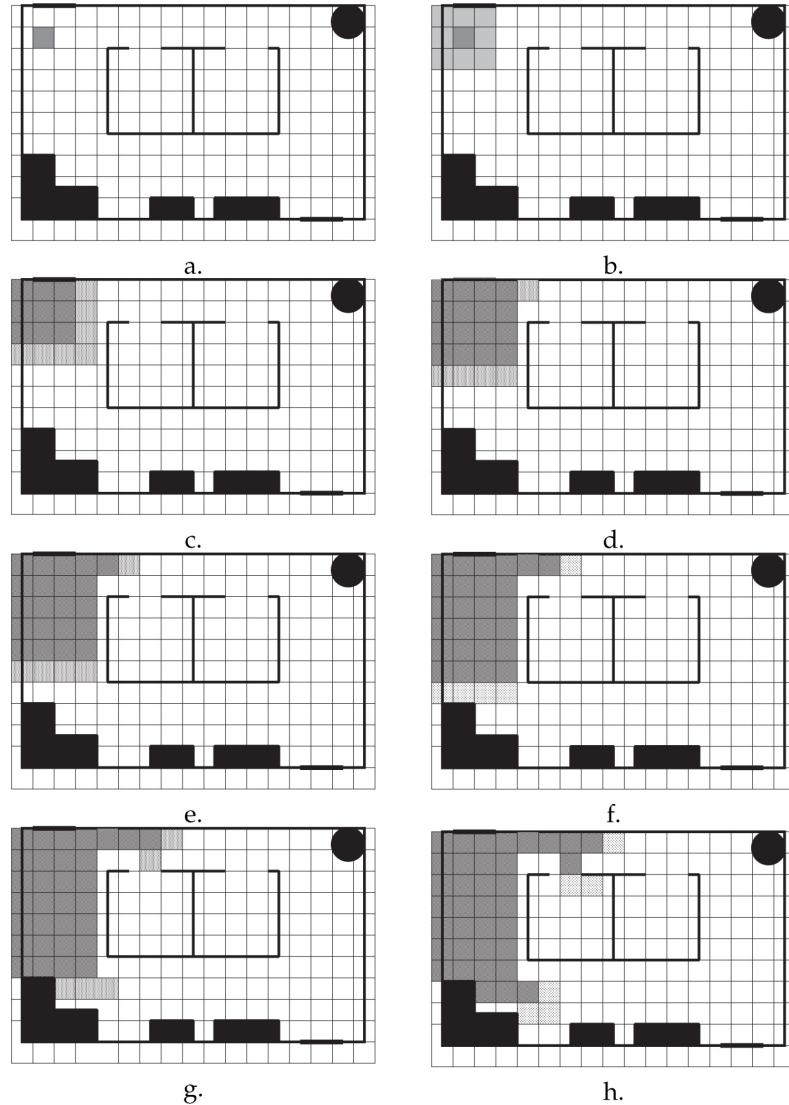


Figure 14.12 A wave propagating through a regular grid. Elements holding the current front are shown in gray; older elements are shown in dark gray.

14.5 Executing a Planned Path

Most path planning algorithms are employed in a strict plan-once, reactively-execute style. Almost all techniques break a path into segments; even a wavefront planner actually produces a goal location (waypoint) for each directional vector. Decomposition of a path into segments is well suited for an architecture with a Cartographer handing off path segments, or an entire path, to a Sequencer. The Sequencer can then employ a series of move-to-goal behavior instances, deactivating and re-instantiating the behavior as a new subgoal is met. Unfortunately there are two problems with reactive execution of a metric path as described above: *subgoal obsession* and the lack of *opportunistic replanning*.

14.5.1 Subgoal Obsession

SUBGOAL OBSESSION

TERMINATION CONDITION

Subgoal obsession occurs when the robot spends too much time and energy trying to reach the exact subgoal position, or more precisely, when the termination conditions are set with an unrealistic tolerance. The problem with setting a *termination condition* for subgoals is best described with an example. Suppose the next waypoint is at location (35, 50). If the robot has shaft encoders or GPS, determining that the robot has arrived at the waypoint should be straight forward to do in theory. In practice, it is very hard for a robot, even a holonomic robot, to reach any location exactly because it is hard for the robot to move precisely. The robot may reach (34.5, 50). The behavior sees that the goal is now 0.5 meters ahead, and the robot moves again to attempt to reach (35, 50). On that move, it may overshoot and end up at (35.5, 50.5). Now it has to turn and move again, resulting in see-sawing motions that can go on for minutes. This wastes time and energy, and makes the robot appear to be unintelligent. The problem is exacerbated by nonholonomic vehicles which may have to back up in order to turn to reach a particular location. Backing up almost always introduces more errors in navigation.

To handle subgoal obsession, many roboticists program into their move-to-goal behaviors a tolerance on the termination condition for reaching a goal. A common heuristic for holonomic robots is to place a tolerance of $+/-$ the width of the robot. So if a cylindrical, holonomic robot, with a diameter of 1.0 meter, is given a goal of (35, 50), it will stop when $34.5 < x < 35.5$ and $49.5 < y < 50.5$. There is no common heuristic for nonholonomic robots because the maneuverability of each platform is different. A more vexing aspect of subgoal obsession is when the goal is blocked, and the robot cannot reach the termination condition. For example, consider a subgoal at the opposite end of a hall from a robot, but the hall is blocked and there is no way around. Because the robot is executing reactively, it does not necessarily realize that it is not making progress. One solution is for the Sequencer to estimate a maximum allowable time for the robot to reach the goal. This estimate can be implemented either as a parameter on the behavior (terminate with an error code after n seconds), or as an internal state releaser. The advantage of the latter solution is that the code can become part of a monitor, leading to some form of self-awareness.

14.5.2 Replanning

Related to subgoal obsession is the fact that execution of plans often lacks opportunistic improvements.

Suppose that the robot is heading for Subgoal 2 when an unmodeled obstacle diverts the robot from its intended path. Now suppose that, from its new position, the robot can perceive Subgoal 3. In a classic Nested Hierarchical Controller style of implementation, the robot would not be looking for Subgoal 3, so it would continue to move to Subgoal 2 even though it would be more optimal for the robot to head straight for Subgoal 3.

The need to replan opportunistically also arises when an a priori map turns out to be incorrect. What happens when the robot discovers it is being sent through a patch of muddy ground? Trying to navigate reactively around the mud patch seems unintelligent because choosing left or right may have serious consequences if the robot gets stuck in the mud. Instead the robot should return control to the Cartographer, which would update its map and replan. The issue becomes: How does a robot know it has deviated too far from its intended path and needs to replan?

There are two approaches to replanning. One is to continuously replan, essentially imposing a hierarchical Sense, Plan, Act cycle. The other is to replan when there is some event, exception, or indication that the plan execution is not working. Event-driven replanning can be used in a hybrid Plan, then Sense-Act architecture but it requires the addition of deliberative *monitoring*.

D* SEARCH ALGORITHM

The *D** search algorithm is a popular example of continuous replanning, while an extension to the Trulla algorithm is an example of event-driven replanning. Both planners begin with an a priori map and compute the optimal path from every location to the goal. *D** does it by executing an *A** search from each possible location to the goal in advance; this reformulation transforms *A** from a single-source shortest path algorithm into an all-paths algorithm. All-paths algorithms are computationally expensive and time-consuming, but that is not a problem since the paths are computed when the robot starts the mission and is sitting still. Since Trulla is a wavefront type of planner, it generates the optimal path between all pairs of points in Cspace as a side effect of computing the path from the starting location to the goal.

Computing the optimal path from every location to the goal actually helps with reactive execution of the path. It means that, if the robot can localize itself on the a priori map, the robot can read the optimal subgoal for move-to-goal on each update. If the robot has to swing wide to avoid an unmodeled obstacle in [figure 14.13](#), the robot automatically becomes redirected to the optimal path without having to replan. Note how the metric path becomes a virtual sensor, guiding the move to goal behavior by replacing the direct sensor data. This is an example of how the deliberative (path planning) and reactive (execution) components of Hybrid architectures interact.

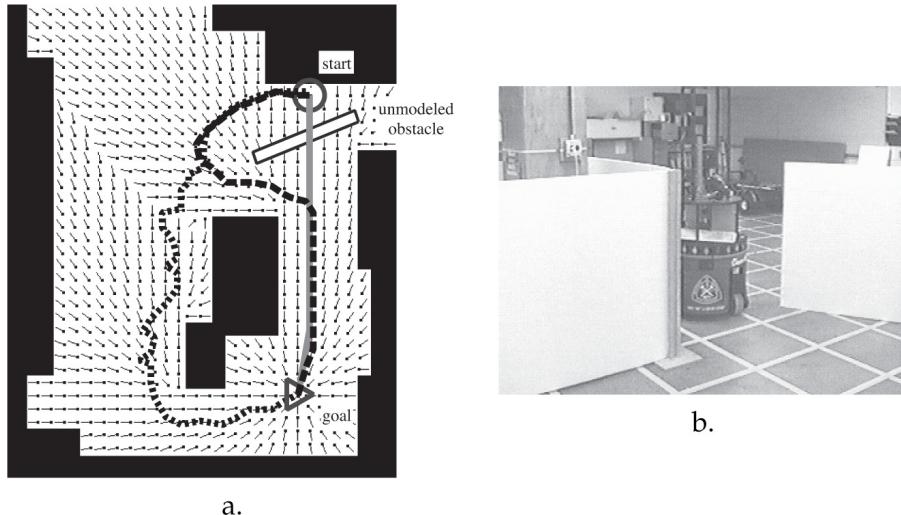


Figure 14.13 Layout showing unmodeled obstacle. a.) Gray line shows expected path, long dashed line shows the actual path with Trulla, and short dashed line shows purely reactive path. b.) *Clementine* opportunistically turning.

CONTINUOUS REPLANNING

This approach eliminates subgoal obsession since the robot can change “optimal” paths reactively and opportunistically move to a closer waypoint. As with most things in life, too much of a good thing is bad. At some point though, the sheer number of unmodeled obstacles might force the robot to get trapped or wander about, changing subgoals but making no real progress. The D^* solution to this problem is to continuously update the map and dynamically repair the A^* paths affected by the changes in the map. D^* represents one extreme on the replanning scale: *continuous replanning*.

Continuous replanning has two disadvantages. First, it may be too computationally expensive to be practical for a robot with an embedded processor and memory limitations, such as a planetary rover. Second, continuous replanning is highly dependent on the sensing quality. If the robot senses an unmodeled obstacle at time T_1 , it computes a new path and makes a large course correction. If the robot no longer senses that obstacle at time T_2 , because the first reading was a phantom from sensor noise, it will recompute another large course correction. The result can be a robot which has very jerky motions and actually takes longer to reach the goal.

EVENT-DRIVEN REPLANNING

In the cases of path planning with embedded processors and noisy sensors, it would be desirable to have some sort of *event-driven* scheme, where an event noticeable by the reactive system would trigger replanning. Trulla uses the dot-product of the intended path vector and the actual path vector. When the actual path deviates by 90° or more, the dot product of the path vector and the actual vector the robot is following becomes 0 or negative. Therefore the dot product acts as an affordance for triggering replanning: the robot does not have to know why it is drifting off course, only that it has drifted noticeably off course.

This is very good for situations that would interfere with making progress on the originally computed path, in effect, situations where the real world is less amenable to reaching the intended goal.

But it does not handle the situation where the real world is actually friendlier. In [figure 14.14](#), an obstacle thought to be there really is not. The robot could achieve a significant savings in navigation by opportunistically going through the gap.

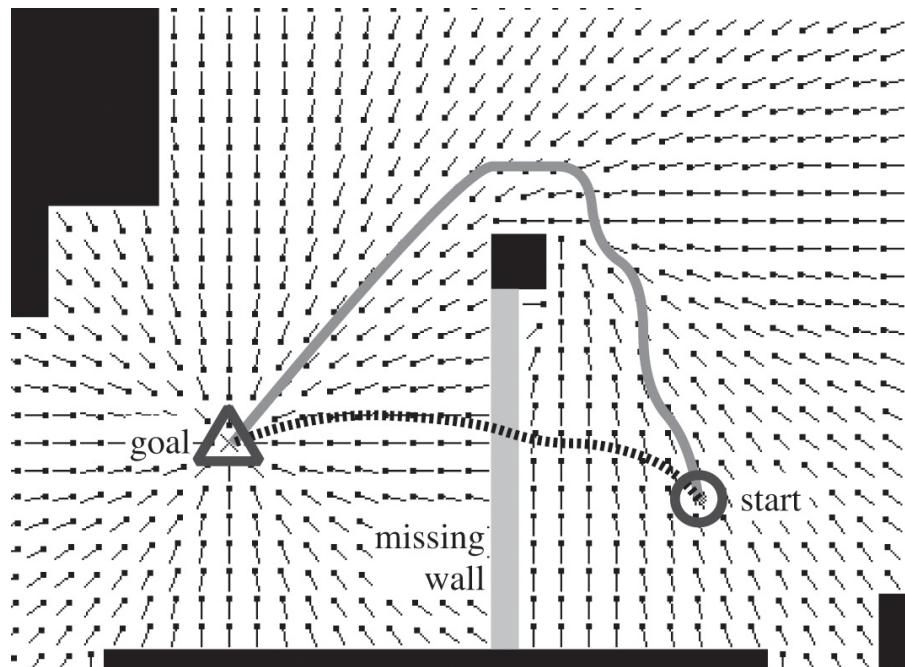


Figure 14.14 Opportunity to improve path. The gray line is the actual path, while the dashed line represents a more desirable path.

Such opportunism requires the robot to notice that the world is really more favorable than originally modeled. A continuous replanner, such as D^* , has a distinct advantage since it will automatically notice the change in the world and respond appropriately, whereas Trulla will not notice the favorable change because it will not lead to a path deviation. It is an open research question whether there are affordances for noticing favorable changes in the world that allow the robot to optimize its path opportunistically.

14.6 Motion Planning

Motion planning stems from industrial manipulator branch of robotics where the dynamics of robot cannot be abstracted away by assumptions of holonomicity and the shape of the robot—or the part it is carrying—cannot be ignored. As an example of why a robot may not be holonomic, consider if a robot is carrying a 225kg load of parts from one side of the factory to another at a high speed. It may need to slow down and take a wider turn, or risk tipping over. As an example of why the shape is important, consider that the robot might need to move an arm through an opening and then lift it up and around obstacles in order to assemble a part. In these situations where the planning space has a high number of

dimensions to consider, traditional route-generation path planning methods do not work and thus fall into a field of inquiry called motion planning. Motion planning to incorporate vehicle dynamics is generally treated as a controls problem that modifies path planning and execution, so it is not covered in this book. However, motion planning that incorporates different poses and constraints is often addressed in artificial intelligence and will be discussed below.

PIANO MOVER'S PROBLEM

The class of applications where pose cannot be abstracted away is often referred to as the *piano mover's problem* because moving a robot through an extremely cluttered, 3D environment is analogous to a group of movers planning how to rotate and flip the piano at each step in order to get a piano through a door, out of an apartment, down a flight of stairs, and through another door, and so on. The piano mover analogy should make it clear that the pose of the robot at any point in the path is important, and thus the robot cannot be treated as a round dot nor the configuration space reduced to three degrees of freedom. It should also illustrate that the possible combinations of locations and poses are essentially infinite. Fortunately, the piano mover's problem becomes tractable if the goal is simply to generate a collision-free plan, but not necessarily an optimal one. After all, the goal is to get the piano out of the apartment and down the stairs undamaged, not to do it optimally or quickly. The generally accepted method for generating a collision-free path is to use a variant of the rapidly evolving random tree algorithm.

More formally stated, in the piano mover's problem, motion planning must consider the location and the pose (*location, pose*) of the robot at each point in space and also whether it is possible to translate and rotate to move from $(location_i, pose_i)$ to $(location_{i+1}, pose_{i+1})$ without a collision. One way to look at motion planning is to think of every point in space as having a set of possible valid poses that the robot could be in without touching a wall or object. Planning a path must consider all of the possible poses, not just the location. Clearly, the search space is very large and needs to be abstracted into something more computationally tractable. Discretizing the world into a 3D regular grid or octree helps reduce the search space but each element has a large number of poses. Then there is the matter of transitions: if the robot is at $(location_a, pose_a)$ there may be only five poses it can transform into at $(location_b, pose_b)$, but if the robot is at $(location_c, pose_c)$, there may be an infinite set of valid poses at $location_b$.

RAPIDLY EXPLORING RANDOM TREES (RRT)

A strikingly different approach, called *rapidly exploring random trees (RRT)* based on the work of Steven LaValle, uses randomness to sample the space and poses. [Figure 14.15](#) illustrates the random tree generated by one variant called RRT-Connect and the partial path computed through the tree.

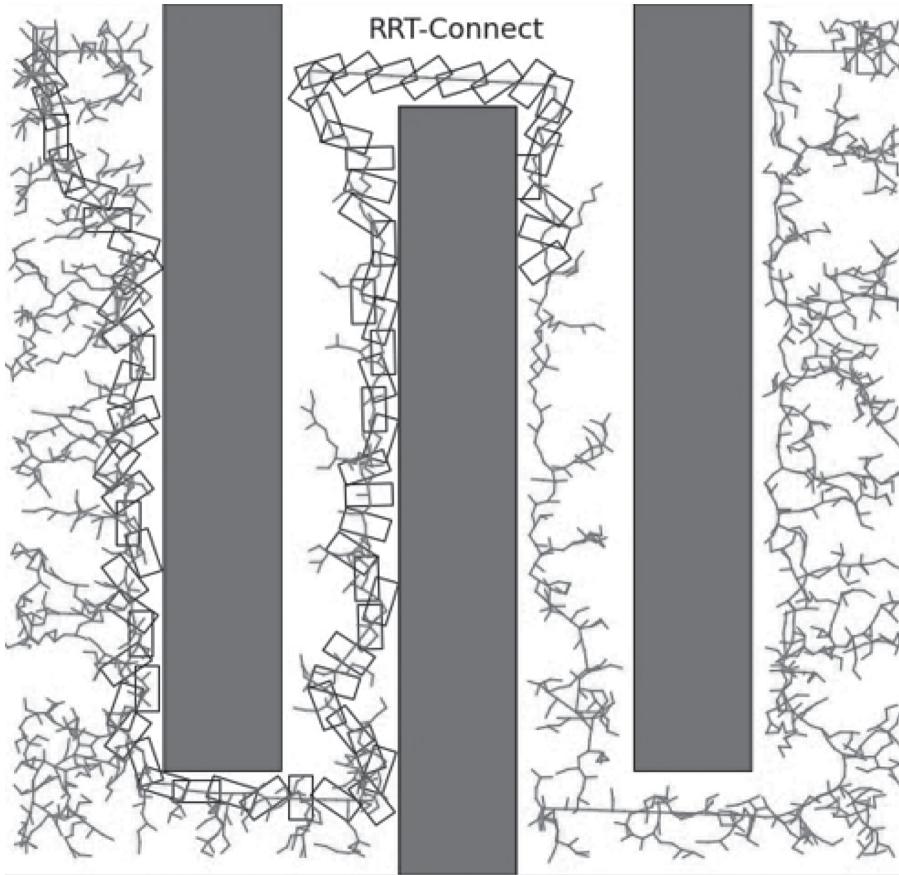


Figure 14.15 A random tree generated by the RRT-Connect algorithm and a partial path through the tree. Courtesy of Dmitry Trifonov.

The basic algorithm is as follows:

- Start with an a priori map and a starting node (location,pose) which serves as the root of the tree.
- Randomly sample the space and generate a list of candidate (location, pose) nodes.
- Randomly pick a candidate node from the list and check to see if the node contains a valid pose where the robot or part is not inside a wall or object. If the node is not valid, repeat until you find a valid node. Note that the methods only check to see if the randomly selected (location, pose) is valid, they do not search for all valid poses at the location.
- For the valid candidate node, check to see if there is a collision-free path between the nearest vertex in the tree and the node, that is, can the robot physically translate and rotate from the vertex in the tree to the candidate node without hitting anything. If so, add the node to the tree. If not, discard the candidate node and pick another.
- Continue until the tree includes the goal.
- Once the tree is built, then plan a path through the tree.

14.7 Criteria for Evaluating Path and Motion Planners

At this point, it should be clear that a robotics designer has a wide variety of techniques to choose from, spanning almost 30 years of research. As with everything else robotic, the choice of technique depends on the ecological niche the robot will operate in. The minimum criteria for evaluating the suitability of a path or motion planner is:

1. *Complexity.* Is the algorithm too computationally or space-intensive to execute or reside within the limitations of the robot? Many UAV planners run on a laptop before the flight because there is no onboard computer, or the onboard computing is too limited.
2. *Sufficiently represents the terrain.* Many researchers work in indoor environments which are flat. Outdoor robots may be operating in rough terrain with steep inclines or undesirable areas, such as slippery sand or mud. If the path planning algorithm is built to generate paths from a binary representation (a region is either navigable or not), then it may lead the robot into trouble if it is applied to a more diverse environment.
3. *Sufficiently represents the physical limitations of the robot platform.* Robots have physical limitations. The most profound limitation which impacts path planning is whether a robot is holonomic or not. Recall that holonomic robots can turn in place, and velocity can change instantaneously.
4. *Compatible with the reactive layer.* Path planners are deliberative by definition, but the reactive layer will be responsible for executing the path. A technique which simplifies this transformation into executable actions is desirable.
5. *Supports corrections to the map and replanning.* Path and motion planning requires an a priori map which may turn out to be seriously wrong. For example, the Quince robot, used at Fukushima Daiichi nuclear accident, was sent to climb stairs to the third floor of a reactor building; however, it discovered that the stairway landings were narrower than shown on the as-built drawings, and, thus, could not turn.¹⁵⁷ The robot had to abandon the mission and back down the stairs. Therefore, a robot may start out with one map, discover the map is incorrect, and need to update the map and replan. Clearly techniques like D^* , which permit the existing plan to be repaired rather than be scrapped and computed from scratch, are desirable.

14.8 Summary

Metric path planning converts the world space into a configuration space, or *Cspace*, representation that facilitates optimal path planning of a route. There are many different ways to represent an area or volume of space, but all convert space to a “bird’s-eye” view, independent of the position and viewpoint of the robot. Cspace representations typically result in a graph or tree suitable for an A^* search. Regular grids are currently the most popular Cspace representation in practice, though generalized Voronoi graphs have interesting properties. A^* methods generally assume that the robot is holonomic, and Cspace methods that grow obstacles to the size of the robot help make this assumption viable. A Cspace method may lead to discretization errors and thus to paths with spurious turns; these can be eliminated with a path relaxation or string-tightening algorithm.

Metric methods often ignore the issue of how to execute a planned path, especially with regard to the impact of sensor noise or uncertainty in localization. Two problems in interleaving metric path planning with reactive execution are subgoal obsession and when to replan. Optimal path planning techniques for a priori fixed maps are well-understood, but it is less clear how to update or repair the path(s) without starting over if the robot encounters a significant deviation from the a priori map. One solution captured by the D^* algorithm is to replan continuously if resources permit and sensor reliability is high; another is event-driven replanning which uses affordances to detect when to replan.

A possibly even more serious omission of the constraints of physical world on path planning is that popular route-generation path planners apply only to holonomic vehicles operating in relatively open spaces. Instead, motion planning examines how to plan movements of a robot through a very cluttered, high-dimensional space where vehicle dynamics and pose matter. The rapidly exploring random tree class of algorithms is popular for solving the piano mover's problem.

In terms of artificial intelligence, path and motion planning are actually search problems where the objective is to find an answer efficiently in the haystack of a priori knowledge. In this case, the answer is a sequence of waypoints or movements that can be connected with a tree or graph. This type of graph planning has a different flavor than the classic planning and problem solving machinations seen in Strips. While motion planning uses randomness, it is not inference. Recall that inference uses deliberation to add missing information, supply a missing relationship between data sets or concepts, or infer new knowledge. Motion planning uses random sampling to help with a computationally daunting search through what could be explicitly represented, not to bridge the gap between the known and the unknown.

Returning to the questions posed in the introduction, the answer to: *What is the difference between topological navigation and metric navigation or path planning?* is that topological navigation focuses on stored, sensed routes, while metric methods rely on using a priori maps to generate optimal paths or sequences of movements. The chapter described several different configuration spaces and algorithms driving the practical question: *What is commonly used or works well enough?* In practice, the most common route-generation algorithms are an A^* or D^* variant which consider the actual nonholonomic characteristics of a robot and its velocity and trajectory. These algorithms typically use a regular grid configuration space. For motion planning, the RRT class of algorithms is popular. Finally, *How much path planning do you need?* has many possible answers. Certainly one important consideration is whether the dynamics or pose of the robot has to be considered. If the robot can be approximated as holonomic and it works in relatively open spaces like roads and hallways, an A^* algorithm is sufficient for route planning. If not, motion planning is needed. A second consideration is whether the desired output is a route to a destination or a path that maximizes sensor coverage of an area. A third consideration is when replanning occurs. Moore's Law has led to situations where it is now computationally possible to replan at each step. This can enable the path planner to detect and plan around, rather than react to, obstacles or changes in the world. However a designer may want to preserve the distinction between deliberation and reaction and thus interleaving planning and execution may be more elegant.

Metric path and motion planning depend on having a map of the world and that a robot can localize itself to that map as it executes the plan. The next chapter covers simultaneous localization and mapping. Cspace representations and algorithms often do not consider how to represent and reason

about terrain types, and special cases such as the robot actually conserving or generating energy going down hill are usually ignored. Chapter 15 covers terrain.

14.9 Exercises

Exercise 14.1

Define:

- a. Cspace
- b. path relaxation
- c. digitization bias
- d. subgoal obsession
- e. termination condition.

Exercise 14.2

Represent your indoor environment as a GVG, regular grid, and quadtree.

Exercise 14.3

Represent your indoor environment as a regular grid with a 10 cm scale. Write down the rule you use to decide how to mark a grid element empty or occupied when there is a small portion of an obstacle or wall overlapping it.

Exercise 14.4

Consider a regular grid of size 20 by 20. How many edges will a graph have if the neighbors are:

- a. 4-connected?
- b. 8-connected?

Exercise 14.5

Convert a meadow map into a graph using:

- a. the midpoints of the open boundaries, and
- b. the midpoints plus the two endpoints.

Draw a path from *A* to *B* on both graphs. Describe the differences.

Exercise 14.6

Create a generalized Voronoi graph (GVG) of the indoor environment in figure 14.16.



Figure 14.16 GVG exercise.

Exercise 14.7

What is a heuristic function?

Exercise 14.8

Use the A^* algorithm to plan the path from E to J in [figure 14.17](#).

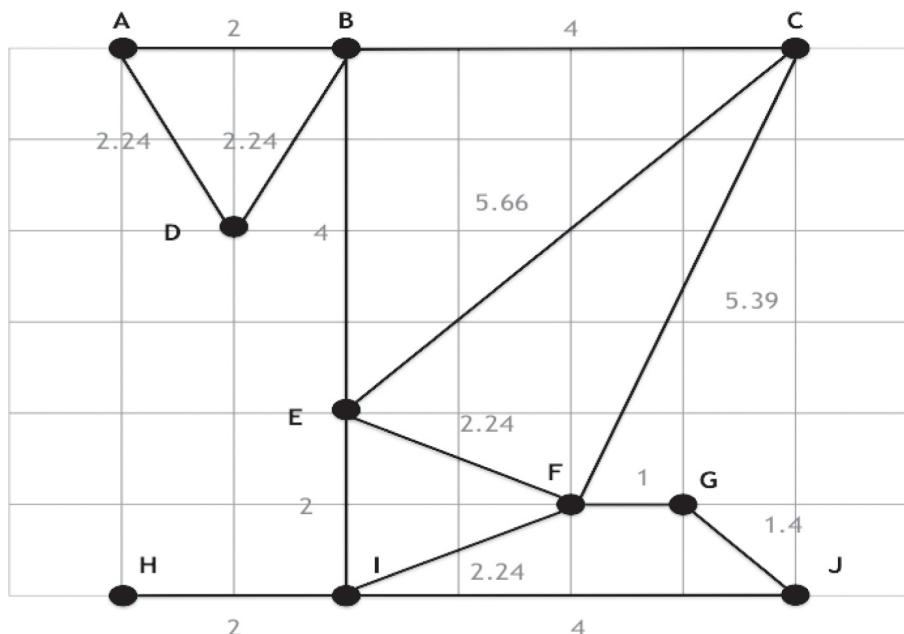


Figure 14.17 A^* exercise.

- a. In order to ensure that your application of A^* can be followed, give the formula for the A^* evaluation function and explain each term.
- b. Show each step and the value of each node evaluated. If there is a tie, choose the node with lowest letter in the alphabet.

Exercise 14.9

Apply waveform propagation to a regular grid.

Exercise 14.10

Describe the piano mover's problem and why it is different than route planning.

Exercise 14.11

Describe RRT types of planning algorithms and what they are used for.

Exercise 14.12

Explain the similarities and differences between RRT and A^* types of algorithms.

Exercise 14.13

List the criteria for evaluating a path planner.

Exercise 14.14

Consider a UAV path planner for covering an area that works as follows: The user first specifies a polygon, such as a field, on a map that the UAV should cover. The UAV then computes a set of waypoints to cover the area to give a path through the polygon. The path guarantees that the UAV will pass over every spot inside the polygon at least twice in case there is positional error. Use the criteria for evaluating a path planner to rate the planner's utility.

Exercise 14.15

Subgoal obsession has been described as a problem for metric planners. Can hybrid systems exhibit subgoal obsession if they use topological planning?

Exercise 14.16

Trulla uses a dot product of 0 or less to trigger replanning, which corresponds to 90° from the desired path. What are the advantages or disadvantages of 90° ? What would happen if Trulla used 45° or 135° ?

Exercise 14.17

Describe the difference between *continuous* and *event-driven replanning*. Which would be more appropriate for a planetary rover? Justify your answer.

[*Programming*]

Exercise 14.18

Program an A^* path planner. Compare the results to the results for the Dijkstra's single source shortest path program from chapter 13.

14.10 End Notes

For a roboticist's bookshelf.

Robot Motion Planning by Jean-Claude Latombe,¹¹³ Stanford University, is the grandfather of books dealing with configuration space, but *Principles of Robot Motion: Theory, Algorithms, and Implementations* by Howie Choset, Kevin Lynch, Seth Hutchinson, George Kantor, Wolfram Burgard, Lydia Kavraki, and Sebastian Thrun⁴³ is the indispensable reference.

Voronoi diagrams.

Voronoi diagrams may be the oldest Cspace representation. *Computational Geometry: Theory and Applications*⁵⁸ reports that the principles were first uncovered in 1850, although it was not until 1908 that Voronoi wrote about them, thereby lending his name to the diagram.

On creative heuristic functions for path planning.

Kevin Gifford and George Morgenthaler have explored some possible formulations of a heuristic function for path planning over different terrain types. Gifford also developed an algorithm which can consider the energy savings or capture associated with going downhill.⁸³

People really use A.*

In my travels for US Department of Defense boards, I have met many pilots. One group had taught themselves C++ so that they could program the A* algorithm on their personal laptops and bring them along for planning flight paths. Their cost functions were more complex than the typical Euclidean distance heuristic, combining distance and wind into an admissible fuel cost.

15

Localization, Mapping, and Exploration

Chapter Objectives:

- Define the *pose* of a mobile robot.
- Explain the difference between *localization*, *mapping*, and *exploration*.
- Define each term in: $bel(\mathbf{x}_t) = f(bel(\mathbf{x}_{t-1}), \mathbf{u}_t, \mathbf{z}_t, m)$.
- Discuss the difference between the *Markov*, *Extended Kalman Filter (EKF)*, *grid-based*, and *Monte Carlo* localization algorithms.
- Describe the difference between localization and *simultaneous localization and mapping (SLAM)*.
- Define the *loop closure* problem.
- Describe the difference between *digital terrain elevation data (DTED)* maps, *digital elevation model (DEM)* maps, *digital surface maps (DSM)*, and an *orthomosaic* image.
- Give at least one example of two types of methods for *proprioceptive terrain identification* and *exteroceptive terrain identification* and discuss the limitations of each.
- List and explain the five attributes of *traversability*: *verticality*, *surface properties*, *tortuosity*, *severity of obstacles*, and *accessibility elements*.
- Compare and contrast *frontier-based* and *generalized Voronoi graph (GVG)* based exploration.

15.1 Overview

The previous chapters have covered two of the four fundamental questions in navigation: *Where am I going?* or the mission planning problem and *What is the best way there?* or the path planning problem. Two other questions remain: *Where am I?* or localization problem and *Where have I been?* or mapping problem. This chapter will cover localization and mapping plus algorithms for the related topic of exploration.

On the surface, localization and mapping appear to be problems that are solved. People unfamiliar with AI robotics often ask: *Why is this a problem? Can't you just use GPS, some sort of RFID beacons, or 3D scans from RGB-D sensors?* Indeed GPS provides accurate position information outdoors, and

indoor environments can be engineered with RFID or other mechanisms to track a robot. These mechanisms for tracking a robot are not the same as those mapping the world along that track. Generally a map contains more information about the larger context than just the location of the robot. A “map” of a mine might mean a digital elevation map that can be used to determine how material has been removed. A “map” of an office building might mean a 2D floor plan that shows all the doors and exits for emergency evacuation preparedness. A “map” of the underwater portion of a bridge might have a 3D model of the pilings and the sea floor.

Roboticists ask more specific questions. One is: *How can you simultaneously map the world and be sure where you are?* In other words, what are the specific techniques for creating a map of the world in spite of any errors incurred while moving. Another question is: *How do you explore new areas efficiently or at least consistently?* A more subtle question for artificial intelligence is: *How do you label the map with objects and features or terrain?*

The chapter begins with localization as the first step in mapping. It then moves to simultaneous localization and mapping, or SLAM, techniques. SLAM has been used primarily in or around buildings or urban structures; therefore a separate section is devoted to outdoor mapping. One aspect of outdoor mapping is generally concerned with understanding a priori terrain maps, with another aspect concerned with generating terrain maps. Both abilities are essential for outdoor navigation. Another aspect of outdoor mapping, such as for geospatial or agricultural applications, relies on stereophotogrammetry. In ground robotics, mapping is used to plan paths or react to obstacles. The success of a robot depends on whether the world is traversable. Thus a section is devoted to that topic. Next, the chapter discusses the pragmatics of exploring the unknown. The chapter relates localization, mapping, and exploration to topics in artificial intelligence, especially semantic world knowledge, before concluding with a revisit to the common questions.

15.2 Localization

LOCALIZATION

POSE

Thrun, Burgard, and Fox define mobile robot *localization* as “the problem of determining the pose of a robot relative to a given map of the environment.”²⁰⁷ Recall that the *pose* of a robot is its position plus its orientation. For two dimensions, the pose is $\mathbf{x} = (x, y, \theta)^T$.

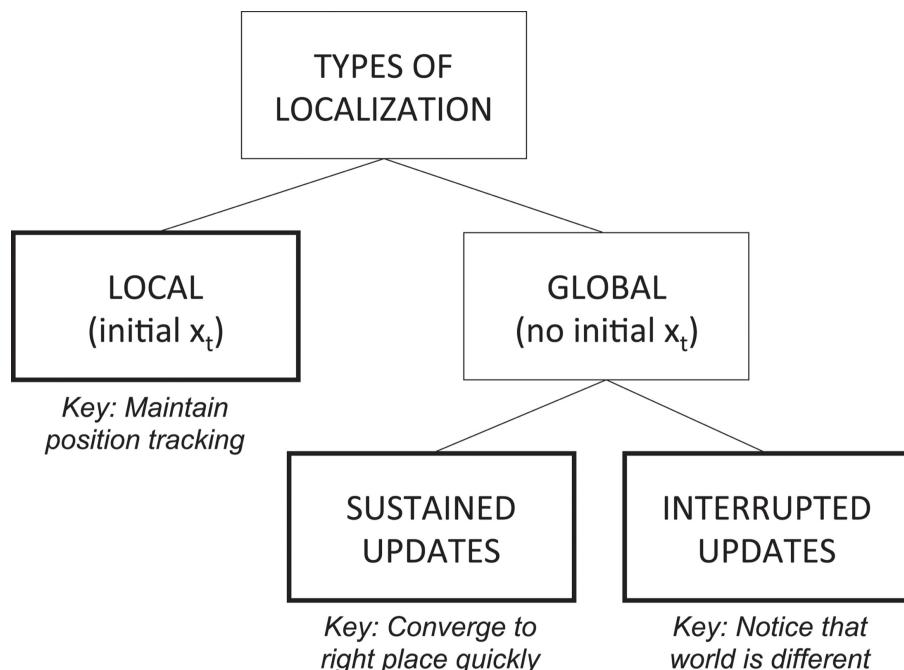
Mobile robot localization is sometimes called *position estimation* but as the orientation of the robot is important in most cases, localization is the more complete term. Note that localization assumes that there is an a priori map and the robot is localizing itself relative to it. Localization is hard for a variety of reasons described below. To make the problem even harder, there are three general types of localization. Each of the types uses a different class of algorithms that will be detailed in subsequent sections.

Thrun, Burgard, and Fox²⁰⁷ give a list of reasons why localization is challenging:

- It requires a model that incorporates the robot’s actuators, its sensors, and the accuracy and repeatability of those sensors in different environments.
- There may be sensor noise, either proprioceptive and exteroceptive, which introduces uncertainty.

- Methods for localization are computationally expensive.
- In some localization problems, the robot may not know its initial position.
- The robot may be operating in a dynamic work envelope where objects move, causing the robot to have to consider whether sensor readings are a result of the robot moving or *Object A* moving.
- The path or task may not support localization. For example, if the robot travels across a large, open, featureless warehouse, it may not be able to sense any features to localize against.

There are three types of localization problems which have a different flavor. [Figure 15.1](#) shows the relationship between the three. In local localization, the robot starts with an initial x_t , perhaps explicitly entered at the start of a mission, and the challenge becomes how to maintain localization from that time forward. Generally, local localization is about tracking the robot's position and orientation. The global localization problem occurs when the robot does not have an initial x_t . Regardless of why the robot does not have an initial x_t , the problem is further subdivided into whether the robot can count on getting sustained sensor updates or if those updates may be interrupted. An example of an interrupted update is when the robot is turned off and is moved to a different part of a building without anyone explicitly resetting where it is now; this is humorously referred to as the “kidnapped robot” problem. In the global localization with sustained updates problem, the goal is to create algorithms that allow the robot to quickly converge on its correct location. In the global localization with interrupted updates problem, the challenge is for the robot to notice that the world has changed, and, as a result of that change, it should restart global localization rather than accept an erroneous localization.



[Figure 15.1](#) Three types of localization.

The three types of localization problems are not the same as *localization algorithms*. Localization algorithms are variants of Bayes Filter algorithms which use Bayesian probabilistic methods to estimate the probable location. These algorithms typically are categorized by their approach—either they localize the robot relative to features (feature-based localization) or they localize by matching raw sensor readings (iconic localization). The algorithms within a category can be used for any of the three localization problems, though some algorithms are more suitable for specific types. In practice, Monte Carlo localization methods dominate localization, both for feature-based and iconic categories of input.

15.3 Feature-Based Localization

Feature-based localization extracts features from raw data and matches those features to the map. These features might be corners, walls, doors, or whatever can be both perceived and identified on a map. The advantage of feature-based localization is that it abstracts the world into a small set of features, and thus it should take less computation time to match the features being sensed to corresponding features on the map. The disadvantage is that it is hard to extract features reliably.

MARKOV LOCALIZATION

The most common feature-based localization is *Markov localization*. Markov localization computes the belief in each of the possible poses, with the highest belief being the most likely pose. The belief is a function,

$$bel(\mathbf{x}_t) = f(bel(\mathbf{x}_{t-1}), \mathbf{u}_t, \mathbf{z}_t, m)$$

where

- \mathbf{x} : is the set of possible poses; in practice the space may be discretized
- $bel(\mathbf{x}_t)$: is the belief that the robot is at \mathbf{x} at time t
- $bel(\mathbf{x}_{t-1})$: is the belief that the robot was at \mathbf{x}_{t-1} in the previous time step
- \mathbf{u}_t : is the set of control actions or what movements the robot is commanded to execute at time t
- \mathbf{z}_t : is the set of measurements or what the robot observed at time t
- m : is the map.

Markov localization is attractive for global localization problems because it should converge to the correct pose fairly quickly. To conceptualize this, consider [figure 15.2](#). The robot starts in unknown pose, \mathbf{x}_{t1} . It is able to observe

a door on its left, \mathbf{z}_{t1} , which it can match to the map m ([figure 15.2a](#)). This results in $bel(\mathbf{x})$ where x is the set of four equally probable poses, shown in [figure 15.2b](#). The robot moves forward, \mathbf{u}_{t2} and updates sensing \mathbf{z}_{t2} . It now sees a door on its right ([figure 15.2c](#)). Matching this feature to the map results in two highly probable poses and two less likely poses, shown in [figure 15.2d](#). Belief in x_3 may not go to zero because of sensor uncertainty. The robot moves forward and updates sensing again and can now detect Corner 1. The belief in the robot being at x_1 at t_3 increases while $bel(x_2)$ diminishes and $bel(x_4)$ may be zero. $bel(x_3)$ may have increased because it is possible to detect a door on the left and a corner and miss

sensing the door on the right.

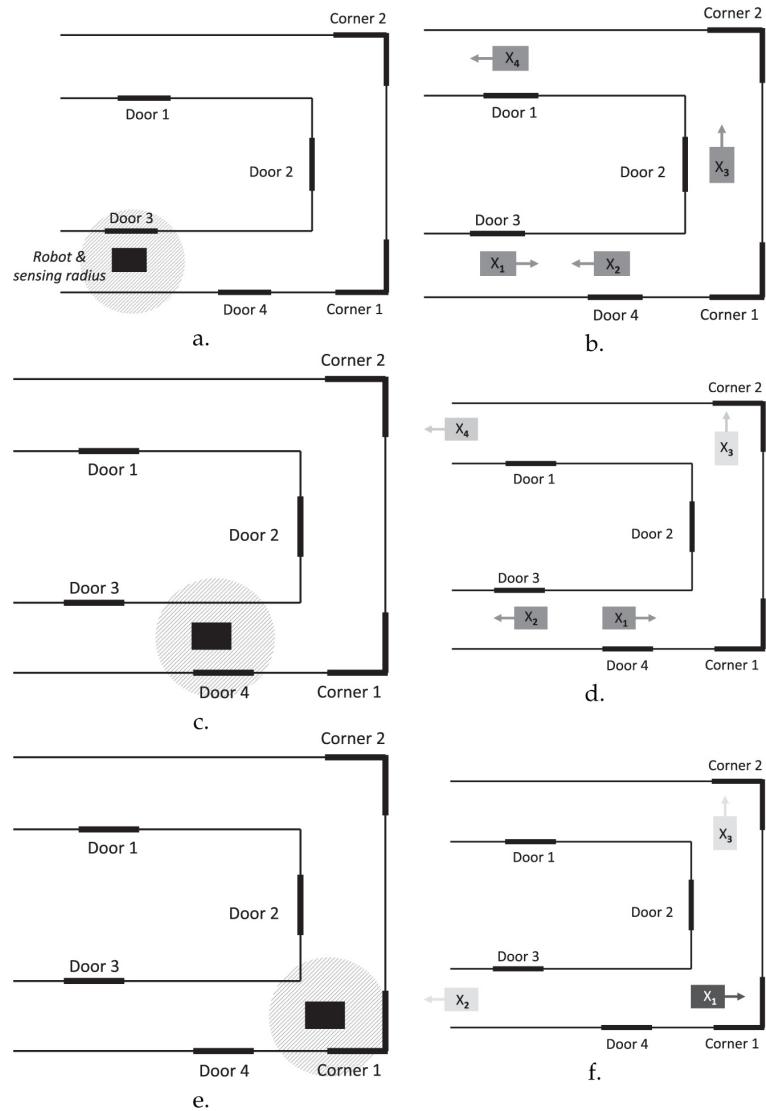


Figure 15.2 Markov global localization example. a.) Robot senses unknown location, b.) resulting in four equally probable poses, c.) moves forward and updates sensing, d.) resulting in two highly probable poses and two less likely poses, e.) moves forward and updates sensing again, f.) resulting in a set of three possible poses, one of which is clearly more probable.

EXTENDED KALMAN FILTER (EKF)

An *Extended Kalman Filter (EKF)* is often used for local localization. EKF algorithms are designed to predict what the robot will sense at the next time step, given the control action. It then takes the difference between the prediction and what it actually sensed to correct or fine-tune its estimates. For

example, the robot may be drifting to the left a bit instead of moving straight forward, and the EKF would eventually adjust for that. In this case, the EKF would extract a set of observations about the features that it perceived, \mathbf{z} . It would then match those features to the map; these features are called *correspondence variables*. The more accurately the robot can fine-tune its predictions about how it is moving, the more certain it can localize its position relative to the features.

15.4 Iconic Localization

Iconic localization uses raw sensor readings to match actual observations to expected observations if the robot was at a particular location. In the previous Markov localization example, the robot was matching only six features, four doors and two corners. There might be hundreds of sensor readings from a LIDAR alone, presenting a massive computation challenge. There are two common iconic methods for localization: grid-based methods and Monte Carlo Localization.

GRID-BASED LOCALIZATION

In *grid-based localization*, the sensed world is partitioned into a tessellation of convex polygons. The algorithm then computes the likelihood of all possible poses within each polygon given the observation. The advantage is that the grid acts to reduce the computational complexity by discretizing the space. In practice, grid-based localization algorithms often further reduce complexity by restricting matching to a local “sub-map.” The disadvantage is that the sub-map approach works only for local localization where the robot always knows what general region of the map it is in.

MONTE CARLO LOCALIZATION (MCL)

Monte Carlo Localization (MCL) is similar to the Monte Carlo methods for RTT and other applications. Particles, or sample poses, are scattered through space. Then the algorithm computes the belief that the robot is at the pose given the observations. In the next time step, more particles are added, and some particles “die” if they have a low probability. Technically Monte Carlo localization is not restricted to use with raw sensor observations, but it is typically used this way.

15.5 Static versus Dynamic Environments

STATE AUGMENTATION

OUTLIER REJECTION

Up to this point, localization has been described as if the world were a static place. Unfortunately, the robot may be working in human-occupied areas where people move about unpredictably or at construction sites or mines where the structure of the environment changes and maps are not up to date. Most algorithms break due to the dynamic changes in the world. The changes often exceed what can be accommodated by sensor noise. They also effectively add a “hidden state” to the localization process, representing an estimate of what the robot should observe at the next time step. There are two fundamental techniques for trying to cope with localizing in a dynamic environment: state augmentation and outlier rejection. In *state augmentation* the goal is to reveal the hidden state, that is, to estimate the impact of people on the observations. There has been some computer vision research in estimating crowd flows. In *outlier rejection*, the goal is to use knowledge about the sensors and the

environment to eliminate suspicious readings. For example, the robot might discard surprisingly short-range readings, which might be people, if it thinks it is in a wide hallway. The worst that would happen by eliminating the outliers is that it would take longer to estimate the pose from the remaining observations. In a very crowded hallway, the robot may not have enough measurements to generate belief.

15.6 Simultaneous Localization and Mapping

MAPPING

SIMULTANEOUS LOCALIZATION AND MAPPING (SLAM)

Mapping answers the question: *Where have I been?* with a representation of the environment. Mapping an unknown environment obviously means that there is not a map, but without an a priori map, the robot cannot localize itself. Thus, in practice, mapping means that the robot has to build a map and simultaneously localize itself to that map as it goes; this is called *simultaneous localization and mapping (SLAM)*. In general, the path that the robot has taken through the world is of interest.

Ideally, the robot would observe the world (\mathbf{z}_{t1}), move a small amount (\mathbf{u}_{t2}) such that the next set of observations \mathbf{z}_{t2} would overlap \mathbf{z}_{t1} , use the overlap to merge \mathbf{z}_{t1} and \mathbf{z}_{t2} into a map, and then localize itself relative to the new map. The new map represents the maximum likelihood of the world, that is, the map represents the highest probability of what the world really is. The problem is that the robot may not move precisely as expected, and thus there are multiple possible poses and thus multiple possible maps.

RAO-BLACKWELLIZED FILTERING

Rao-Blackwellized Filtering is the dominant method in SLAM for computing the maximum likelihood map. As with Monte Carlo Localization, Rao-Blackwellized Filtering uses particle filtering. In this case, each particle represents a path and a local map instead of only one version of the map. After each observation, the algorithm updates only the sensed area of the maps and computes the belief in each particle, discarding unlikely particles and increasing the numbers of particles for promising samples. An interesting computational note is that the algorithm uses a tree to save all the particles that form the history of the current set of particles.

LOOP CLOSURE

SLAM is generally reliable in a local region, but it is possible that the map has errors over a larger region. As a result, a robot can go around a set of connecting halls and, due to incremental errors in mapping, either not realize that it has returned to the starting point or has produced a map that shows a physical misalignment between the start of the map and the end of the map; this is essentially the same problem that was reported for odometry. Therefore, an additional method has to be executed to check and adjust the map if needed. The mechanism by which a robot notices that it has returned to a previously visited place is called *loop closure*. This method requires additional sensing of more features, a different sensor or hybrid feature-sensor combinations, or additional statistical processing of expectation matching. Fortunately, closing the loop provides a better path estimate, and that path will propagate backwards and improve the overall map and localization.

There are four main approaches to resolving the loop closure problem:

- *Feature matching*: Match the geometric features within the map. This process is similar in spirit to matching distinctive places to a topological map (chapter 13). Like distinctive places, the robot may encounter problems if the environment is homogeneous, for example, just corners and doors.
- *Sensor scan matching*: Does the sensor scan match a *complete* profile of the area? In this approach, instead of matching sensor readings only in front of the robot or in the direction of travel, the robot periodically considers all sensor readings and matches the readings to a larger region. Sensor scan matching may still produce errors if the environment is homogeneous because there is nothing to match against.
- *Hybrid feature-scan matching*: Do visible features and the range readings have the same profile? This is a better solution because it fuses multi-modal sensor readings.
- *Expectation matching*: Is it statistically likely that the robot has returned to the starting location? This is less error-prone and does not depend on multiple sensor modalities, but the method is computationally expensive.

15.7 Terrain Identification and Mapping

The discussion of localization and mapping has generally involved indoor settings or outdoor settings near urban structures, such as a ground vehicle mapping a city, where the terrain is presumed horizontal and amenable to mobility. A ground vehicle traveling cross-country or roaming a planet needs to identify the terrain because the difference between a level hard surface or a sandy dune impacts its mobility. An unmanned aerial vehicle mapping a county or province to identify areas at risk of flooding works on a different scale to measure the elevation of the terrain. Therefore it is useful to understand terrain identification and mapping.

NATURAL TERRAIN

There is no precise definition of terrain or a catalog of what terrain features are important to a robot. However, the US Army has studied terrain extensively, especially in the context of ground vehicles, such as tanks, and their work⁵⁹ serves as a foundation for discussing identifying terrain features. The US Army presumes that a soldier is capable of identifying terrain given a terrain map. In the US Army's taxonomy,⁵⁹ terrain maps generally contain two types of information: *natural terrain* and *man-made features*. There are four key attributes of *natural terrain* information. The *surface configuration* represents the landforms, relief, and slope (or gradient) of the terrain. The terrain map also tries to capture the *vegetation features*, such as trees or bushes, *soil features*, and *water features*, which impact navigation. An outdoor space will also have man-made features. Man-made features can be those that simplify mobility (highways, railroads, bridges, etc.) or that could serve as features for localization (airports, cell towers, points of interest, etc.).

15.7.1 Digital Terrain Elevation Maps

DIGITAL TERRAIN ELEVATION DATA (DTED)

Roboticians generally have access to *digital terrain elevation data (DTED)* or maps showing the elevation of the ground. In the US, these are produced by National Imagery and Mapping Agency

(NIMA) and the maps have five levels of resolution. DTED 1 maps give the average elevation for patches of ground 100 meters square. DTED 2 maps are at 30 m resolution, DTED 3 at 10 m, DTED 4 at 3 m, and DTED 5 at 1 m. Lower resolutions of 4 cm are theoretically possible with data collected from UAVs flying at lower altitudes.

Digital terrain elevation maps are very useful for a Mission Planner or Cartographer to determine a safe route for a ground robot. A $10\text{ m} \times 10\text{ m}$ or $30\text{ m} \times 30\text{ m}$ resolution is usually sufficient for path planning, estimating transit time, and projecting mobility suitability and energy costs. DTED maps can enable a mission planner to route a robot so that it stays hidden behind a ridge line or avoids certain man-made features.

15.7.2 Terrain Identification

Digital terrain elevation maps are not sufficient for navigation because they do not show locations of trees, large rocks, slick muddy areas of low traction, and so forth. Surface properties of water, sand, and different soils can impact movement. Foliage can appear to be an obstacle when, instead, it could be safely run over. Another reason why they are not sufficient for navigation is that the maps may be outdated and, therefore, not reflect the true conditions.

Therefore, robots need algorithms for terrain identification to enable a robot to adapt to its environment in four ways. One, make *sensori-motor changes* where the parameters or gains of active behaviors change, for example, when accelerometers report bumpy movement, slow the robot down. Two, make *schematic changes*, where the set of active behaviors changes in some way, perhaps to substitute a different sensor better suited for navigating in foliage. Three, make *deliberative changes*, for example, machine learning may be used to associate which behaviors and sensors work best for the current situation. Finally, terrain identification may be *distributed* to other robots so that they can change and adapt their navigational strategy.

Terrain information typically comes from one of three sources. A terrain map can be constructed from satellite or high-altitude aircraft imagery and provided to the robot as an *a priori* map (such as a DTED map), but, as noted earlier, that information is not usually sufficient for safe navigation. The robot can get more recent or real-time information from another robot who is surveying the area ahead. More commonly, the robot can construct its own terrain map as it moves about.

TERRAIN IDENTIFICATION

The challenge for robots navigating complex terrain is whether they have the on-board sensing to perform *terrain identification*. Using *projective terrain identification*, the robot can identify the terrain before it actually traverses it, while *reactive terrain identification* is accomplished as the robot experiences the terrain.

Project terrain identification relies on exteroceptive sensors, such as cameras, multispectral imagers, and lidar, to see the terrain ahead of the robot. Researchers have concentrated on using color cameras,^{60:54} because every robot will have a camera. The majority of computer vision work on terrain identification has involved supervised learning. A common approach since the late 1990s is to detect changes in the texture in multispectral images that signify a change in terrain.²² Lidar (which is sometimes called LADAR or spelled as LIDAR) can return accurate depth maps and has been used since the early 2000s. Much of the work in terrain identification was abandoned because the specialized

sensors were too large, too heavy, or too expensive to be used in practice, but terrain identification is now being revisited due to advances in sensor miniaturization.

All exteroceptive methods continue to struggle with determining the difference between a rock and tall grass. Consider [figure 15.3](#). The ground robot treated the tall grass as if it were an obstacle. The unfiltered plot from the SICK laser range finder shows how blades of grass on the right of the robot were blocking the light and returning short ranges. Using just the unfiltered data indicated that the robot could move forward only through a narrow gap in the vegetation. The high resolution of the lidar actually made the problem harder because it could see individual blades of tall grass. However, a variant of median filtering was used to report the maximum reading over a window of readings that were bigger than a blade of grass, and this eliminated the thin profiles of the tall grass.

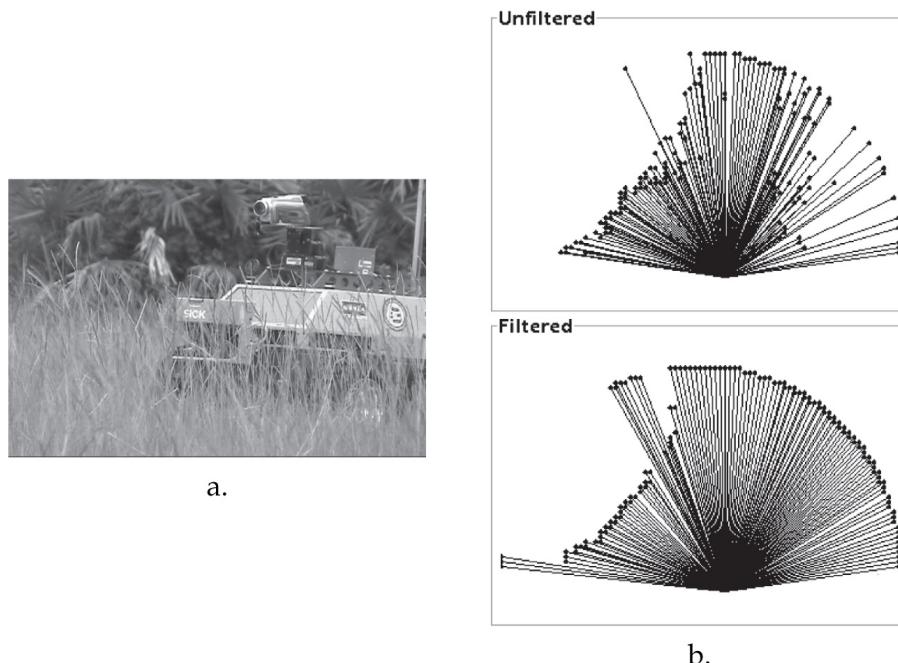


Figure 15.3 a.) An ATRV robot crossing a field with tall grass and b.) a comparison of the unfiltered and filtered laser range scans.

Reactive terrain identification relies on proprioceptive sensing to infer the terrain. Common methods use an inclinometer or other mechanism for detecting slope.^{93;112} Another method listens for changes in the pattern of wheel interaction with the terrain.^{111;112}

15.7.3 Stereophotogrammetry

Major advances have been made in stereophotogrammetry that allow photos from a robot to produce maps similar to, but with a higher resolution than, maps from satellite imagery. Stereophotogrammetry is becoming commonly used with small unmanned aerial vehicles, but the algorithms work for ground and marine vehicles as well.

SCALE-INVARIANT FEATURE TRANSFORM (SIFT)

The “stereo” in stereophotogrammetry refers to using images of the same location taken from two positions rather than with a stereo camera. Enough of the same location must be in both images, called the image overlap, so that a *scale-invariant feature transform (SIFT)* algorithm can find and match features in both images. If the pose of the camera is known for each image, for example, the GPS coordinates, altitude, and angle of the camera on a UAV, geometry can be used to extract distances.

ORTHOMOSAIC

In order to understand how stereophotogrammetry is used in robotics, some definitions are in order. The term *orthomosaic* usually refers to a high-resolution master image compiled from multiple images that have been tiled together, though the term is often used synonymously for all products of stereophotogrammetry. [Figure 15.4](#) is an example of an orthomosaic image.



Figure 15.4 An orthomosaic of a mudslide derived from approximately 20 images taken by a small unmanned aerial vehicle.

DIGITAL ELEVATION MODEL (DEM)

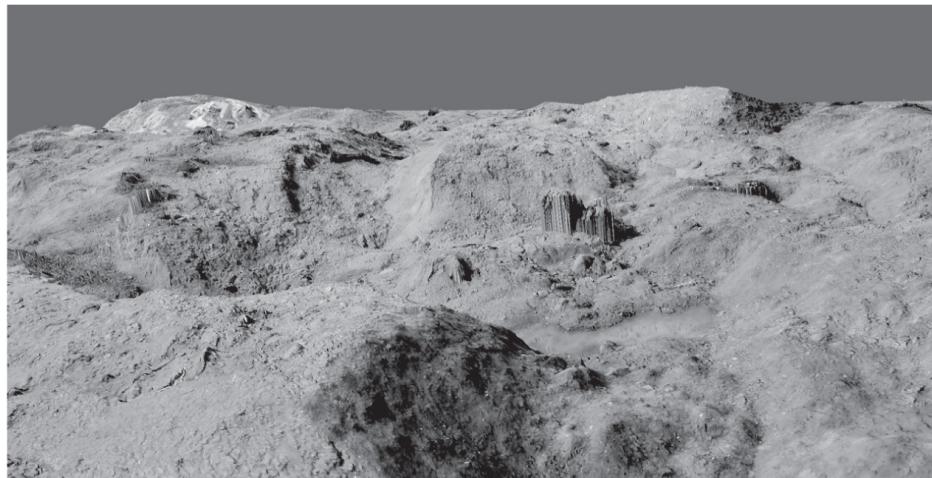
Another product of stereophotogrammetry is a *digital elevation model (DEM)* map. A DEM is equivalent to the military DTED map, and usually consists of a map of grids or pixels, where each grid element contains the average or maximum height of the patch of terrain covered by the grid element. DEMs are useful because they are independent of the vegetation and ground cover and thus capture the underlying structure. Because they are digital three dimensional representations, users can interact with the map as if in a video game. A user can zoom, rotate, and explore the map. In many programs, the user can click on a pixel to get the elevation reading at that point.

A *topographical map* is similar to a DEM and represents the contours of the terrain, both natural and man-made but without vegetation. The term generally refers to a paper map using two dimensional representations, such as concentric lines showing slices of earth with the same elevation.

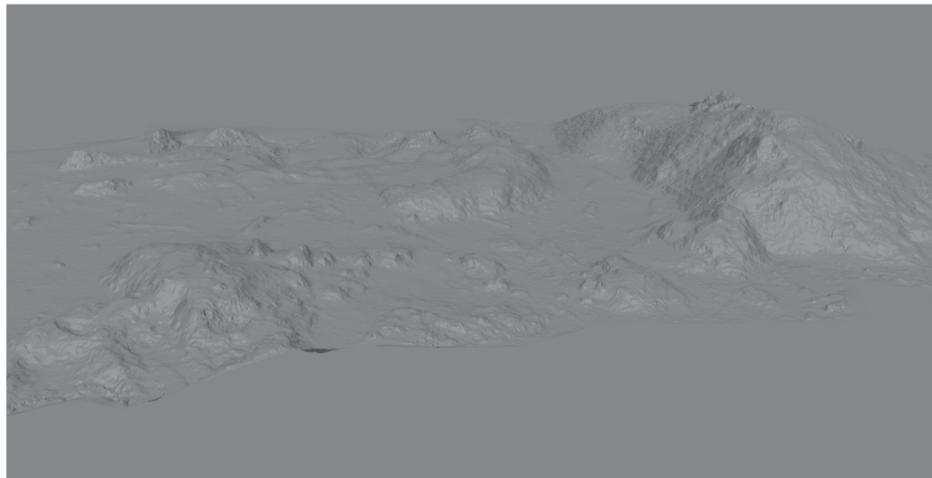
DIGITAL SURFACE MAPS (DSM)

Yet another product is *digital surface maps (DSM)*. DSMs are DEMs that include vegetation and ground cover. Unlike DEMs, each pixel in the DSM map has the elevation of the highest surface, not

the ground, for a grid area. DSMs are shown as the elevation with the video imagery superimposed (see [figure 15.5a](#)) or as a “naked” point cloud in [figure 15.5b](#). However, DSMs may not have the accuracy of a point cloud generated by a lidar or other form of direct range sensing.



a.



b.

Figure 15.5 a.) a digital surface map and b.) a point cloud of a mudslide derived from approximately 100 images taken by a small unmanned aerial vehicle.

An unmanned system can use stereophotogrammetry to produce DSMs. Software packages are commercially available for UAVs that produce photogrammetric products from geotagged images; the software is not executed in realtime but rather as postprocessing of a batch of images. Indeed, many UAV manufacturers and third-party applications provide flight path planning and coordinated camera

control algorithms to optimize image collection. However, stereophotogrammetry is far from perfect. The quality of stereophotogrammetry is heavily dependent on the data collected. Some rules of thumb for optimizing results are that flights of an area should be taken as close in time as possible so that changes in shadows and brightness do not change the features. Flights around noon reduce shadows and thus improve the product. Not all scenes can be processed successfully. For example, areas with running water often lead to blurry reconstructions. The elevation is consistent within the DSM map (i.e., local coordinates), but the map has to be anchored to surveyed points in order to generate accurate readings in absolute coordinates.

Stereophotogrammetric products are subject to at least four kinds of errors.¹⁸⁴ These are: *ghosting*, where an object appears superimposed over itself, *misalignment of structures*, *misproportions* where structures or objects are out of proportion, and *a swirling blur* dubbed the Dali effect after the artist Salvador Dali's surrealistic style. For agricultural use, these errors may not be significant, but they could be a serious problem for structural inspection.

15.8 Scale and Traversability

Terrain identification is only one component of the larger issue of whether a region is traversable by a robot. The size and content of a region, beyond just identifying the terrain type in front of the robot, also present opportunities and risks to a robot. Therefore it is helpful to think more formally about the robot's size relative to its intended environment, or the *scale*, and the *traversability* of the environment, following the metrics in.¹⁴¹

The operating workspace, or environment E , for a robot consists of one or more *regions*, and each region may have different size limitations and unique traversability characteristics. For example, a ground robot exploring a mine collapse might have to move through a narrow, smooth symmetrical borehole (one region) and then enter a muddy mine floor covered with debris (a second region). The two regions suggest different mobility constraints that have to be combined into a successful system. A large robot could easily handle the muddy floor and climb over debris but it could not pass through the borehole. As an another example, a small UAV may be able to fly long-duration missions at altitudes above the tree line (a region) but requires a large meadow clear of trees and brush for launching and landing (a different region from the sky above the tree lines). Likewise an ROV exploring a shipwreck has to be able to move in open water but also inside the wreck.

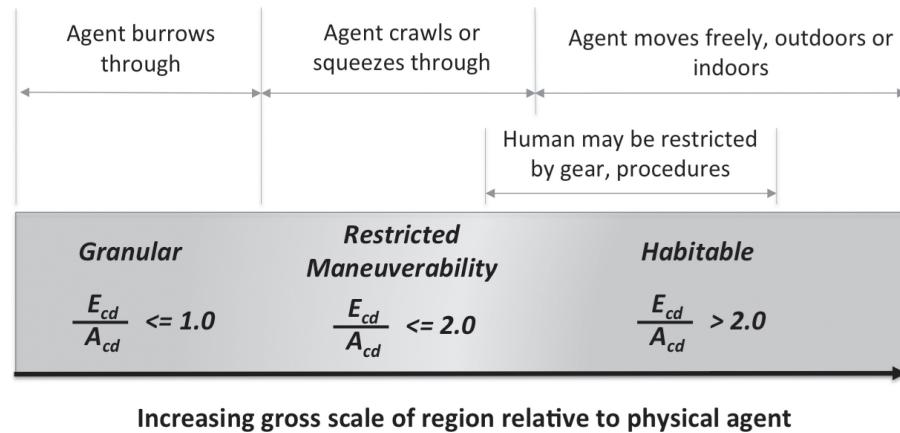
15.8.1 Scale

The ability of a robot to operate in an environment is affected by the relative scale of the robot compared to the environment—a small robot can work in smaller regions than a large robot can. The size of a robot or an environment can be expressed as the characteristic dimension cd . The characteristic dimension for a ground robot is its turning radius. For a UAV, it is the standoff distance it can safely hover from objects, or how closely the UAV can comfortably fly near objects. The characteristic dimension of an environment is the cross section of the narrowest spot a robot would be expected to move through. If a robot was to go into a pipe that gradually became smaller or larger, the characteristic dimension would be the smallest cross-section of the pipe.

The ratio of the characteristic dimension of the environment E_{cd} to the robot (or a human or canine agent) A_{cd} roughly divides environments into *three mobility regimes*:

- $\frac{E_{cd}}{A_{cd}} < 2.0$, *habitable* regime. In this regime, the robot has the space to move freely. A ground robot can turn around rather than have to reverse itself and back up. Mobility may be compromised by traversability, but it is not limited by the size of the region.
- $\frac{E_{cd}}{A_{cd}} \leq 2.0$, *restricted maneuverability* regime. In this regime, the robot can move but not necessarily turn in place. The robot has a much lower margin of error.
- $\frac{E_{cd}}{A_{cd}} \leq 1.0$, *granular* regime. This regime presents a more difficult case of restricted maneuverability because the robot is essentially burrowing into the environment and is in physical contact with the environment beyond normal vehicular mobility.

[Figure 15.6](#) illustrates the three regimes. Note that a human might normally be able to move in areas, such as a nuclear power plant, that would be considered habitable spaces, but once the human dons protective gear, the characteristic-dimension ratio may change and put the person in closer proximity to walls, equipment, and so forth.



[Figure 15.6](#) Three regimes of gross scale of a region's environment relative to that of a physical agent.

15.8.2 Traversability Attributes

TRAVERSABILITY

Traversability is the ability to move through a region. The traversability of any robot in a region is influenced by at least four properties of the environment, listed below.

- *Verticality*. Verticality is the maximum slope of the region. A bomb squad robot, designed to move horizontally on flat floors, may not be able to move down a sharp drop off in a collapsed building. UAVs rarely change altitude during indoor navigation.² Therefore it is important to speculate if a region will require the robot to move vertically up or down, and, if so, how much.
- *Tortuosity*. Tortuosity is a measure used by biologists to analyze an animal's movements; it is the

number of turns an agent takes per unit direction. A high-tortuosity path indicates that the animal is searching or frequently confused. A high-tortuosity environment indicates that a robot will have to be able to navigate significant twists. See [figure 15.7](#).

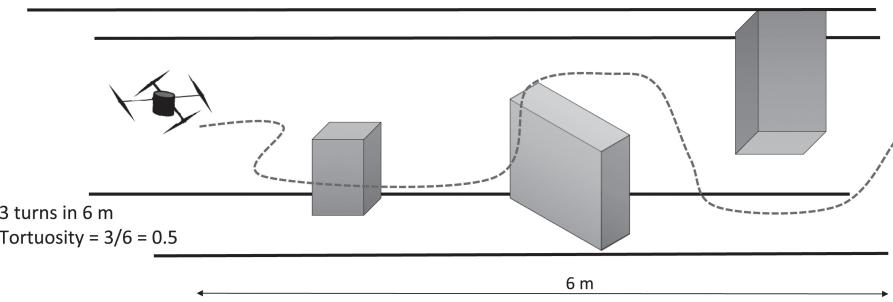


Figure 15.7 An example of the tortuosity of a region for a UAV (from Agarwal and Murphy).²

- *Severity of obstacles.* The severity of obstacles is similar to tortuosity. It is an estimate of clutter or intrusions into the space. An a priori map of the environment might suggest it has a low tortuosity region but the map may not show the location of furniture, vegetation, or rocks that would reduce the traversability of a ground robot through the region. In theory, a UAV might be able to fly at a nominal altitude above furniture and below hanging lighting fixtures but, in practice, it might have to deviate. Thus it might contact the obstacles, which could cause the UAV severe damage.
- *Surface properties.* An environment will have surfaces that impact mobility. For example, a tracked vehicle that works perfectly in the muddy interiors of concrete sewer pipes may have problems on thick shag carpeting.
- *Accessibility elements.* These are architectural elements, such as doors, windows, elevators, and stairs, that connect regions. A problem encountered at the Fukushima Daiichi nuclear accident was that the ground robots had largely been designed for use in a single outdoor region. The buildings required the robots to go into a series of rooms and hallways, pass through doors, and climb and descend stairs. The environment consisted of multiple regions with difficult to navigate stairs or hard to open doors.

Note that the tortuosity, severity of obstacles, and accessibility elements are a function of the scale of the robot with respect to the environment. A door threshold between rooms, a common accessibility element, can be impossibly large for a very small robot or insignificant to a large robot.

Traversability issues for a ground robot also include surface properties such as the soil features or paving. A marine vehicle would also have to contend with currents, waves, and possibly wind. A UAV would also have wind and factors, such as humidity, that would impact its ability to move in a region.

15.9 Exploration

EXPLORATION

Exploration, perhaps the opposite of mapping, asks: *Where haven't I been?* instead of: *Where have I*

been? While exploration can be accomplished reactively, a central concern in exploration is how to cover an unknown area efficiently. There are two general categories of algorithms for deliberative exploration: *frontier-based* and *generalized Voronoi graph* (*GVG*) approaches. These approaches work satisfactorily indoors but their utility for outdoor work spaces has not been established.

15.9.1 Reactive Exploration

Exploration can be accomplished behaviorally. One way is with a random search; the robot literally wanders around randomly (using a random potential field). After some (often very large) period of time, statistics indicate it should have covered the entire area. Another reactive method is to permit a short-term persistence of proprioception (odometry), where the robot is repulsed by areas that have been recently visited; an *avoid past* behavior.¹⁷ The *avoid past* behavior can be implemented as a repulsive field, generated by every visited cell in a coarse occupancy grid or for every previous heading. Another behavioral approach is to exploit evidential information in an occupancy grid. As the robot explores a new area, many cells on the grid will be unknown (rather than occupied or empty, recall chapter 14), and the robot can use the centroid of the unknown area as a goal for a move-to-goal. While the above behavior-oriented approaches are simple and easy to implement, they often are inefficient, especially when presented with two or more unexplored areas. Suppose a robot has encountered a hallway intersection; how does it choose which area to explore?

The two basic styles of exploration methods that have emerged which rank unexplored areas and make deliberative choices are: *frontier-based* and *generalized Voronoi graph* methods. Both work well for indoor environments; it is less clear how these work over large open spaces. Both use behaviors for navigation, but they are different in how they set the navigational goals. This section provides a highly simplified overview of each method.

15.9.2 Frontier-Based Exploration

FRONTIER-BASED EXPLORATION

Frontier-based exploration was pioneered by Brian Yamauchi.¹⁸⁹ The approach assumes the robot is using a Bayesian occupancy grid (a Dempster-Shafer grid can be used as well). As shown in figure 15.8, when a robot enters a new area, there is a boundary between each area that has been sensed and is open and the area that has not been sensed. (The boundaries between the occupied areas and the unknown areas is not interesting because the robot cannot go through the occupied area to sense what is behind it.) There are two such boundaries in figure 15.8; each of these lines forms a *frontier* that should be explored.

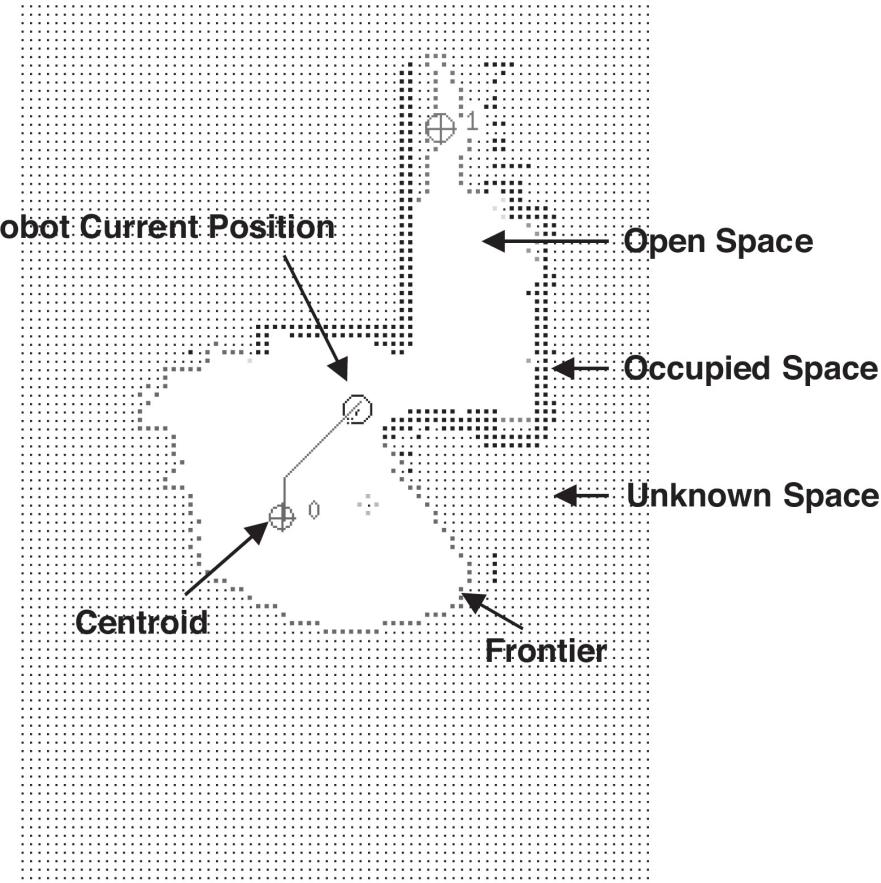


Figure 15.8 Example of a robot exploring an area using frontier-based method. (Figure courtesy of the Navy Center for Applied Research in Artificial Intelligence, Naval Research Laboratory.)

The choice of which frontier to explore first can be made in a variety of ways. A simple strategy is to explore the nearest frontier first. Another is to explore the biggest frontier first. Since the world is unknown, the robot has no way of knowing that upon reaching a big frontier, it will discover a wall just a meter away. This means that the robot might move across a room, briefly explore one area, then return almost to its starting point, explore that area, and then go to another place, and so on. In practice, this does not happen that often with indoor environments.

15.9.3 Generalized Voronoi Graph Methods

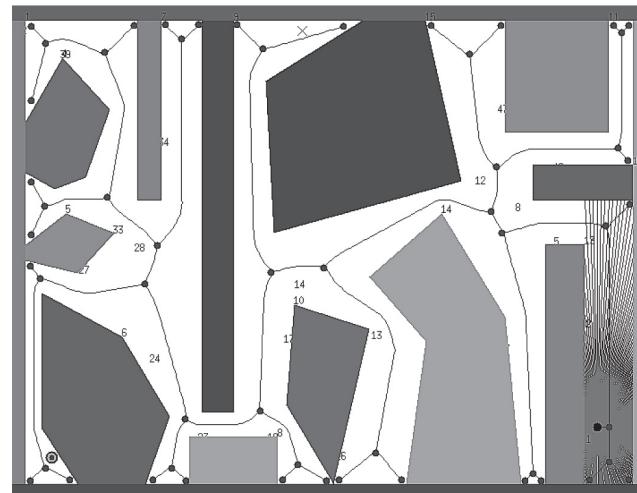
Another method of deciding how to explore a space is to have the robot build a reduced generalized Voronoi graph (GVG) as it moves through the world and to detect gateways or opportunities for its movement choices. This method has been used extensively starting with Howie Choset.^{45;44;43}

GENERALIZED VORONOI GRAPH (GVG)

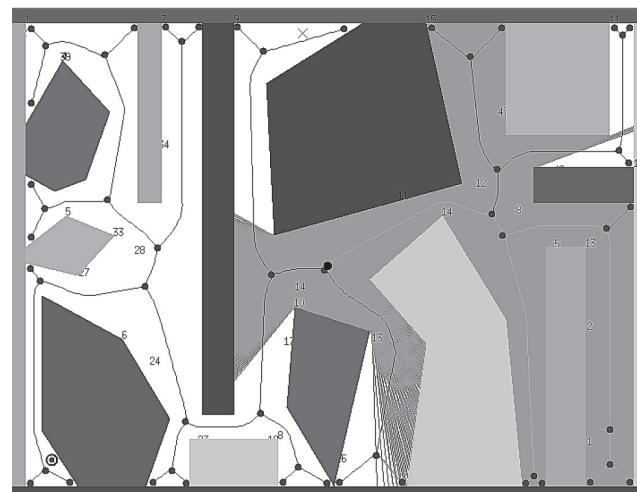
In the *GVG approach*, the robot attempts to maintain a path that places it equidistant from all objects

it senses as it moves. Essentially, the robot tries to move ahead but stay in the middle between, or at a tangent to, surrounding objects. This path is a GVG edge, the same as would be generated by decomposing the space into a Voronoi graph. Generating and following the generated path is straightforward to do with a behavior.

When the robot comes to a dead end or a gateway, there are multiple GVG edges that the robot could follow. As shown in [figure 15.9](#), dead ends produce two GVG edges. But, in this case, the robot can perceive that both of these edges end at objects, so there is no reason to follow them. The robot can then backtrack along the path it had been on, either to the starting point or to another branch. If the robot encounters branches in the GVG edges, it can choose one at random to follow.



a.



b.

Figure 15.9 Example of a robot exploring an indoor area using a GVG method. a.) The robot starts in the lower right corner and b.) explores the center. The black lines indicate the ideal GVG and the white lines indicate the portion of the

GVG that the robot has traveled. (Figures courtesy of Howie Choset.)

[Figure 15.9](#) shows how the robot would explore an area. For convenience, the figure shows the entire GVG that would be drawn *after* the robot had fully explored the area. The robot would begin by sensing the walls on each side (without any recognition that it was in a hall). The sensed area is shown in light grey. The robot would attempt to center itself between the walls while moving perpendicular to the line of intersection. Eventually it reaches an intersection. The intersection creates two edges, neither of which appear to terminate at an object. The robot arbitrarily chooses the left edge and stores the right in case the algorithm must backtrack. It does this repeatedly as seen in [figure 15.9b](#) and continues up the hall until it comes to the dead end. The robot then backtracks, using the same edge behavior (stay in the middle). It continues to favor exploring to the left until it comes to a dead end in the lower left, and then backtracks as shown in [figure 15.10a](#). The robot continues to explore and backtrack until the entire area is covered as shown in [figure 15.10b](#).

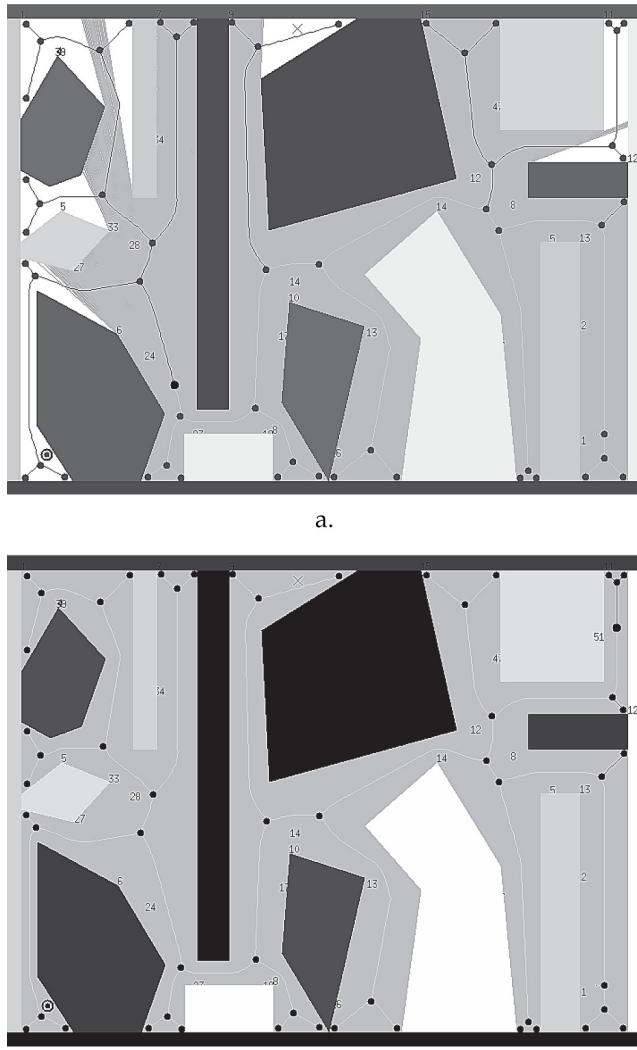


Figure 15.10 Example of a robot exploring an indoor area using a GVG method, continued. a.) reaches a dead end and backtracks, and b.) completely covers the area, though it has not traveled all edges. (Figures courtesy of Howie Choset.)

15.10 Localization, Mapping, Exploration, and AI

Localization, mapping, and exploration involve many of the seven key areas of artificial intelligence introduced in chapter 1. These functions are inherently perceptual, linking the perceived world to symbolic maps; they rely on *vision*. Exploration can involve deliberative *planning*, but it can also be done reactively. Most of the algorithms involve probabilistic *search* and *inference* to find poses that match the map or best align with the previous scan. At first glance, localization, mapping, and exploration would primarily require *learning*, as in learning a new environment. However, the

algorithms are perceiving and constructing representations, not learning in the classic AI meaning of the term. Classic machine learning is used for terrain identification, where the system associates proprioceptive or exteroceptive cues with terrain types. Localization, mapping, and exploration do not involve *understanding natural language*. They can engage *distributed AI* using multiple robots to map out a large or complex area concurrently.

However, more work in AI is needed, especially in *knowledge representation*. This statement may be counterintuitive because a map is a knowledge representation. The challenge is the symbol-ground problem, which may be the limiting factor in mapping. As seen with localization and SLAM algorithms, reliably matching features on a map to features in the environment or learning what features should be incorporated into a map is difficult. Localization, mapping, and exploration may be where more sophisticated problem solving, inference, and learning will add value to robot navigation as a whole.

15.11 Summary

Mobile robot localization requires determining the pose of a robot relative to a given map of the environment. Localization algorithms are variants of Bayes Filter algorithms, where the belief in a pose is a function of the robot's pose, the control actions, and a map. Iconic localization is more common than feature-based localization due to reliability problems in extracting features. Feature-based localization historically has used extended Kalman filtering solutions but these methods require correspondence variables to link the observations of features to symbolic features in the map. In practice, Monte Carlo localization (MCL) methods dominate, both for iconic and feature-based approaches. Localization works very well for static environments with good models, but dynamic environments are difficult.

Simultaneous localization and mapping (SLAM) algorithms work well for indoors, especially if the robot uses range scans and can make multiple loops through an area and close the loops. Frontier-based and generalized Voronoi graph methods are two types of exploration, but they are not necessarily as intuitively efficient as might be expected.

Terrain maps, now available at low resolutions, are good for planning but terrible for execution. Outdoor navigation requires the ability to sense terrain. There are no reliable proprioceptive or exteroceptive methods for terrain identification, though proprioception appears more promising for practical use in the near term. Stereophotogrammetry is enabling robots to create orthomosaics, digital elevation maps, and digital surface maps.

The mobility of a robot in an environment can be partially quantified in terms of the relative scale of the environment to the robot and the traversability of the environment. In practice, most environments consist of multiple regions with different scales and traversability. The relative scale of an environment E to a robot or other type of agent A is the ratio of their characteristic dimensions. The ratios can be used to divide regions in three categories of mobility. If the environment is large and spacious compared to the robot, $\frac{E_{cd}}{A_{cd}} < 2.0$, then the region is easily *habitable* for the robot. If the environment and robot are similar in scale, $\frac{E_{cd}}{A_{cd}} \leq 2.0$, then the robot has restricted maneuverability. If the environment is smaller than the robot and the robot is essentially burrowing into the region, $\frac{E_{cd}}{A_{cd}} \leq 1.0$, then the region is granular. Traversability has four general characteristics: verticality, tortuosity,

severity of obstacles, and accessibility elements. In addition, ground robots are impacted by surface properties, aerial vehicles by wind, and marine vehicles by waves.

Localization, mapping, and exploration involve most of the fundamental areas of AI. However, the symbol-ground problem remains a general challenge for AI, and existing solutions are limited and highly domain-specific.

Returning to the questions posed in the introduction, the answer to *Why is this a problem? Can't you just use GPS, some sort of RFID beacons, or 3D scans from RGB-D sensors?* should be clear. Localization, mapping, and exploration are complex. The answer to *How can you simultaneously map the world and be sure where you are?* is addressed by research into simultaneous localization and mapping (SLAM). SLAM techniques rely heavily on probability theory. *How do you explore new areas efficiently or at least consistently?* can be answered with either frontier methods that direct the robot to the largest unexplored area at any time or GVG methods that systematically work locally and aggregate globally. The question of *How do you label the map with objects and features or terrain?* remains unsolved and is an area of active research.

Localization and mapping complete the coverage of navigation, with exploration as an activity related to, but distinct from, navigation. While navigation is only one application of deliberation in an intelligent robot, it is essential for mobility and thus appears in every technical architecture. The next chapters shift attention to the interactive layer.

15.12 Exercises

Exercise 15.1

Define the pose of a mobile robot.

Exercise 15.2

Describe the difference between iconic and feature-based localization.

Exercise 15.3

Define each term in: $bel(\mathbf{x}_t) = f(bel(\mathbf{x}_{t-1}), \mathbf{u}_t, \mathbf{z}_t, m)$.

Exercise 15.4

Discuss the differences between the EKF, Grid, and Monte Carlo Localization algorithms and identify which one is associated with each of the three types of localization problems.

Exercise 15.5

Describe the differences between *localization* and *simultaneous localization and mapping*.

Exercise 15.6

Define the *loop closure* problem.

Exercise 15.7

Compare and contrast *frontier-based* and *generalized Voronoi graph (GVG)* based exploration.

Exercise 15.8

Give at least one example of each of the two types of methods for proprioceptive terrain identification and exteroceptive terrain identification and discuss the limitations of each.

Exercise 15.9

Describe the differences between DTED, DEM, DSM, and orthomosaics.

Exercise 15.10

Explain the five attributes of traversability: verticality, surface properties, tortuosity, severity of obstacles, and accessibility elements.

Exercise 15.11

Give examples of two types of methods for proprioceptive terrain identification and exteroceptive terrain identification by ground robots and discuss the limitations of each.

Exercise 15.12

List the possible uses of terrain data by a robot and the impact of terrain on the robot's functions.

Exercise 15.13

Give the ratios for each of the three scales of an agent to the environment and discuss what they mean for a robot.

Exercise 15.14

Would it be hard for a robot without GPS to localize itself in a desert? Why or why not?

Exercise 15.15

What is the difference between projective and reactive terrain identification?

Exercise 15.16

What is the impact of the symbol grounding problem on map making and interpreting a map?

15.13 End Notes

For a roboticist's bookshelf.

Probabilistic Robotics by Thrun, Burgard, and Fox, MIT Press (2005)²⁰⁷ is the classic introduction to Bayes filtering.

16

Learning

Chapter Objectives:

- Describe four major types of learning from experience (*supervised*, *unsupervised*, *semi-supervised*, and *reinforcement learning*) and their associated techniques.
- Describe the following learning techniques: *induction*, *support vector machines*, and *backpropagation*.
- If given a feedforward *artificial neural network* with inputs, weights, and an *activation function*, compute the output of a unit.
- If given a *rewards matrix* and a random exploration, write the formula for *Q-learning*, and then create a *policy*.
- Define *evolutionary robotics*, and explain why the associated methods are not considered learning.
- Define the processes in *genetic algorithms*: *selective reproduction*, *random mutation*, and *genetic recombination*.
- If given a set of pairs of *genomes* and a *cross-over point*, produce the next generation in the population.
- Define *critic*, *new term problem*, *overfitting*, *credit assignment problem*, and *online learning*.

16.1 Overview

The previous chapters have conveyed the challenges in organizing and implementing the SENSE, PLAN, and ACT primitives for locomotion, navigation, and manipulation, recognizing objects, and deliberation. The complexity and difficulty of reaction and deliberation seem at odds with how natural these capabilities are for a person, and this leads to the cry: *Can't robots just learn to do things?* Indeed robots can learn to do things, as well as recognize objects and situations, but, sadly, learning is another illustration of the rubric that if it is easy for a person, it is hard for a computer. Desiring to endow robots with learning leads to two questions: *What is learning?* and *What are the types of learning?* While one might suspect that there are different types of learning, neural networks, especially deep learning, feature prominently in entertainment and news media. Therefore, it is natural to ask: *Why are*

robots not using neural networks all the time? They are like the brain, right? From a more practical design viewpoint, the question: *Where does learning go in our operational architecture?* needs to be answered.

This chapter will attempt to answer all of these questions. The chapter begins by defining what learning is and by describing the general types of learning. The types of learning generally fall into two categories: the agent learns by example or experience or the agent learns by reasoning or analogy. One surprise is that genetic algorithms, and a related technique called simulated annealing that will not be covered here, are not technically learning algorithms though they are associated with learning. Next, the chapter presents, in more detail, three classes of algorithms for learning by example or experience, beginning with unsupervised learning and the popular artificial neural networks. The second class of algorithms deals with supervised learning, which leverages the investments in data mining targeting learning how to recognize patterns. The two main techniques of supervised learning are induction and support vector machines. The third class of algorithms is Q-learning, which is possibly the most popular type of learning in robotics because it focuses on learning optimal actions, or what to do next. Once the types of learning have been explored, the chapter tries to pull it all together by giving an overview of where learning fits into a robot architecture. Next, the chapter describes genetic algorithms and discusses how they relate to learning. The chapter concludes with observations about learning and artificial intelligence.

16.2 Learning

LEARNING

Learning is defined as the process by *which an agent improves its performance on future tasks after making observations about the world.*¹⁸¹ Learning allows the agent to adapt to changes in the world, for example, changes in terrain or tasks, or to changes in itself, such as having to expend more energy to compensate for the wear and tear of motors, versus long term changes that would be captured by evolutionary processes, for example, climate change.

There are many techniques for machine learning, and, in each case, according to Russell and Norvig,¹⁸¹ the designer has to know:

- *The component that is to be improved.* Machine learning can be used to improve functions within any of the five subsystems discussed in chapter 4: Planning, Cartographer, Navigation, Motor Schema, and Perception.
- *The prior knowledge available to the agent.* Does the robot just wake up as a tabula rasa, with no prior knowledge, or does it already have a way of performing the function but needs to improve it?
- *The representation of the data and the component to be learned.* Are the data RGB images? A list of values? As is known from the study of algorithms, a good representation can make the algorithmic portion of a solution much simpler and faster to execute.
- *The available feedback to learn from.* In order to improve, a robot needs a feedback error function of some type. The robot may be told whether it is performing well or it may have sensor readings that let it measure improvements internally.

Here are four examples of how machine learning has been used in robotics and what the designer had to know in terms of component, prior knowledge, representation, and feedback. The first example is one of the original applications of machine learning to robotics which originated in the late 1980s,^{47;46} when roboticists began learning control laws in order to control the trajectory of a manipulator or other repetitive robotic motions. Many industrial manipulation applications require the robot to perform repetitive actions, for example, holding a tray of parts perfectly flat as it transfers them. In these applications, the desired trajectory is known. The challenge is to determine the control law for coordinating the torque and timing of the individual joints of a multi-degree of freedom robot designed to follow a trajectory and to adapt to noise from wear and tear in actuators or environmental factors over time. The complexity of continuous interactions between joints and torques is nonlinear and thus difficult to represent mathematically and solve. Fortunately, the robot can be observed so that the difference between the desired trajectory, often called the reference trajectory, and the actual trajectory can be measured. The robot can then learn through repetitive, or iterative, trials how to control itself. In this example, the component to be improved is the control law for motion, the prior knowledge is the reference trajectory, and the feedback is the measured error between the actual and reference trajectory. Specific iterative learning algorithms will have different data representations, such as a learning gain matrix to record learning over the iterations.²¹⁶

A second example is learning places. A persistent problem in robotics is converting noisy sensory data into symbolic data, especially in robotic navigation where there is a large difference between an affordance of a gateway (“I can change direction”) and place recognition (“I am in front of the door to the main office”). Many techniques for learning places divide the world into patches or vectors of features such as the viewframes in chapter 13. The robot uses Bayesian statistics to associate the viewframe, which will not exactly match what the robot sees, with the location coordinates, which will also be subject to error. In this example, the component to be improved is the map of the world, the prior knowledge consists of the types of features that should be used, the representation is the viewframe or vector, and the feedback is from a set of trials where the robot approaches the place from different angles.

The third example is learning terrain. It is often impractical for robots to learn how to navigate difficult terrain through computer-based simulation or controlled trials. Difficult terrain, which may have changes in elevation, mud or rocks, and different types of vegetation, is extremely hard to model. One solution is to continuously update the terrain model using online learning. The robot uses its sensors to predict the terrain ahead of it and then uses feedback, such as torque (e.g., indicating mud), vibration (e.g., bumpy), IMU (e.g., am climbing), and so forth, to confirm whether the projection was correct. In this example, the component to be improved is terrain modeling, the prior knowledge is how the sensors infer terrain features, the data representation is a sliding map of the projected terrain, and the feedback is actual use.

A fourth example is learning allocations of behaviors. The modularity of behaviors can support learning selections of behaviors through trial and error for a new situation, such as how to find an object and bring it back.^{122;130} A robot can learn when to use each of its set of behaviors, say search, move-to-goal, grasp, and avoid, or elementary operations,^{24;23} in the right sequence or with the right gains. Returning to the list of design elements, the component to be improved for this example is the action selection, the prior knowledge is the set of available behaviors, the representation is the list of

combinations and sequences of behaviors that have been tried, and the feedback is whether the robot accomplishes the goal.

16.3 Types of Learning by Example

There are many forms of learning, including learning from example, learning from experience, learning from demonstration (also called imitation learning), learning by reasoning, case-based or explanation-based learning, and learning by analogy. Learning from example, where the agent is presented with examples of a concept or action, is the most common and has been used successfully with robots.

A traditional way to categorize learning by example algorithms is by the style of feedback.¹⁸¹ There are four categories of feedback: *supervised*, *unsupervised*, *semi-supervised*, and *reinforcement learning*. Of these four, the first two are commonly used in robotics for learning how to perceive the world. Semi-supervised learning is also often applied to perception but less frequently than supervised or unsupervised learning to robotics and will not be discussed further. Reinforcement learning is perhaps the most common form of robot learning because it can be applied to learning multi-step sequences of actions. These are called policies.

SUPERVISED LEARNING

Supervised learning relies on explicit and direct feedback. The robot is trained to recognize a concept—an object, place, state, or association. It is presented with examples, perhaps an image, that are labeled with the correct output for each input. The robot takes this set of (input, output) pairs and learns what attributes in the input leads to the output classification. When presented with a new input, the robot recognizes whether the concept is present or not. For example, training a humanoid robot to perform a task by watching what motions a person uses.¹⁸⁵ The person's motions are the labelled input for the concept, which is the task or skill.

UNSUPERVISED LEARNING

Unsupervised learning uses indirect feedback. The robot is given a set of attributes for an input data set and learns to recognize how attributes cluster without being told what to learn. The robot examines the set of inputs and uncovers the objects, places, states, or associations hidden in the input by examining the mathematical significance of the occurrence of their attributes. Unsupervised learning is often associated with *data mining* and *clustering*. An example is Rogue, a robot which learned to avoid taking routes through the lobby during times of the day when it was likely to be crowded and would instead, take a faster, though longer, path.⁸⁵

SEMI-SUPERVISED LEARNING

Semi-supervised learning uses a small set of labeled (input, output) pairs as per supervised learning to help guide unsupervised learning from unknown inputs. For example, an autonomous car can be given a few images of a person on a bicycle and a few tracks, and then learn to track a bicyclist and predict where the cyclist might dart in and out of traffic.²⁰⁴

REINFORCEMENT LEARNING

Reinforcement learning uses feedback from trial and error to learn sequences of actions. The robot is given a goal, tries possible actions to reach the goal, and then estimates the value of each action in the

sequence of reaching the goal. Reinforcement learning is similar to unsupervised learning because the attributes, in this case the optimal set of actions, are hidden. Unlike unsupervised learning the goal, or labeled output, is known. Unlike with supervised learning, the robot generates its own inputs by exploration. One example is Brachiator, a robot that learned to swing across a set of horizontal rungs like a gibbon.¹²⁸ Brachiator used reinforcement learning to learn how to coordinate the sensory-motor actions so that the legs generate the swinging momentum and the grippers reach and grasp the next handhold.

16.4 Common Supervised Learning Algorithms

In supervised learning, the agent observes some example (input, output) pairs during the training phase and learns a function, h , that maps the input onto the output. The output is labeled as positive or negative. The major techniques are: induction, support vector machines (SVM), and decision trees. Induction and support vector machines are very similar and support vector machines now dominate supervised learning in robotics.

16.4.1 Induction

INDUCTION

Induction infers a concept from labelled (input, output) pairs. During training, some of the pairs will be examples of the concept (“Is”) or positive instances. But some of the pairs will not have the concept, or be a negative instance (“Is Not”). The learning algorithm draws a line that best divides the input into “Is” and “Is Not,” as shown in [figure 16.1](#). The line is called the hypothesis, h , though it is technically a hypothesis function. In practice, the input may have multiple attributes so that the x axis is really multi-dimensional. The concept involves more than “Is” or “Is Not” and reflects categories of a larger concept, say fruit that may contain “apples,” “oranges,” and “grapes.” The set of features x might be the color, shape, and size of the input being classified.

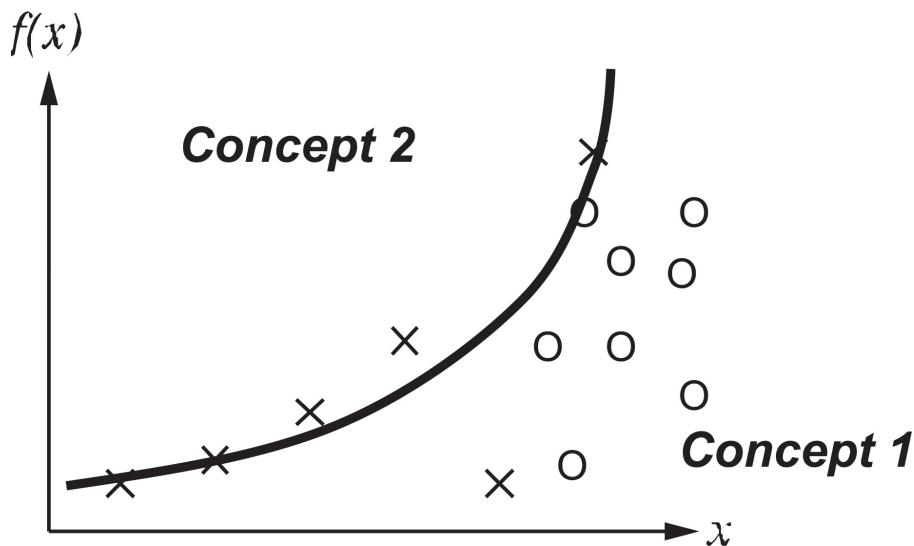


Figure 16.1 An example of induction.

OVERFITTING

A major problem in induction, and supervised learning in general, is *overfitting*. It is possible that the algorithm generates a h that perfectly classifies the input during training but makes major mistakes when it is given real inputs. [Figure 16.2](#) shows an example of two different h functions which are *consistent* with the training data but probably will not do well classifying new data once it is put into operation.

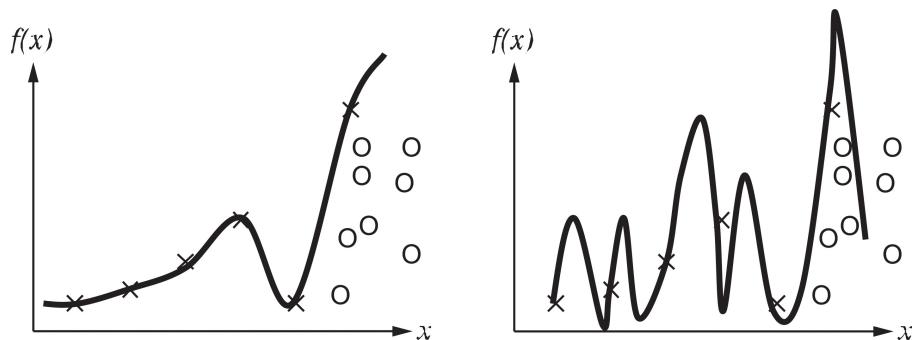


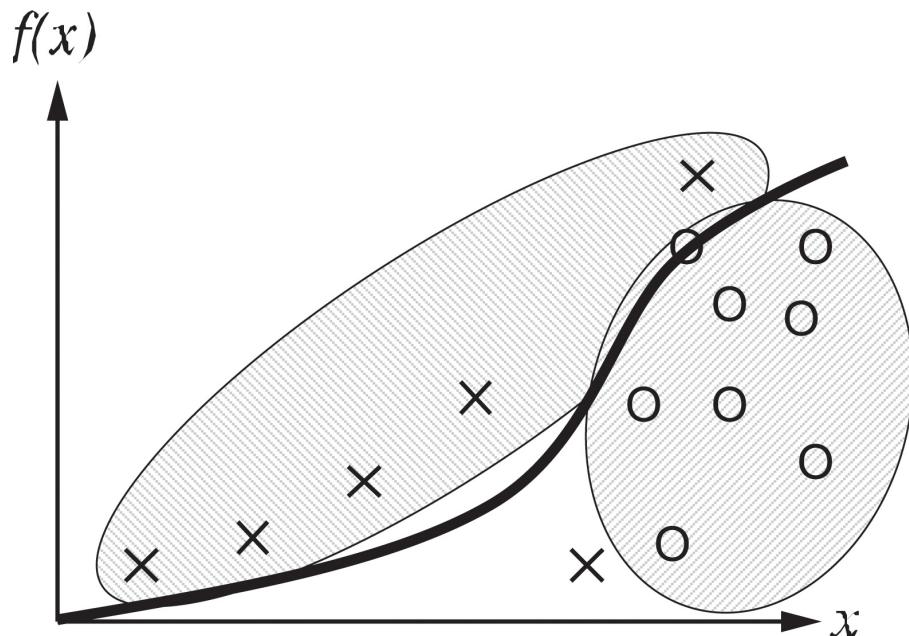
Figure 16.2 Two examples of overfitting. Both are consistent with the training data but unlikely to correctly classify new data.

16.4.2 Support Vector Machines

SUPPORT VECTOR MACHINES

Support vector machines are inductive learning mechanisms which try to avoid overfitting by using the statistical properties of the data. Classic induction techniques typically try to minimize the error between individual outputs and h . This is conceptually similar to curve fitting with the least squares

method. Induction can produce a curve that perfectly, or nearly perfectly, divides the output into the right concept. However, h may be overfitted. Support vector machines try to avoid overfitting. They do not consider the individual outputs but rather look at groups of outputs which form a statistical distribution as shown in [figure 16.3](#). The curve is fitted to partition these groups, thereby reducing the likelihood that the curve overfits the data. There is the possibility that a few outliers will be misclassified and will fall on the wrong side of the curve, but that possibility existed with inductive methods as well.



[Figure 16.3](#) An example of grouping of outputs by a support vector machine.

16.4.3 Decision Trees

DECISION TREES

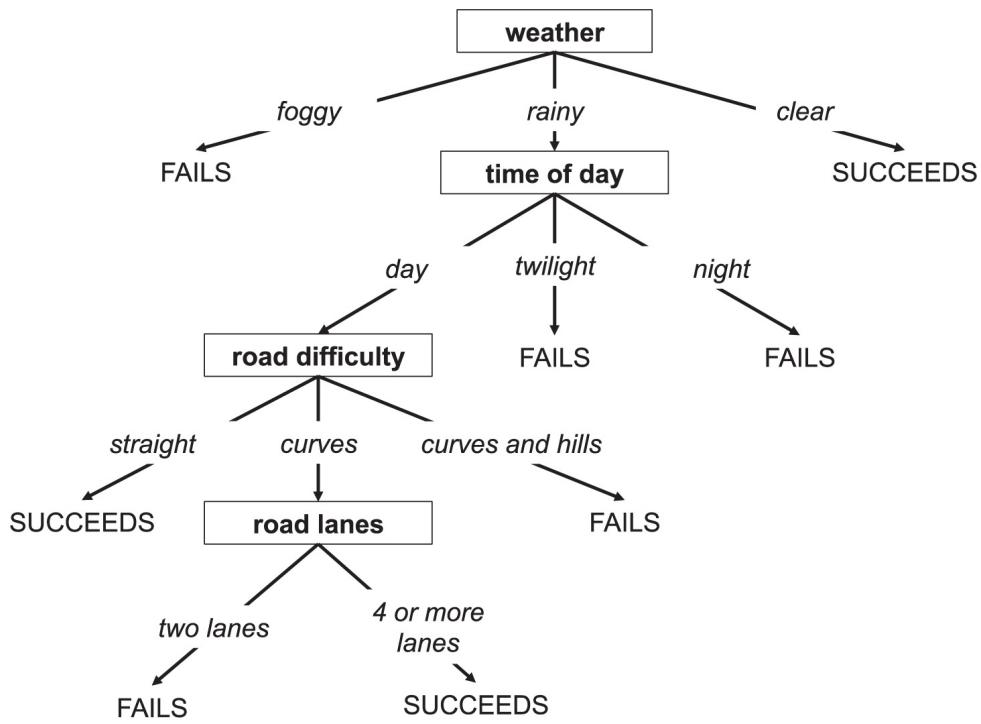
Decision trees differ from support vector machines in that decision trees learn which set of measurements are valuable and the order in which to get the measurements. Decision tree algorithms use information theory to learn the best choice or action.

For example, consider predicting when an autonomous road system will fail during a drive so that the system can alert the human driver to the possibility of having to take over. Suppose a robot had access to a table of measurements about the terrain and weather attributes for various trials and whether the autonomous road following system had failed and the human had to take over. See [figure 16.4a](#). Based on that data, the robot could learn to predict whether it would fail for a particular combination of attributes. But it could, at the same time, also learn *what order to query the data*. For example, the robot might learn that if it is foggy, the autonomous control always fails. Thus the robot could save time by downloading the weather report first to see if it is foggy. If it is not foggy, the robot can then

look at the next most informative attribute. This process results in a decision tree such as the one in figure 16.4. ID3¹³² is the classic decision tree algorithm.

time of day	weather	road lanes	road difficulty	result
day	clear	4 or more lanes	straight	SUCCEEDS
rainy	twilight	4 or more lanes	curves	FAILS
day	clear	two lanes	curves and hills	SUCCEEDS
foggy	twilight	two lanes	curves and hills	FAILS
rainy	day	4 or more lanes	curves	SUCCEEDS
foggy	night	4 or more lanes	curves	FAILS
...

a.



b.

Figure 16.4 An example of a.) a training set and the resulting b.) decision tree.

16.5 Common Unsupervised Learning Algorithms

Unsupervised learning learns patterns hidden in the input even though no explicit feedback is supplied. It is a popular type of learning, and it is used by web applications for *data mining* about customers to discover trends or preferences. In robotics, a robot might learn different types of terrain by crossing the terrain and associating a particular vibration with it or by determining which sensors work best for

different environments. The two major algorithms for unsupervised learning are clustering or pattern recognition algorithms, of which *Bayesian statistical learning* is possibly the best known, and *artificial neural networks (ANN)*, which are the basis for *deep learning* techniques. ANN can also be supervised. This section reviews a feedforward ANN with backpropagation at a high level in order to acquaint the reader with the general idea and terminology. The reader will need a more detailed study to apply the concepts.

16.5.1 Clustering

Clustering attempts to learn a set of concepts or actions by noticing similarities in the input set. For example, a learning algorithm may notice that oranges on a tree are round while leaves are a different shape. The algorithm would extract statistical properties that define Item 1 (oranges) and Item 2 (leaves). Then if presented with a new image, the program could compute how close the new image is to Item 1 and Item 2 and label the new image with the closest match. A challenge in clustering is that the attribute(s) being clustered must be specified. Furthermore, an algorithm, such as *k-means clustering*, requires that the number of clusters be specified in order to execute; this means that the designer has to know what is “hidden” or experimentally determine the best number of partitions by trial and error.

16.5.2 Artificial Neural Networks

ARTIFICIAL NEURAL NETWORKS (ANN)

Artificial neural networks (ANN) have been a subject of study within AI since the founding of the field. The idea from neuroscience, especially from the work of David Rumelhart,¹⁸⁰ is that the strength of inputs causes neurons to fire, thus creating a stimulus to other neurons which then fire at different strengths and so on through a network of connections, eventually increasing the strength of an output object or concept. The network of connections can capture nonlinear relationships that are hard to derive from clustering or support vector machines. ANN tend to be used for situations where there is a large set of attributes but no model of how the attributes contribute to recognizing an object or concept and where there is a large set of training examples to enable the network learning process to converge.

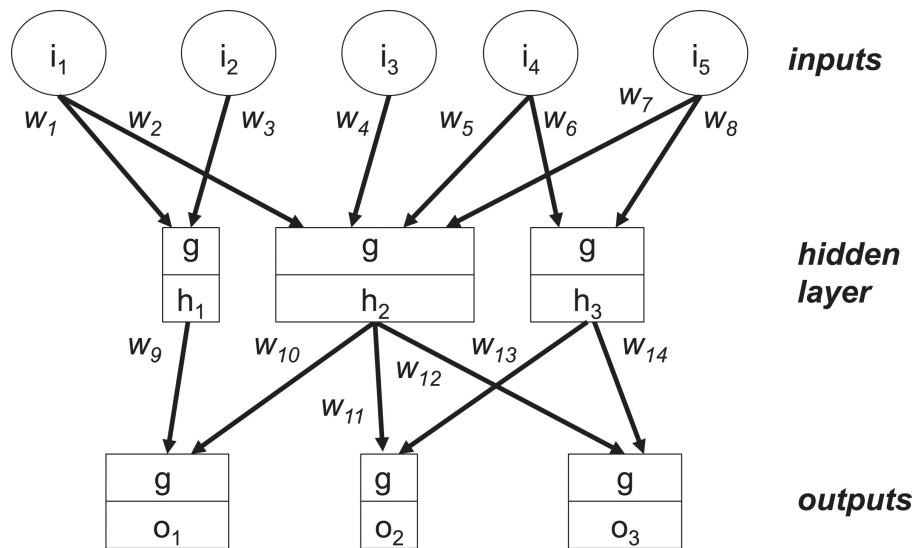
This section will first describe artificial neural networks in terms of the network topology and then use that topology to explain how the most common learning method, *backpropagation*, works. Each node or unit in the network has a set of weighted inputs that are transformed into an output. Readings or values for a training instance propagate forward through the network to produce the values for the network outputs. The unit with the highest value is the network output. The network learns from a training instance by comparing the computed network output with the ideal output. The difference is the error which is then used to adjust the weights. The designer specifies the starting topology of the network, which means that the network will not spontaneously learn a totally new term. A new term would, in effect, be created when none of the output choices specified in the topology is appropriate because there is a new object or concept not originally engineered in the system. Unfortunately, ANN are generally limited by the engineered topology and thus, by definition, cannot learn anything that was represented as potential output.

NEW TERM PROBLEM

The challenge of learning a new term versus uncovering a hard-to-find correlation is called the *new term problem*. There is significant research in generating new terms, either with ANN or other types of learning, but at this time there is no practical learning mechanism for robotics.

Network topology

[Figure 16.5](#) shows a network with five network inputs i , or attributes of interest, that will ultimately produce a strength or certainty in three network outputs o . The network inputs are usually sensed values such as percent green pixels in the object, leg up, or positive change in the encoder. Each node in the network represents a neuron called a *unit*. The network has four components, units, weighting of stimulus, hidden units, and activation functions, that allow it to learn complex relationships.



[Figure 16.5](#) An example of a feedforward artificial neural network.

The units are connected by edges reflecting the weighted strength of the stimulus to that unit. For example, the stimulus for percent green pixels in the object of an object with no green should be 0, but the stimulus value of a completely green object should be very close to 1.0. The stimulus may feed into multiple units. For example in [figure 16.5](#), the stimuli from i_1, i_4 , and i_5 go to two different units. However, the value or importance of the stimuli may be different for each of the receiving or stimulated units. Therefore, each stimulus for a unit has an associated weight, w_n for that unit. The weights w are learned. In [figure 16.5](#), the input to unit h_2 is $\text{input}_{h_2} = i_1 * w_2 + i_3 * w_4 + i_5 * w_7$.

ACTIVATION FUNCTION

PERCEPTRON

LOGISTIC FUNCTION

The unit takes the weighted inputs and applies an *activation function*, g_u , to combine the weighted stimuli and produce an output. The output is either multiplied by the w connecting it to the next unit, or it is the strength of the final network output o_j . The activation function can be anything. If the activation function is just a binary threshold, it is called a *perceptron*. Perceptrons are simple and extremely limited. Another type of activation function is a *logistic function*:

$$g_u = \frac{1}{1 + e^{-u}}$$

If the i_1 is the percentage of green pixels in a blob, say 0.9, with a $w_1 = 0.8$, and i_2 is how closely the shape of the blob approximates a circle, say 0.5, with $w_3 = 0.7$, then the input to h_1 is

$$\text{input}_{h_1} = i_1 * w_1 + i_2 * w_3 = 0.9 * 0.8 + 0.5 * 0.7 = 1.07$$

The output of the hidden unit h_1 would be:

$$g_{h_1} = \frac{1}{1 + e^{-h_1}} = \frac{1}{1 + e^{-1.07}} = \frac{1}{1 + 0.343} = 0.745$$

The output of a unit can be used as the stimulus for another neuronal unit or as the strength of the overall stimulus for the output concept. In [figure 16.5](#), the output of h_1 becomes the input only to one network output, o_1 . All multiple output concepts will have some degree of activation, and the one with the highest activation will be considered the correct concept (though ANN may make misclassification errors).

HIDDEN LAYERS

Artificial neural networks are able to make nonlinear classifications because they can associate the inputs with outputs with great flexibility due to the presence of *hidden layers*. ANN are not restricted to generating weights for the output concept based only on the direct inputs. For example, a traffic light might have sensed inputs of yellow rectangle, green round shape inside yellow rectangle, red round shape inside rectangle. If an object has strong yellow rectangle and green round shape inside yellow rectangle stimulus or strong yellow rectangle and red round shape inside rectangle, it is probably a traffic light. However, if both a green round shape inside yellow rectangle and a red round shape inside rectangle are strongly sensed at the same time, something is wrong and the network may not be able to identify the object. The flexibility in learning relationships is provided in ANN by the intermediate neuronal units called hidden layers. [Figure 16.5](#) shows a ANN with one hidden layer.

DEEP LEARNING

The word “hidden,” in the term hidden layers, turns out to be apt. In networks with large numbers of units and hidden layers, also called *deep learning*, the nonlinear relationships between the units become so complex that it is difficult or impossible to analyze what the network has learned. This means that the designers would not be able to express the neural network in the traffic light example in the previous paragraph in terms of if-then rules or a decision tree. The designer just has to trust that the hidden part of the neural network is doing its job. Hidden layers also present a design challenge because the optimum number of hidden layers for a problem is unknown; a designer may have to run

the training data over several versions of the network with different numbers of hidden layers. It should be noted that [figure 16.5](#) shows the network after it has learned the connections between units. The network often starts out with all possible inputs being directed to a unit, or a fully connected single directional graph. It learns that some connections have no relevance for a particular unit, and thus those edges can be eliminated.

FEEDFORWARD

RECURRENT

The units in the network can be connected in two primary ways: feedforward and recurrent. In *feedforward* networks all the arrows go in one direction, with the stimulus ascending from the sensory inputs to activate the output concept. [Figure 16.5](#) shows a feedforward network. A *recurrent* network allows bidirectional arrows and arrows that descend to lower hidden layers to create loops. Feedforward networks are the most common.

Backpropagation

BACKPROPAGATION

In the previous subsection, the topology illustrated how to classify an object or a concept based on inputs, or moving forward through the network, but not how the network learns. The learning mechanism in neural networks works by adjusting the weight w_i to reduce the error between the output o_i computed by the network and the target output it should have computed for the input i . The most common method for adjusting the weights is called *backpropagation*.

Backpropagation distributes the error between the outputs backwards along the weights on the connecting edges. For example, in [figure 16.5](#), the total error E can be estimated as:

$$\begin{aligned} E_{total} &= \sum \frac{1}{2}((target_{o_i} - out_{o_i})^2) \\ &= \frac{1}{2}(target_{o_1} - out_{o_1})^2 + \frac{1}{2}(target_{o_2} - out_{o_2})^2 + \frac{1}{2}(target_{o_3} - out_{o_3})^2 \end{aligned}$$

Backpropagation uses that error to update the weight. The idea is that the partial derivative of the change in the error E_{total} with respect to each individual weight w_i can be computed as:

$$\frac{\delta E_{total}}{\delta w_i}$$

For example, in order to adjust w_9 to reduce the error, one would compute $\frac{\delta E_{total}}{\delta w_9}$. Since δw_9 is not known, the chain rule can be applied:

$$\frac{\delta E_{total}}{\delta w_9} = \frac{\delta E_{total}}{\delta o_1} * \frac{\delta o_1}{\delta input_{o_1}} * \frac{\delta input_{o_1}}{\delta w_9}$$

The chain rule says that the error in the weight w_9 is a function of how much the total error changes due to a change in the output of o_1 , how much the output of o_1 changes due to changes in the input to o_1 , $input_{o_1}$, and how much the $input_{o_1}$ changes due to the change in w_9 .

Mathematically this generalizes to the delta rule:

$$\frac{\delta E_{total}}{\delta w_9} = -(target_{o_1} - output_{o_1}) * output_{o_1}(1 - output_{o_1}) * output_{h_1}$$

The delta rule is used to calculate the contribution of w_9 to the total error, how much the error in E_{total} changed due to the error in w_9 . In order to adjust w_9 , this error is subtracted from w_9 . However, a discount or learning rate, α , is often applied to the delta rule in order to dampen any large changes in the computed error due to noise in the training set. The update for w_9 , which is commonly marked with a superscript + to indicate it has changed is:

$$w_9^+ = w_9 - \alpha * \frac{\delta E_{total}}{\delta w_9}$$

The process is repeated, working backward (thus the term “backpropagation”) from the network outputs until all the weights have been adjusted. The network is presented with another example and works forward to compute the network outputs. Those outputs may still have a significant error, that then leads to the backpropagation phase. After many trials, perhaps hundreds or thousands, the network converges on the optimal values of the weights. However, that optimum is for the specific number of hidden layers and output categories; a different number of hidden layers may produce a better result.

16.6 Reinforcement Learning

POLICY

Supervised and unsupervised learning is typically associated with recognizing objects or learning patterns, while reinforcement learning is often used when the agent wants to learn a *policy*, denoted as Π , of what action to take given a set of circumstances. The policy may be quite sophisticated. Consider learning to throw a ball where the next action is part of a larger sequence of raising the arm, lifting a foot, positioning the torso, flexing the wrist, throwing a bit harder to overcome the current wind effects, and so on. Furthermore each element in the sequence has many variables that can be adjusted, including changing the velocity of a joint if its position is slightly off. The basic idea is that an agent learns those adjustments as it explores the problem space, just as a pitcher learns, through practice, how to coordinate his or her body to throw the ball which reinforces what is working and what is not.

CREDIT ASSIGNMENT PROBLEM

In reinforcement learning, the goal is to learn Π , each set of actions to achieve the goal. The agent can tell when it reaches the goal or even if it has gotten close to reaching the goal, just as a pitcher can see the results of a throw. But the agent generally does not know the utility of any intermediate steps or state—Was the windup good? Was the foot lifted too high or too soon? Ideally, a robot would like to rate the utility of intermediate states in order to build up a policy that says “Always lift your foot so high, and then, if the wind is head on, adjust the snap, ...” If reaching the goal is a reward, the learning program is distributing the reward “backward” to each state in the sequence in order to increase the utility rating of things that seemed to work. The difficulty in determining a numerical measure of the contribution of each state to reaching the goal is called the *credit assignment problem*. The credit

assignment problem asks: What state did the most work? Credit assignment is superficially similar to backpropagation as both work backward. But backpropagation decreases the error in generating an output while credit assignment increases the reward to states that directly lead to the output.

MARKOV DECISION PROCESS

Reinforcement learning algorithms usually assume that what is being learned is a *Markov decision process*. This means that the robot can choose any action that is possible from the current state, and the choice depends only on the current state. The choice of action is conditionally independent of previous choices. The pitcher can experiment with lifting a foot at any point in trying out a pitching sequence as long as it is possible to do so.

The major techniques for reinforcement learning are *utility functions* and *Q-learning*. Both are described below.

16.6.1 Utility Functions

UTILITY FUNCTIONS

Utility functions were employed in one of the original investigations of learning. In 1959 Samuels, not realizing that checkers is more computationally complex than chess, set out to create a program that would learn to play checkers. His checkers program was eventually successful and is still a foundation of machine learning. The key idea was to represent the current position of the pieces on the board, have the program learn the utility of that board position, and then, during play, search for the next best board position. The value of a board position was represented as a utility function, with a weight assigned to each aspect of the board position: the number of black pieces on the board, BB , the number of red pieces on the board, RB , the number of black kings, BK , the number of red pieces threatened, RT , and so on. The weights in the utility function are similar to the weights in a simple perceptron ANN, where the value V of a move resulting in a board position b is:

$$V^b = BB * w_1 + RB * w_2 + BK * w_3 + RT * w_4 + \dots$$

A major challenge in utility functions is determining the attributes, not so much the weights. ANN have a similar problem in that the inputs to the network must be informative.

16.6.2 Q-learning

Q-LEARNING

Utility functions are intuitively appealing, but, as Samuels found, it can be hard to generate a single function that expresses the value of a wide range of possible states. An alternative approach is simply to rank the utility of a state in reaching a goal. If lifting a foot at the beginning of a windup is usually associated with a good pitch, than that state should have a higher ranking at the beginning. The problem, once again, is how to do the credit assignment problem while avoiding representing, generating, and setting dozens of weights. One solution is *Q-learning*, the most common class of reinforcement learning algorithms, where the reinforcement for reaching a goal is distributed through the sequence. The utility of choosing an action at a state is then updated through many trial sequences

and eventually converges on an optimal policy.

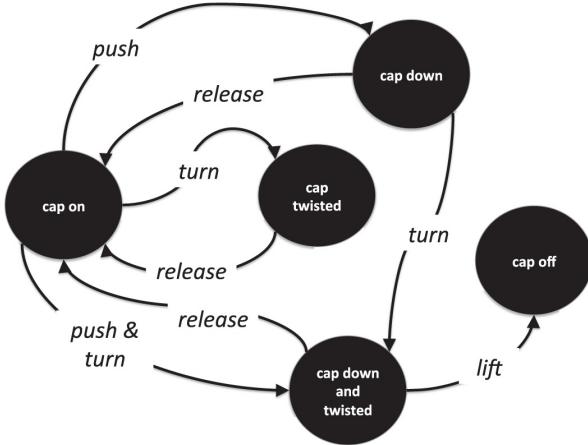
Q symbolizes a *Q function* that computes the expected utility Q of taking a specific action a when in state s . The most basic formula for Q-learning is:

$$Q(s, a) = R(s) + \gamma \text{argmax}_{a'}(Q(s, a')) \quad (16.1)$$

where $Q(s, a)$ is the estimated utility of being at state s and taking action a . The projected utility is the reward $R(s)$ for being in the current state plus $\gamma \text{argmax}_{a'}(Q(s, a'))$, which is the estimated contribution from taking the action a and once in the new state s' then taking the best possible a' .

The estimated utility has three parts. The first is the a priori reward $R(s)$ for reaching state s . In most cases, the reward will be known only for the goal state, which is why the credit assignment problem exists. The second term, the $\text{argmax}_{a'}(Q(s, a'))$, is a one-step look ahead to get the estimated utility of the next best step. The third is the discount function, γ , which incorporates the notion that the farther away an intermediate state is from a desired state, the less utility it probably has, and thus it should have a diminishing share of the reward or credit.

[Figure 16.6](#) provides an example of Q-learning. Consider a robot that is presented with a bottle and is tasked to remove the child-proof cap. A person might fumble around trying different actions until the cap is off and then tries it a couple of times more to make sure the policy works. In Q-learning, the robot does much the same fumbling around, though it is called state exploration.



Rewards Matrix

	push	turn	Push and turn	lift	release
cap on	0	0	0	-1	-1
cap down	-1	0	-1	-1	0
cap twisted	-1	-1	-1	-1	0
cap down and twisted	-1	-1	-1	100	0
cap off	-1	-1	-1	-1	-1

	push	turn	Push and turn	lift	release
cap on	0	0	0	-1	-1
cap down	-1	0	-1	-1	0
cap twisted	-1	-1	-1	-1	0
cap down and twisted	-1	-1	-1	100	0
cap off	-1	-1	-1	-1	-1

Rewards Matrix

	push	turn	Push and turn	lift	release
cap on	0	0	0	-1	-1
cap down	-1	0	-1	-1	0
cap twisted	-1	-1	-1	-1	0
cap down and twisted	-1	-1	-1	100	0
cap off	-1	-1	-1	-1	-1

Rewards Matrix

	push	turn	Push and turn	lift	release
cap on	0	0	0	-1	-1
cap down	-1	0	-1	-1	0
cap twisted	-1	-1	-1	-1	0
cap down and twisted	-1	-1	-1	100	0
cap off	-1	-1	-1	-1	-1

Rewards Matrix

	push	turn	Push and turn	lift	release
cap on	0	0	0	-1	-1
cap down	-1	0	-1	-1	0
cap twisted	-1	-1	-1	-1	0
cap down and twisted	-1	-1	-1	100	0
cap off	-1	-1	-1	-1	-1

Rewards Matrix

Q Matrix (end of pass 1)

	push	turn	Push and turn	lift	release
cap on	0	0	0	0	0
cap down	0	0	0	0	0
cap twisted	0	0	0	0	0
cap down and twisted	0	0	0	0	0
cap off	0	0	0	0	0

Q Matrix (end of pass 3)

	push	turn	Push and turn	lift	release
cap on	0	0	0	0	0
cap down	0	80	0	0	0
cap twisted	0	0	0	0	0
cap down and twisted	0	0	0	0	0
cap off	0	0	0	0	0

Q Matrix (end of pass 4)

	push	turn	Push and turn	lift	release
cap on	0	0	0	0	0
cap down	0	80	0	0	0
cap twisted	0	0	0	0	0
cap down and twisted	0	0	0	0	0
cap off	0	0	0	0	0

Q Matrix (end of pass 5)

	push	turn	Push and turn	lift	release
cap on	0	0	0	0	0
cap down	0	80	0	0	0
cap twisted	0	0	0	0	0
cap down and twisted	0	0	0	0	0
cap off	0	0	0	0	0

Figure 16.6 State transitions for removing a cap from a bottle with Rewards Matrix and Q Matrix.

Initially, the robot knows only the possible transitions between states and actions and the reward for choosing a particular action. The set of transitions are a Markov decision process. Recall that a Markov decision process is memoryless because it does not care how the robot arrived in the current state; it cares only about what to do now that it is there. The memoryless property is handy because the algorithm does not have to keep up with all the possible ways to reach a state. The Markov decision process could be visualized as a finite state machine diagram ([figure 16.6](#)) that expresses the possible transitions between the states and actions. In real life, there might be parameters for *push* to represent the robot pushing harder or softer, yielding a more complex set of transitions.

Returning to the child proof cap problem, a visible inspection of the diagram in [figure 16.6](#) yields

multiple policies to get the cap off. The two most direct ones are simultaneously to *push* and *turn* then *lift* the cap, or to *push*, then *turn*, then *lift*. *push* and *turn* then *lift* is intuitively appealing as the optimal policy because it requires only two steps, while *push*, then *turn*, then *lift* requires three steps. Thus the credit assignment process might favor shorter sequences.

The optimal policies will not be readily apparent or discovered by an *A** search algorithm in more realistic situations. Searching can produce paths, but, in this case, there are no costs associated with the links, which prevents a search algorithm from identifying the optimal paths. If there was a credit assignment process, the utility of each step in the sequence could be rated.

Another way to approach the problem of finding policies is to consider how a person addresses getting off a child proof cap for the first time. The person may fumble around, that is, explore the (*state, action*) space, trying different things and eventually getting the cap off to reach the goal state and eventually succeeds with the policy of *push, release, push and turn*. This policy is clearly not optimal, but it will work. Ideally, the person would realize that having *push* and *release* in the policy added no utility (i.e., apply credit assignment) and then try it again with just *push and turn* then *lift*, and see if that indeed worked (i.e., further exploration). If it did, *push and turn* then *lift* would be remembered as the simpler policy to use in the future. Alternatively, it might be discovered that two other policies, *push and turn* simultaneously or *push then turn*, allowed the cap to be lifted off. Or the person might only remember “do whatever until you get the cap pushed down and twisted, then you can lift.” Regardless of the policy, getting to the *cap down and twisted* state is the most valuable state to reach because the goal can be directly reached from that state.

Q-learning attempts to duplicate this type of human learning by exploring the (*state, action*) sequences and updating the estimated utility using the computer science strategy of dynamic programming. Q-learning duplicates the fumbling around by randomly exploring the (*state, action*) space. It duplicates the cognitive credit assignment process, that some states are determined to be extraneous or more valuable than others, by dynamic programming. Recall that dynamic programming is a programming technique that reduces computation by storing partial solutions or solutions to subproblems that it encounters as the program works towards solving a bigger problem. In this case, the subproblems are the estimated utility of each (*state, action*) for reaching the goal. If the Q-learning algorithm is allowed to explore the space over many trials, it will traverse through many policies, land on all the states, and, if there is a credit assignment method, estimate the utility of each of the steps. Eventually the algorithm will converge on the estimated utility of each state, which then leads to the optimal policy. If the estimated utility is known for each state, the agent knows what is the best action to take next if it starts with the *cap off*. Then when it takes the next step and enters a new state, it knows what is the best action from that new state, and so on. Notice that the agent does not have to remember what sequence to take, only what to do if it is in a particular state.

Q-learning handles the credit assignment problem with the discount function, γ . The general idea is that $Q(s, a)$ is the estimated utility of the best possible choice of the next state and action, or a one-step lookahead. Thus, the best possible choice of action is the maximum of $Q(s', a')$. But, intuitively, being at (s, a) is not as good as being at the best possible (s', a') , so $Q(s, a)$ should be less than $\text{argmax}(Q(s', a'))$. Therefore, we discount $\text{argmax}(Q(s', a'))$ by an arbitrary amount, say 80%, where $\gamma = 0.8$. If the exploration lands on a state *cap down and twisted* and takes an action *lift* that moves it to the goal and the goal has a reward of 100, equation 16.1 says the estimated value of $Q^*(\text{cap down and twisted}, \text{lift})$

is 80. Then the next time the exploration lands in a state and action whose best choice is *cap down and twisted, lift*, the value of that (*state, action*) is updated to 64.

16.6.3 Q-learning Example

Figure 16.6 formalizes the possible state transitions for removing a child proof cap from a bottle. The general method for removing the cap is to push down and twist simultaneously, then lift the cap off. Alternatively, the cap can be pushed down, then twisted, and finally lifted. These state transitions are known a priori.

REWARDS MATRIX

The a priori rewards and current estimated Q for each state and action are maintained in parallel matrices shown in figure 16.6. Note that, in this example, the only reward is for lifting the cap off as shown by the single entry of 100 in the *rewards matrix*. This is not uncommon; Q-learning is often applied to situations where only the end state is known. However, there can be rewards distributed through out the rewards matrix. Regardless, the reward matrix will remain unchanged during the execution of the algorithm. All entries in the Q matrix are initialized to 0 and are updated as the agent explores.

The algorithm is iterative:

```
for each episode
    select a random initial state s
    do
        randomly select an action a from
            the set of possible a for that s
        estimate the utility of the new state s'
        update the Q-matrix
    until goal state is reached
```

Q-learning presumes that the learning process will be unable to explore all possible combinations due to the large number of possibilities, and thus the designer specifies the number of episodes or trials. The more episodes, the more the learning process will converge on the optimal policy.

ONLINE LEARNING

Q-learning can be used for *online learning*, sometimes referred to as continuous learning, as the robot executes its normal routine using what it has learned so far. In online learning, the robot would use the current estimates of utilities in the Q-matrix to choose the next action, and, if all the actions led to states with a 0, then it would chose randomly and thus explore the space. This is in contrast to supervised learning where all the learning is offline, for example, learning to recognize different objects.

The algorithm starts an episode. In the case of learning to remove bottle caps, the only legal initial state is *cap on*. To start learning, *cap on* is s . Next, the algorithm enters the do loop. In the first pass through the do loop, the *cap on* row in the rewards matrix indicates that there are three possible actions: *push, turn, push and turn*. These three choices are highlighted in figure 16.6. Random selection chooses *push*, so the current (s, a) tuple is (*cap on, push*). Executing the action *turn* will produce a new state s'

of *cap down*. Now [equation 16.1](#) can be applied to estimate the utility of choosing *turn* from the *cap on* state; the corresponding element in the Q matrix is highlighted in [figure 16.6](#). Returning to the Rewards Matrix, the possible actions from the *cap down* state are $\{turn, release\}$, which are also highlighted.

$$\begin{aligned} Q(\text{cap on}, \text{push}) &= R(\text{cap on}, \text{push}) + \\ &\quad 0.8 \text{argmax}(Q(\text{cap down}, \text{turn}), Q(\text{cap down}, \text{release})) \\ &= 0 + 0.8 \text{argmax}(0, 0) \\ &= 0 \end{aligned}$$

The robot knows at this instant of time that by starting at *cap on* and then applying *push*, *push* has a utility of 0. However, *push* may have a higher utility that will be discovered later on. The Q matrix remains unchanged.

The robot is at the state of *cap down*, which is not the goal state of *cap off*, so the episode is not complete. Therefore the algorithm makes a second pass through the do loop. The current state is $s=\text{cap down}$, and the next step is to pick a legal action randomly from $\{\text{turn}, \text{release}\}$, which is highlighted in [figure 16.6](#). Let us say the selection is *turn*, which means the robot has explored a policy of starting at *cap on*, applying *push* and then *turn*. That leads to $s'=\text{cap down and twisted}$. The utility of the $(\text{cap down}, \text{turn})$ step in this sequence can be estimated with [equation 16.1](#):

$$\begin{aligned} Q(\text{cap down}, \text{turn}) &= R(\text{cap down}, \text{turn}) + \\ &\quad 0.8 \text{argmax}(Q(\text{cap down and twisted}, \text{lift}), \\ &\quad Q(\text{cap down and twisted}, \text{release})) \\ &= 0 + 0.8 \text{argmax}(100, 0) \\ &= 80 \end{aligned}$$

The Q matrix is updated to represent that the utility of applying *turn* from *cap down* is worth 80. The current state is *cap down and twisted*, which is not the goal, so the algorithm is at the top of the *do* loop for a third pass. Once again a random state is chosen, with $(\text{lift}, \text{release})$ as the options highlighted in [figure 16.6](#). Let us say the random selection is *release*, even though we can look at the states and know that *lift* would be better. Eventually the robot will figure it out. This puts the robot back into a state of *cap on*. The estimated utility of $(\text{cap down and twisted}, \text{release})$ is:

$$\begin{aligned} Q(\text{cap down and twisted}, \text{release}) &= R(\text{cap down and twisted}, \text{release}) \\ &\quad + 0.8(Q(\text{cap on}, \text{push}), Q(\text{cap on}, \text{turn}), \\ &\quad Q(\text{cap on}, \text{push and turn})) \\ &= 0 + 0.8 \text{argmax}(0, 0, 0) \\ &= 0 \end{aligned}$$

The Q matrix remains unchanged. The algorithm begins a fourth pass through do loop. The robot is now back at the *cap on* state with choices of $\{\text{push}, \text{turn}, \text{push and turn}\}$, highlighted in [figure 16.6](#). This time, the robot randomly selects *push and turn*, which returns the robot to the *cap down and twisted* state. The estimated utility of this choice is:

$$\begin{aligned}
Q(\text{cap on}, \text{push and turn}) &= R(\text{cap on}, \text{push and turn}) + \\
&\quad 0.8 \arg\max(Q(\text{cap down and twisted}, \text{lift}), \\
&\quad Q(\text{cap down and twisted}, \text{release})) \\
&= 0 + 0.8(0, 0) \\
&= 0
\end{aligned}$$

The Q matrix is unchanged even though we can see that the robot is really close to the goal. A fifth pass through the do loop begins. The current state s is *cap down and twisted* with choices of $\{\text{lift}, \text{release}\}$ highlighted in figure 16.6. This time it randomly selects *lift*. In this example the state diagram shows that there are no actions to be taken once the goal is reached, though in some applications, it is possible to reach the goal and then wander off instead of staying in that state (think of young children fumbling with a new task). The estimated utility of $(\text{cap down and twisted}, \text{lift})$ degenerates to the reward $R(\text{cap down and twisted}, \text{lift})$:

$$\begin{aligned}
Q(\text{cap down and twisted}, \text{lift}) &= R(\text{cap down and twisted}, \text{lift}) + \\
&\quad 0.8(\text{no states}) \\
&= 100 + 0.8(\text{no states}) \\
&= 100
\end{aligned}$$

The Q matrix is updated, and, since the $(\text{cap down and twisted}, \text{lift})$ reaches the goal state, the episode is complete. By creating a random walk through the $(\text{state}, \text{action})$ space of $\{\text{push}, \text{turn}, \text{release}, \text{push and turn}, \text{lift}\}$, the robot learns that if it is ever in the *cap down and twisted* state to chose *lift* and if it is in the *cap down* state it should choose *turn*. Further episodes will create different random walks and eventually the full Q matrix will converge.

16.6.4 Q-learning Discussion

In the above example, the robot had well-specified state transitions, but, in real life, all the states and state transitions may not be known until they actually occur. Returning to the example of a pitcher throwing a ball, the system does not need to explicitly express all the states in advance because they can be computed from the kinematics as needed, essentially—*Want to have the leg up and arm back? Well, the only physical motions for the leg are to lower it or raise it higher*. In that case, the Q-learning algorithm can dynamically build a table of transitions as it encounters them.

Equation 16.1 is the most basic Q-learning algorithm, but there are many more sophisticated variants. Some variants take into account that the initial estimate of utility may be optimistic and needs to be revised on subsequent visits to that state. Other variants use a learning rate α in conjunction with the discount rate to make learning more gradual so that the robot does not stop with the first policy it makes work successfully.

16.7 Evolutionary Robotics and Genetic Algorithms

EVOLUTIONARY ROBOTICS

Evolutionary robotics uses biomimetic processes to design a robot's shape (morphology), to have

the robot learn a new motor skill (like gaits), a task, or adapt to clutter in an environment. The idea is to duplicate evolution. Intuitively, evolution appears to be synonymous with learning because the agent is improving its fit with the world. A significant difference is that AI learning connotes how an individual agent learns from its own experiences, not how a population of agents learn over several lifetimes.

Evolutionary robotics typically employs genetic algorithms, artificial neural networks, or a combination to evolve the capabilities of the robot. Artificial neural networks have already been discussed; genetic algorithms will be described in more detail below. In practice, evolutionary robotics is only interested in duplicating genetic adaptation, not in understanding how agents' culture and society evolve, which is sometimes referred to as epigenetics.

GENETIC ALGORITHMS

Genetic algorithms (GA) follow the evolutionary process of *selective reproduction*, *random mutation*, and *genetic recombination* to satisfy a *fitness function*.¹⁶³ The symbol for a fitness function may follow the pattern recognition nomenclature of h but also may be written as ϕ .

GENOME

[Figure 16.7](#) illustrates the general process for using a GA to evolve a planetary rover that could navigate through a rocky desert to reach a sample return vehicle. As shown in [figure 16.7a](#), the rover has two binary target sensors that can see over the rocks, so if the return vehicle is in range, the rover will be able to see it. The rover has eight binary obstacle sensors. Suppose the rover has skid steering and uses a motor decoder that distills the inputs into three binary outputs to the motor. The genome for the rover is a vector of size 11 (8 inputs, 3 outputs) and the *genome* reflects a policy for controlling for the robot. The goal is to learn the mapping between the inputs and outputs. The set of inputs is $2^8 = 256$ and there are $2^3 = 8$ possible outputs. This means there are $8^{256} = 10^{231}$ possible associations captured by the vector to be searched. The large size of the search space motivates using a technique that samples the search space randomly rather than using a tree-based, branch and bound, search method, such as the *A*search* algorithm described in chapter 14.

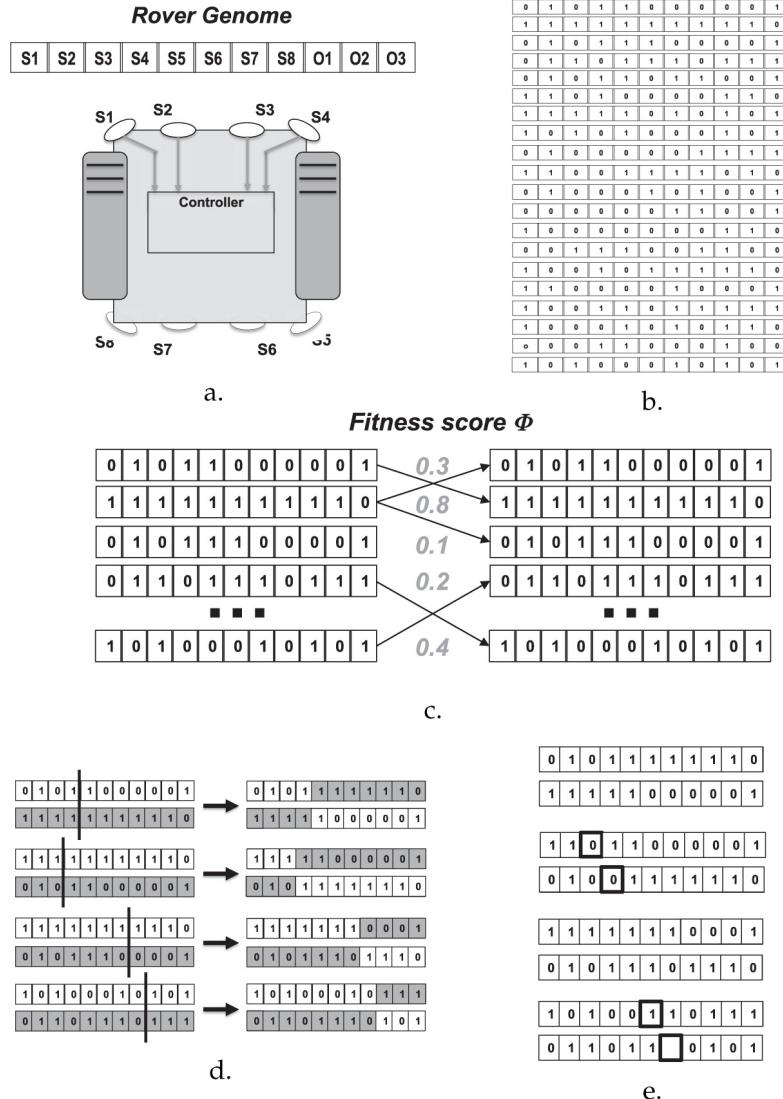


Figure 16.7 Genetic algorithm example: a.) The rover and its genome, b.) the members of the first generation, c.) associated belief mass, d.) resultant belief function, and e.) mutations.

The process starts with a generation of rovers operating in a reasonably similar rock field and the same target. A generation would consist of a population of individual rovers, each with a different genome. Each genome is a different configuration of the vector. The different configurations for each rover reflect diversity in the population. For this example, there are 20 different configurations of rovers, so that population is 20 rovers, each with a different genome, as shown in figure 16.7b. Each individual rover would be placed in a random location in the rock field and allowed to move according to its genome until it either reached the target, it hit an obstacle, or time ran out.

Each genome would receive a score for its performance using a fitness function ϕ . Designing an appropriate fitness function is similar to designing a heuristic function for a search algorithm (see

chapter 14) in that a good function leads to a better solution, but there is no equivalent admissibility criterion.

This example uses the fitness function from Keymeulen:¹⁰²

$$\phi = 0.5\left(1 - \frac{Distance_{remaining}}{Environmental_{Distance}}\right) + 0.5\left(1 - \frac{Number_{steps}}{Steps_{maximum}}\right)$$

This fitness function returns a scalar between [0,1], where 1 means a perfect score for fitness. ϕ gives 50% of the credit to how close the rover gets to the target through the term $\frac{Distance_{remaining}}{Environmental_{Distance}}$. $Distance_{remaining}$ is the distance between where the rover stops and the target; it is 0 if the rover reaches the target. $Environmental_{Distance}$ is an arbitrary measure of the overall area the rover has to traverse. Even if a rover does not reach the target, it gets a higher score the closer it gets to the target before it fails. The other 50% of the score is based on the number of steps, $Number_{steps}$, or cycles through the robot controller loop that were executed. This score is also an arbitrary measure of the maximum number of steps considered reasonable for the rover. $\frac{Number_{steps}}{Steps_{maximum}}$ approaches 1 as the overall $1 - \frac{Number_{steps}}{Steps_{maximum}}$ approaches 0, so a rover that takes fewer steps will get a higher score.

Each individual rover is tested repeatedly in the rock field. This might be done by placing an individual rover in a random location, letting it move until it reaches the target or time runs out, scoring it, and then moving it to another position and continuing. Suppose that there are 64 starting positions chosen randomly. An individual rover would therefore have 64 trials, while the whole generation of 20 rovers would have $64 \times 20 = 128$ trials. The genome for each rover would have a fitness score for each trial and an average fitness score over 64 trials as shown in [figure 16.7c](#).

CROSSOVER FUNCTION

After the average fitness scores for each of the 20 rovers has been calculated, the next step is to apply the selection genetic operator to the current population in order to create the next generation of rovers through recombination. The *selection* operator selects pairs of rovers and then swaps portions of their vectors using a *crossover function*. As shown by the arrows in [figure 16.7c](#), the pairs of rovers are selected semi-randomly, where higher scoring rover configurations are paired more frequently. Using an animal husbandry analogy, a high scoring rover is put out to stud. Selection algorithms eliminate very poor performers but generally do not try to eliminate all the low-scoring genomes because there may be something valuable in their configurations if they were combined later with another configuration or if the rover encountered a different environment; the assumption is called the “selfish DNA hypothesis.”¹⁶³

Once the pairs are selected, the crossover function determines which genes to swap based on the location of the crossover point for the pair. The crossover point for a pair can be selected randomly. It is shown by the vertical lines in [figure 16.7d](#).

The mutation genetic operator can be applied as well. Mutation helps the process from becoming trapped in a local maximum, where the population may not be changing enough or changing only at the larger chunks created by the crossover point. [Figure 16.7e](#) shows that four rovers had mutations.

Once selection, recombination, and mutation have been applied, there is a new generation of rover genomes. The process can let the populations grow or enforce population control and keep each

generation at 20 rovers.

The next generation of rover genomes goes through the same process with the same 64 starting locations, applies the selection, recombination, and mutation operators, and so on. In Kermuellen, a useful robot controller that could navigate a cluttered area was evolved in 20 generations, though, in general, a heuristic is that 100 generations are required to see a convergence.¹⁶³

The above example highlights the two biggest problems with evolution: it takes a long time and is impractical to use with physical robots. A designer would have to plan to run the rovers $1,000 \times 128 = 12,800$ times. A rover may be damaged by running into a rock, which is a concern if the designer has only one robot for testing and is programming 20 different configurations. An unfit configuration could ruin the rover and stop the evolution. Even if the rover is impervious to damage, it might bump rocks out of position or otherwise change the test environment, so the designer not only has to spend time setting up the next rover configuration, but also has to reset the field. If it took the designer five minutes to set up the robot and reset the field, then 12,800 trials would take 1,066 hours or more than 44 days working nonstop to complete the evolution.

These two problems explain why evolutionary robotics is almost always conducted in a computer simulation. However, recall from chapter 10 that sensor readings in complex environments can be very hard to simulate correctly. A set of rovers evolved in simulation may not perform as well as expected in the physical world. The rover may have hidden vulnerabilities because the simulation did not capture some quirk or rare interaction of sensing in the physical world.

Fortunately, genetic algorithms have a second, perhaps more practical, application to robotics: they can also be used to create a test-set of states for another algorithm. For example, instead of using the genetic algorithm to generate genomes in order to converge on a controller that satisfies the fitness function h , the algorithm generates test cases where the fitness function captures how different or challenging the inputs are. For example, there may be 1,000 combinations of rocks and hills. Another strategy is to use genetic algorithms to produce states that become the input to another algorithm. For example, given this combination of sensor readings, adjust the neural network so it produces the right response.

16.8 Learning and Architecture

The previous sections have described types of learning but have not described where the algorithms would reside within a robot architecture. Where learning goes in the canonical hybrid architecture is often irrelevant because most learning is applied offline, that is, in simulation, rather than online, where the robot learns as it goes through its routine. Online learning can be located anywhere in the canonical architecture because the robot can be learning reactions, deliberative functions, or interactive functions, or some combination.

CRITIC

LEARNING ELEMENT

PERFORMANCE ELEMENT

PROBLEM GENERATION

Online learning itself can have an overall systems architecture, as proposed by Tom Mitchell.¹³² Figure 16.8 presents the learning architecture in terms of the four robot primitives: SENSE, PLAN,

ACT, and LEARN. The architectural flow starts with the robot sensing the world and creating the percept. That percept may be used by itself or for updating the world model, including symbol-ground processing. The percept or updated world model, including what was known a priori, is fed into a *critic*. A critic is a dedicated agent that identifies what is wrong with, or could be improved about, the current state of the world relative to goals or expectations. A critic can be thought of as an error signal, for example, that the object was incorrectly classified or that the pitcher's throw landed low and to the left, but the critic can be more sophisticated. The output of the critic goes to the learning element providing feedback. The *learning element*, or algorithm that actually conducts the learning, takes the input and adjusts the performance element and possibly creates new problems. The *performance element* is the component that actually uses the learning. In Q-learning, the performance element would be the policy. The learning element may need the robot to stop what it is doing to create its own miniature simulation of problems or state explorations. For example, the robot may pause to practice a new skill. In cloud-based robotics, the robot may access the Internet to attempt to find other examples of an object and learn how it has been labeled by other systems. This exploration, either online or offline, would be enabled by the *problem generation* component.

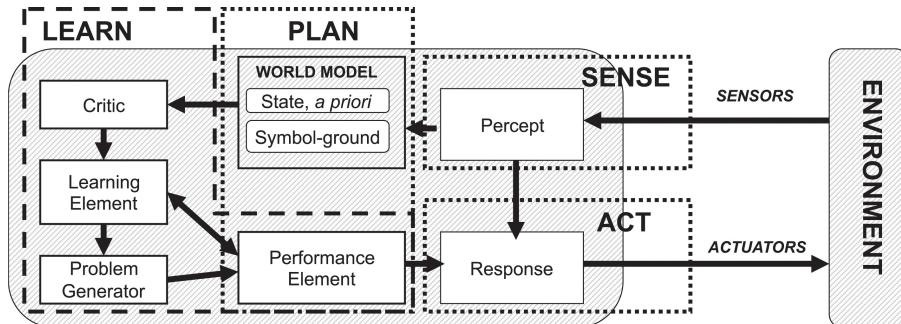


Figure 16.8 Mitchell's learning architecture¹³² adapted for AI robotics.

A LEARN instance can be reactive or deliberative: there is no centralized “learn everything” module. Different types of learning are useful for different functions within a robot, and each instance of learning would be implemented following the same general architecture. The learning architecture does not require that the critic deliberatively monitor and reason about the actual state of the world and compare it to the idea state of the world. A reactive error signal, such as seen by Genghis learning to walk from inputs of moving forward and being level,¹¹⁹ is a critic. Likewise, before resuming operations, the robot does not have to generate new problems to try out what it has just learned.

16.9 Gaps and Opportunities

The 2012 Defense Science Board study on autonomy for unmanned systems¹⁴⁴ provides important insights into machine learning and the remaining gaps in autonomous systems. The highlights are:

- Machine learning has been primarily applied to unmanned ground vehicles, and less so to aerial and marine vehicles.

- The applications have been either unstructured static environments, such as autonomous driving cross-country in the desert, or structured dynamically changing environments, such as autonomous driving in a city with other cars and pedestrians. But few applications have been developed for unstructured and dynamic environments.
- The most commonly used type of learning by example in unmanned systems is supervised learning. A deterrent to using supervised learning is that constructing the training sets requires significant human effort.
- Reinforcement learning requires too many trials to be practical but the process can be sped up with learning by imitation, where a human chooses the sequence of states, or where the human gives advice; in effect the human is the *critic*.
- Machine learning is generally used for recognizing objects or learning policies, but it is not used so much for anomaly detection and situation awareness or learning a *new term*. Recognizing that an element of the environment is different or that the robot may be driving into a trap has not been addressed and remains a hard problem for machine learning to deal with.

16.10 Summary

Learning by example is just one type of learning, but it is the one that has been most successfully applied to robotics. One major application of learning objects or terrain is usually through supervised or unsupervised learning where large numbers of training examples are presented to the robot offline. Creating the training set can be difficult as the number of examples and order of presentation can impact learning. Worse yet, an algorithm can overfit h where it works perfectly for the training set but produces errors with new data. It is important to validate learning with new data, not to reuse the same examples that were used in training. Another application of learning by example is to learn actions or policies that capture what action to take when. Policies are usually created with Q-learning. Learning a “new term” is extremely difficult as most learning algorithms make some sort of a priori assumption about what is being learned; for example, that there are five types of fruit to be learned; that undermine true new term learning; for example, that there are really six types of fruit in the imagery.

Learning can be applied to any aspect of a robot’s performances and is not restricted to deliberative or reactive layers. Online learning is rare; the norm is offline learning, which is learning in simulation. The simulation may be a set of training examples given to an artificial neural network or the robot may be dedicated to exploring a state space using Q-learning. When implementing online learning, it is helpful to think of learning as having four architectural components: a critic, learning element, problem generator, and performance element.

Returning to the questions posed in the introduction, the first question was: *What is learning?* It is improving performance on future tasks after making observations about the world. The chapter answered *What are the types of learning?* by noting that there are many types of learning, but learning by example is the most common in robots. Learning by example can be subdivided into four major types: unsupervised, supervised, reinforcement, and semi-supervised, though semi-supervised learning is still a new subfield and rarely used. The chapter also addressed *Why are robots not using neural networks all the time? They are like the brain, right?* Artificial neural networks are indeed like the

brain, but they are currently a simplification of neural structures. The algorithms often require hundreds to thousands of examples for training purposes, plus some way of specifying the number of hidden layers, so the algorithms can be challenging to implement. ANN are primarily used for recognizing or classifying objects or concepts. In effect, they are for learning perception and symbol grounding, not for learning policies or sequences of actions. The final question was: *Where does learning go in our operational architecture?* Since there is no single component that is to be learned, there is no single place in the architecture for “learning.” Instead, different learning modules may have instances throughout the reactive, deliberative, and interactive layers.

16.11 Exercises

Exercise 16.1

Describe the four major types of learning by example (supervised, unsupervised, semi-supervised, and reinforcement learning). List a technique for each.

Exercise 16.2

Define:

- a. critic
- b. new term problem
- c. overfitting
- d. credit assignment problem
- e. online learning.

Exercise 16.3

Create a table describing the differences in representation between the following learning techniques: induction, support vector machines, Q-learning, and artificial neural networks.

Exercise 16.4

Consider the artificial neural network in [figure 16.9](#) for classifying whether a logo is for Texas A&M University College Station or for Texas A&M University Corpus Christi based on the color (maroon versus blue).

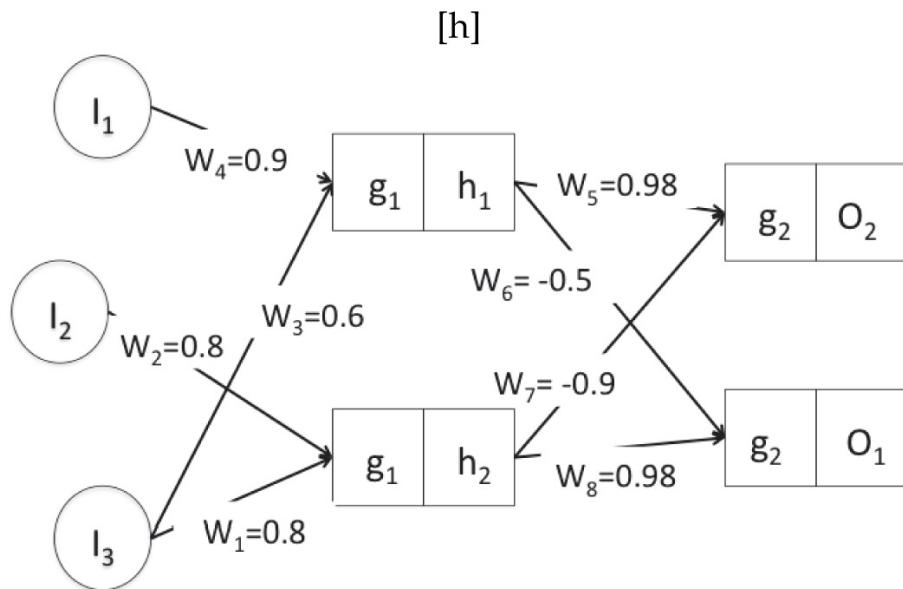


Figure 16.9 ANN for classifying logos by color.

Compute the output of the network, if $I_1 = 1$, $I_2 = 0.3$, and $I_3 = 0.7$, for the activation functions of $g_1 = 1/(1 + e^{-x})$ and $g_2 : 1if x \geq 0.25; 0if x < 0.25$.

Exercise 16.5

Describe the difference between genetic algorithms and learning.

Exercise 16.6

Complete the Q-learning example in section 16.6.3.

Exercise 16.7

Given the rewards matrix in figure 16.10 with G as the goal, chose three random paths (your choice) through the matrix, and show the updated U matrix at each step. Show your work.

	F	T	B	G
R	0	0	-1	100
	F	T	B	G
T	0	0	0	-1
B	-1	0	0	100
G	-1	-1	-1	-1

	F	T	B	G
Q	0	0	0	0
	F	T	B	G
T	0	0	0	0
B	0	0	0	0
G	0	0	0	0

Figure 16.10 Rewards and U matrix.

Exercise 16.8

Discuss the challenge of generating the right training sets of adequate size for learning. For what robotic applications do you think learning is feasible?

Exercise 16.9

Discuss why learning is not “the” solution to creating intelligence in robots.

Exercise 16.10

How is a policy different from a behavior? Are they similar?

16.12 End Notes

Always keep learning.

Marvin Minsky and Seymour Papert were leading researchers in artificial intelligence at MIT. When the concept of artificial neural networks became popular in AI, they concentrated on perceptrons and looked at what could be done with binary activation functions. Their conclusion was that perceptrons were extremely limited, and thus ANN were a dead end. This effectively killed ANN research as no one could come up with a different ANN topology that would avoid the perceptron problem. Decades later, a new generation of AI researchers challenged the notion that ANN were a dead end and returned to the original work of David Rumelhart in modeling neurons. They discovered that other activation functions, such as the logistic function, were not only possible but had been documented in biology. ANN research became popular again, and Marvin Minsky often apologized in his talks for killing, or at least delaying, ANN research.

Learning the hard way.

In the late 1980s and early 1990s, robotics researchers redoubled their efforts in machine learning. Several researchers started with the goal of teaching the robot to avoid obstacles. The work led to hundreds of collisions. Behavioral roboticists pointed out that animals often have hardwired behaviors that require very little learning, for example the babies of prey animals start trying to walk immediately and quickly fine tune their movements in order to escape predators. Given the cost of robots and the damage that they can do to walls (and people), obstacle avoidance seemed like an example of something that should be built into the robot rather than learned by the robot only after extensive trials.

Apologies and thanks.

The genetic algorithm rover example was inspired in part by a tutorial I found on the Web that was aimed at a general science audience; even though I did not duplicate it, I wanted to refer to it because it was a great tutorial. When I went back to verify the references, the tutorial was no longer posted and, even with a historical search, I could not find it (which is why I always tell my students to capture the webpage at the time). If you see the similarity to a tutorial or your work, please let me know and I will immediately update the list of corrections for the book! And thank you for writing a tutorial!

IV

Interactive Functionality

Contents:

- [Chapter 17: MultiRobot Systems](#)
- [Chapter 18: Human-Robot Interaction](#)

17

MultiRobot Systems (MRS)

Chapter Objectives:

- List and describe seven challenges in designing multirobot systems.
- List and describe the key aspects of a *multirobot system task* that impact the choice of a multirobot system.
- List and describe the components of the *coordination dimension* and *system dimensions* that should be considered when designing a multirobot system.
- Name the five most common multirobot teams in multirobot systems and give at least two examples of each.
- Describe the differences between *indirect communication* and *direct communication* and give examples of each.
- If given a description of an intended task, a collection of robots, and the permitted interactions between robots, design a multi-agent system, and describe the system in terms of task, coordination, and system dimensions.
- Give examples of multirobot systems which form *collective swarms* or are *intentionally cooperative*.
- Define *swarm robotics*, *distributed problem solving*, *sensor networks*, *reconfigurable robots*, *cloud robotics*, *stigmergy*, *task interference*, *contract net protocol*, and *networked robots*.
- Explain the impact of *ignorant coexistence*, *informed coexistence*, and *intelligent coexistence* types of *societal rules* for emergent multirobot systems behavior.

17.1 Overview

MULTIROBOT SYSTEMS

Collections of two or more mobile robots working together are referred to as *multirobot systems* (MRS). The chapter addresses common questions about multirobot systems. The most fundamental question is: *What are multiple robots good for?* Another frequently asked question is: *Are there different types of multiple robot teams—teams that act like insect swarms and teams that use more*

complicated intelligence? The chapter will cover how multirobot teams vary in terms of task, coordination, and system dimensions. Any mention of swarms eventually leads to: *Can you ever have too many robots?* and the chapter introduces the concept of task interference. Designers are often concerned with: *How is programming multiple robots different from programming a single robot?* The chapter reframes this question in terms of the three layers in the canonical operational architecture.

The chapter begins by listing the opportunities or advantages offered by a multirobot system but also notes the associated challenges in realizing those advantages. Next, the chapter places multirobot systems within the larger artificial intelligence context of multi-agent systems and distributed AI. With the intellectual foundations now in place, the chapter turns to the details of designing a multirobot system. It first talks about matching a multirobot system to a task and then explains how to design the coordination and system attributes of the specific platforms. The tasks, coordination, and system possibilities are sizable, but the chapter reviews the five most commonly occurring MRS. The chapter then provides overviews of different strategies for programming a team, referring to the layers in the operational architecture introduced in chapter 4. The chapter concludes with a summary and review of the answers to the common questions, and ends with a list of open issues in AI for multirobot systems.

17.2 Four Opportunities and Seven Challenges

Novices studying multirobot systems often underestimate the opportunities or challenges because they consider only insect-like collectives or try to duplicate high-performing human teams. Rather than only considering biological inspiration, this section attempts to describe the advantages and challenges of all types of multirobot teams.

17.2.1 Four Advantages of MRS

SWARM ROBOTS

Mulirobot systems are desirable for at least four reasons given in¹⁶⁶ and expanded upon here. One reason for MRS is that multiple robots may be cheaper or faster than a single robot. Like ants and other insects, many cheap robots working together can replace a single expensive robot, making multi-agents more cost effective. In the case of planetary explorers or removal of land mines, many cheap robots should be able to cover more area faster. Indeed, the term *swarm robots* popularly refers to large numbers of robots working on a single task. Swarm robots often duplicate the mechanisms that insects, fish, and birds use to generate intelligent emergent behavior from relative unintelligent individuals.

A second reason for MRS is that, for some tasks that are too complex for a single robot, the complexity can be reduced by robots working in parallel. Returning to the planetary exploration example, it is hard to imagine a single robot mapping an entire planet. But mapping a planet with many smaller robots seems more feasible. However, robots working in parallel does not always mean that they are simply instances of the same robot performing the same behaviors at a different location. A set of robots could be searching a planet while another set of robots built a base for future human habitation, as described in fiction in Kim Stanley Robinson's award winning science fiction book "Red Mars."

A third reason why MRS are desirable is robustness through redundancy: if one robot fails or is destroyed, the other robots can continue and complete the job, though perhaps not as quickly or as

efficiently. In a famous technical report entitled “Fast, Cheap and Out of Control,”³¹ Rodney Brooks at MIT proposed that NASA send hundreds of inexpensive ant-like reactive robots to Mars. Brooks argued, in part, that having many robots meant that several robots could be destroyed in transit or during landing without having a real impact on the success of the overall mission. This conclusion echoes the land mine example: if one cheap robot in a swarm gets blown up, the demining can still continue.

Finally, some tasks are inherently distributed and thus require MRS. A sports team consists of distributed individual agents who have specialties, or roles, to play. For example, in soccer it is hard to imagine combining the goalie, strikers, and defenders into one robot because the robot would have to be in multiple places at once. But it is important to remember that an individual may be able to change its task or role to take over another position, because people expect robots to have the same adaptability. In relay races, the task is divided into roles and actions that must occur in a sequence and must be performed by individuals; this creates a temporal interdependency between the agents. How to divide up responsibilities among individual robots in a multrobot system is generally referred to as *task allocation*.

17.2.2 Seven Challenges in MRS

While MRS offer new opportunities for robots, they also pose new challenges in programming intelligence into robots because interaction involves more than creating a single competent robot. Since the 1980s, researchers, such as Arkin,¹¹ Bond and Gasser,²⁶ Brooks,²⁹ Oliveira et al.,¹⁶⁴ and Parker,^{166;165} all cite the following problems with teams of multiple agents:

1. *The fundamental system design is harder.* A designer has to decide between making the individual more intelligent or the group more intelligent. The designer also has to recognize the characteristics of a problem that make it suitable for multi-agents.

EMERGENT SOCIAL BEHAVIOR

2. *The choice of task allocation mechanisms is hard.* Either the designer preprograms or the agents themselves divide up the task, generate an MRS plan, and select which members to assign tasks to. This raises the question of how to implement the allocations as behaviors, behavioral coordination mechanisms, or deliberative functions. Individual members of multi-agent teams are usually programmed with behaviors according to either the Reactive or Hybrid Deliberative/Reactive paradigms. Recall that under the Reactive Paradigm, the multiple behaviors acting concurrently in a robot led to an *emergent behavior*. For example, a robot might respond to a set of obstacles in a way that was not explicitly programmed. Likewise, in multi-agents, the concurrent—but independent—actions of each robot leads to an *emergent social behavior*. Group behavior can be different from individual behavior, emulating “group dynamics” or possibly “mob psychology.”

3. *It is not clear when communication is needed between agents or what to say.* Although animals can explicitly communicate, for example songs in birds, and signals like a deer raising its tail to display white, many animals flock or school, maintaining formation without explicit communication. Formation control is often done simply by perceiving the proximity to, or actions of, other agents; for example, schooling fish try to remain equally close to fish swimming on either side. But robots

and modern telecommunications technology make it possible for all agents on a team to know whatever is in the mind of the other robots, though at a computational and hardware cost. How can this unparalleled communication ability be exploited? What happens if the telecommunications link goes bad? Cell phones are not 100% reliable, even though there is tremendous consumer pressure on cell phones. There are many economic incentives for robot communication, so it is safe to assume that robot communications will not be completely reliable. Is there a language for multi-agents that can abstract the important information and minimize explicit communication?

4. *The “right” level of individuality and autonomy is usually not obvious in a problem domain.* A major design decision is specifying who will be in charge, for example, a central leader, distributed leadership, or no leadership.¹⁶⁶ Agents with a high degree of individual autonomy may create more interference with the group goals, even to the point of seeming “autistic.”¹⁶⁴ But agents with more autonomy may be better able to deal with the open world.

TASK INTERFERENCE

5. *There is the potential for task interference: the “too many cooks spoil the broth” effect.* Having more robots working on a task or in a team increases the possibility that individual robots might unintentionally interfere with each other, thereby lowering the overall productivity.
6. *Monitoring a MRS is hard, thereby making it difficult for a team to recognize when it, or its members, are unproductive.* One solution to the “too many cooks spoil the broth” problem is to try engineering the team so that interference cannot happen. One example of such engineering is to have a single, centralized controller. But this may not be possible for every type of team. Also, there are risks that communication with the centralized controller will be lost or that it will be destroyed. To defend itself, the team should be capable of monitoring itself to make sure it is productive. This, in turn, goes back to the issue of communication.
7. *Provably correct designs of behaviors and testing that can cope with behavioral non-determinism are harder to produce.* A major challenge in the design of MRS is the lack of tools to predict and verify social behavior.

17.3 Multirobot Systems and AI

The success of multirobot systems relies heavily on AI research. MRS may use knowledge representations, learning, inference, search, planning and problem solving, and computer vision. But more significantly, artificial intelligence researchers generally view MRS as a subset of the key area of *distributed artificial intelligence (DAI)*. Figure 17.1 shows the relationship of MRS and related research to distributed AI.

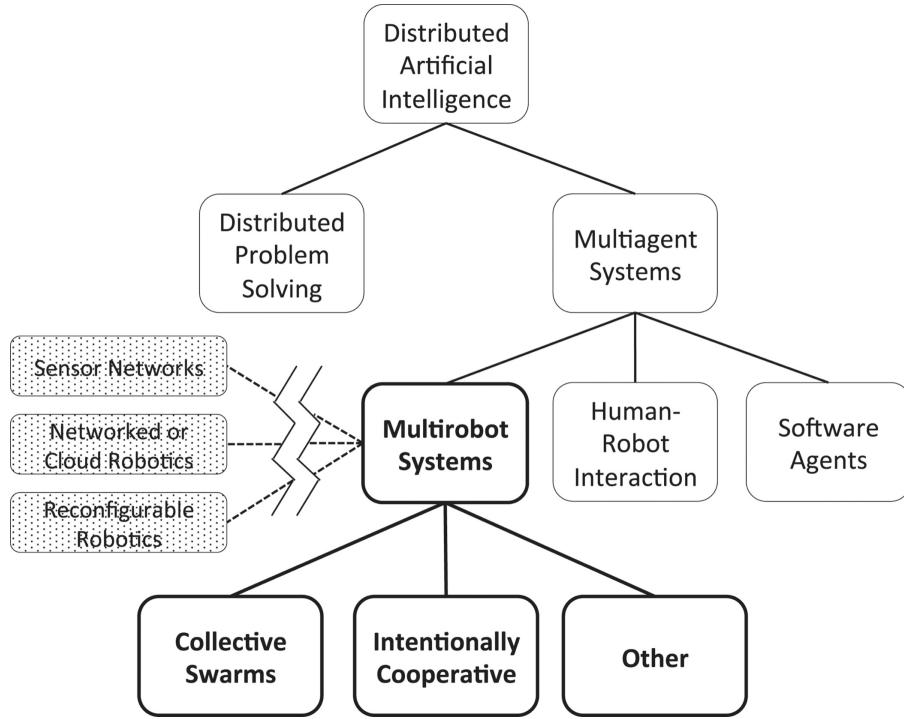


Figure 17.1 Relationship between multirobot systems research and other topics in artificial intelligence and robotics.

DISTRIBUTED PROBLEM SOLVING

MULTIAGENT SYSTEMS

Following figure 17.1, multirobot systems are a specialization of distributed AI research. Distributed AI typically focuses on solving problems or meeting goals through “divide and conquer” strategies. Following Stone and Veloso,²⁰⁰ distributed AI work can be subdivided into research that focuses on distributed problem solving or multiagent systems. *Distributed problem solving* tries to determine the best way to decompose a task or synthesize a solution from solutions contributed by multiple agents. *Multiagent systems* research is about deciding how to divide up the task or responsibilities among agents, in particular, identifying which agent should be assigned what task and how the agents cooperate. Distributed problem solving can be performed on a single computer with software agents. A multiagent system may make use of distributed problem solving algorithms that would be allocated to software agents on different computers or robots.

Multiagent systems research is independent of whether the agent is a software agent, a physically situated agent, or a human. As there are differences between agents that reside in the Web and those that function in the physical world, figure 17.1 shows that multiagent system research is further subdivided, with robot agent research called *multirobot systems (MRS)*.

Even the area of multirobot systems is too big and can be further subdivided into two main camps: *collective swarms* and *intentionally cooperative*. However, as will be seen in this chapter, there are other possible variants of multirobot systems based on the style of coordination and the physical implementation of the system.

There are three areas that are associated with multirobot systems that would not necessarily appear in a book or conference on MRS:

SENSOR NETWORKS

- *Sensor networks* are large numbers of sensors, that often pan/tilt, zoom or physically move and reposition themselves in order to accomplish a joint activity. Sensor networks are sometimes considered MRS if the sensors are mobile robots; either the robots act as the sensors or robots place the sensors at desired locations. However, sensor networks might not be found in the literature on MRS because the hard problems in sensor networks are related to the spatial reasoning needed to maintain sensor coverage of the desired area, not the overall intelligence needed to coordinate the system. For example, a sensor network may need to be dynamically adapted, either because a node or robot may break over time or become occluded by some object moving in front of the sensor or because the robot is tracking a moving object.

RECONFIGURABLE ROBOTS

- *Reconfigurable robots* consist of multiple modules, each of which might be treated as an independent robot that can link up to create an alternative version of the robot. Reconfigurable robots are similar to the robots in the Transformers® TV and movie series. One of the first such systems was called CEBOT for “cellular robot system,”³⁶ and consisted of small identical robots that hooked up to form a useful robot. Some robots, such as CONRO, resemble Transformers in that a robot changes its modules from a long snake into a spider configuration to move through the environment better. Other robots look more like a lattice of blocks that changes shape to grow a part, such as an antenna that adapts to the wireless environment. Other robots look like a netbook-sized version of flat sheets that can fold and unfold like a protein. Reconfigurable robots are similar to MRS in that there are multiple modules that have to work together. However, in practice, the algorithmic challenges tend to limit adaptation. A good overview can be found in Yim, Shen, Salemi, et al.²²²

NETWORKED ROBOTS

CLOUD ROBOTICS

- *Networked robots* or *cloud robotics* focus on how communication networks can enable functionality in a single robot. The challenges for these systems are creating and maintaining the networks and enabling the robot to access the Web to obtain resources or data from other robots in order to accomplish its mission. For example, a sensor network may have so many nodes and has such large volumes of information that it presents a high potential to overload communications; therefore it is important to have efficient routing and throughput algorithms. McKee provides an over-view.¹²⁶

17.4 Designing MRS for Tasks

MULTIROBOT SYSTEM TASK

In addressing the overarching question for this chapter of *how to design a multirobot system*, it is useful to return to the concept of an ecological niche. Recall from chapter 6 that the ecological approach to design identifies the niche that the robot will work in. The niche is defined as the robot’s task, its

environment, and its capabilities as an agent. This section focuses on methods for defining the task in ways that help clarify design requirements. Balch²⁰⁸ describes a multi-agent system task, which reduces to a *multirobot system task*, in terms of four axes: the expectations of the *time* it will take to accomplish the task, the *subject of the action*, the constraints upon the *movement*, and any implicit or explicit *dependencies* between the robots as they execute the task.

17.4.1 Time Expectations for a Task

One advantage of a MRS is that it may be faster; therefore it is useful to consider the expectations about the length of time the robots can take to accomplish a task. These expectations can be grouped into four categories:

- *Fixed time*: a task where the MRS should get as much done on the job as it can before a deadline.
- *Minimum time*: a task where the MRS should perform the job as fast as it can.
- *Unlimited time*: the MRS is not constrained by time and can take as much time as it needs to do a quality job.
- *Synchronization*: the members of the MRS have to synchronize their arrival at the same place at the appointed time or perform a joint task such as box pushing.

To see how time impacts the design of a MRS, consider the “Call a Conference” task in the *Robots Alive!* episode of Scientific American Frontiers. In that competition, a robot had to search an office building to find an empty conference room and then go to each person who needed to be at the meeting and let them know when and where the meeting was. The winner was the robot that was the fastest at finding the empty conference room and then correctly estimated how long it would take to find all the participants. The team from SRI International used multiple robots to perform the task. The task was a *minimum time* task because the primary score was how fast the task could be accomplished. The task could be considered a *synchronization* task since the robots shared their information about where the empty conference room was, but the robots were not tightly coupled. The SRI design concentrated on programming each robot so that it could complete their portion of the task as fast as possible so that the overall task was completed in the minimum time.

Another example of how time impacts the design of a MRS is humanitarian demining where a group of robots scours the terrain for land mines. In this case, the time constraint is *unlimited*. The goal is for the robots to be thorough, no matter how long it takes, as missing a land mine may cost people their lives.

17.4.2 Subject of Action

A second influence of task on design of a MRS is the *subject of action*. There are two possible subjects of action for a MRS. One subject of action is an object (*object-based*), such as robots playing soccer in RoboCup where the ball is the subject. The other subject of action is the robot itself (*robot-based*), such as a mapping task where the robots have to localize and map the world. In soccer, the subject of the task is the ball. The robots direct their motions relative to the ball. Robot motions relative to obstacles and each other are secondary to the greater goal of moving the ball. In collaborative mapping, the task

is about the location of the robots. If the robots do not know where they are and cannot keep track of their motions relative to each other, then they cannot stitch the map together or make sure that they are not duplicating the area of coverage.

To see how the subject of action impacts the design of MRS, return to the “Call a Conference” task. The task is *robot-based* because the competition is essentially a mapping task. The robots need to know where they are to accomplish the task. Their actions are not object-based even though the goal is to find a conference room and then locate the professors. Their actions are relative to where they have been and where the other robots have been.

Humanitarian demining is also usually *robot-based*. The goal for most implementations is to cover an area, so the challenge is determining how the robots should move and what areas they should cover. The collective may act like a flock of sheep moving randomly about, which is one strategy for covering an area.

17.4.3 Movement

The desired *movement* to accomplish a task is a third category for MRS design. Movement can be divided into four categories:

- *Coverage*: the collective spreads out to cover as much area as possible,
- *Convergent*: the collective will eventually converge or meet at a location or object,
- *Move to*: the members of the collective may be starting in different places but all are moving to a single point, and
- *Movement while*: the collective moves while maintaining a positional constraint, which is usually a specific formation, such as moving in a line, a column, a diamond or a wedge.¹⁶

In the “Call a Conference” competition, the movement for the task was *coverage* because the competition was trying to search the office building area to find an empty conference room as fast as possible. Humanitarian demining would likely be a coverage type of movement as well, although an implementation, where the robots were going in a line across a field, would fall into the *movement while* category.

17.4.4 Dependency

The fourth axis of a multirobot system is dependency between the members of the collective. Members can be *independent*; the robots do not have to work together and they do not have to be aware of each other. They can be *dependent* when multiple robots are needed to accomplish the task, like box pushing when one robot cannot do it alone, or the robots have to be aware of each other, for example, when two robots are needed to carry a heavy object. Members can also be *interdependent* when there is a cyclic dependency, such as a resupply operation, where the primary robot is performing a task but needs another robot to bring fuel to it from the home base. The primary robot cannot complete its task until the refiller robot brings the fuel. The refiller robot cannot return to the home base until it resupplies the primary robot. So, despite the fact that the two robots have different tasks, one robot cannot complete its task until the other robot completes its. Thus they are interdependent.

Dependency may be flexible and be based on how the designer conceptualizes the task. The task in the “Call a Conference” competition was clearly dependent because the robots divided the task among themselves. However, a humanitarian demining task does not require a specific type of dependency. If the designer wants the robots to wander and cover the area randomly like sheep, the members would be independent. If the designer wanted the robots to sweep the area in a line, then the task would require the robots to be dependent.

17.5 Coordination Dimension of MRS Design

The task, along with the environment and the robots’ capabilities, establishes the ecological niche but it does not automatically help establish ways robots can interact with each other. Farinelli, Iocchi, and Nardi⁷⁵ conceptualize such interaction along two dimensions: *coordination* and *systems*.

COORDINATION DIMENSION

The *coordination dimension* captures the design decisions that influence how the team members interact; it essentially looks at what makes the collective a MRS rather than just a group of robots that happen to be collocated.

Farinelli, Iocchi, and Nardi⁷⁵ identify four components of the coordination dimension to consider in the design of a MRS:

- *Cooperation*. MRS can either *cooperate* or *compete* to accomplish a task. MRS that cooperate are systems where the robots explicitly work together to accomplish a global task, such as lifting an object. MRS that compete involve robots that are competing to accomplish a task, and the robot that does it first, wins. Competitive MRS are often used for searching or foraging tasks.
- *Knowledge*. Individual robots may be *aware* or *unaware* of other robots in MRS. For example, a robot can search for an object. When robots explicitly divide up the work and talk to each other (i.e., cooperate), they are aware. When many single-minded robots independently set out to do the same task (i.e., compete), they may be unaware or aware. Usually competing robots treat other robots no differently than other moving objects; this is clearly an example of unaware coordination. If a robot is able to recognize other robots and changes how it accomplishes its task based on what the other robots are doing, such as a bird adjusting its position within a flock flying in V-formation, it is aware. Robots can exhibit the aware category of coordination even though they are not communicating, sharing information, or explicitly cooperating.
- *Coordination*. Coordination can either be *weak*, which is implicit coordination, or *strong*, where the mechanism for coordination is explicit. A school of fish is weakly coordinated because the fish are not being assigned to swim next to a specific set of fishes, only to swim near other fishes.
- *Organization*. Organization is the algorithmic approach to control. One type of organization is to design MRS to be *strongly centralized*, where one computer controls all the members. The B1 battle droids in *Star Wars: The Phantom Menace* were strongly centralized and thus Anakin Skywalker was able to shut all of the robots down by destroying the Federation control ship in orbit about the planet Naboo. The opposite is *distributed* where there is no single computer controller, each member of the MRS works on its own or negotiates with others, and the overall MRS behavior emerges.

Distributed organization is common in animals and especially insects. The third type of organization is *weakly centralized*, which has some aspects of both the strongly centralized and distributed organization. An example is a centralized controller allocating tasks to groups and then allowing the members to work on their own.

17.6 Systems Dimensions in Design

SYSTEMS DIMENSION

The coordination dimension attempts to capture the strategic features associated with how robots work as MRS. The *systems dimension* isolates the physical features of the team that enable it to achieve the desired coordination. Farinelli, Iocchi, and Nardi identified four types of physical features: *communication*, *team composition*, *team size*, and *system architecture*.⁷⁵ This section will describe the first three types while the system architecture will be discussed in a separate section.

17.6.1 Communication

INDIRECT COMMUNICATION

DIRECT COMMUNICATION

The physical feature that may have the largest impact on the design of the MRS is communication. Communication refers both to the content of the message, that is, what is being said, and to the method of communication, that is, how it is being said. The content varies, so this subsection will discuss only the methods by which MRS communicate. Communication can be thought of as a spectrum ranging from indirect signals, or *indirect communication*, to explicit conversation-like *direct communication*. The method for communication relies on a way to transmit the message, which in robotics is generally via a wireless network which, in turn, leads to concerns about communication range and communication topology.

The spectrum of methods differs depending on which agents can perceive the message.

STIGMERGY

- *Stigmergic*. Stigmergy is a term used in biology to describe how insects like ants can communicate by what they have left in the environment. For example, ants leaving pheromone trails is stigmergic. In stigmergy, the agents do not perceive each other but rather they detect the results of each other's actions. For example, a hiker breaks twigs to leave signs of the route for other hikers to follow. Any species can encounter the cues that another species has left behind, but the cues and their meaning are typically interpreted only by other members of the same species.
- *Passive communication*. Passive communication means that one agent transmits or broadcasts the message to everyone in the hope of reaching other members of their species. For example, birds sing to communicate the presence of danger, the location of food, and to attract mates. Birds and other animals can communicate reproductive readiness through displays of plumage, skin color, or perineal swelling. These expressions are all active in that they require energy, but they are passive because the method is passive in reaching the intended recipient. A bird does not know if another bird will hear it sing. Stigmergy can be considered a special case of passive communication where

the broadcasted message persists rather than vanishes when the bird song ends.

- *Intentional explicit communication.* Intentional explicit communication occurs when one agent communicates directly with a specific agent using a formal language. A dog may growl at an intruder, which is, in some sense, intentional and explicit, but the intruder has to infer that the dog is about to attack; therefore, the communication is implicit. A human guard saying “Stop and put your hands up!” to another person is explicit communication of a specific intention.

In artificial intelligence, there is a heuristic that a little communication between agents can greatly increase performance, but too much communication can slow performance or make it brittle. Indeed, cognitive scientists have shown that the higher the performance of a team of people, the less the members rely on intentional explicit communication. Players on a top sports team can simply look at each other to know what play in is effect and if someone is ready for a pass. Yelling “Are you ready?” is much slower than looking and seeing that the other player is in position and poised to receive the ball. If there is too much chatter, players can become distracted and make mistakes.

Dudek⁶⁴ describes two practical concerns in communication between members in a multirobot system:

- *Communication range.* In general, the designer needs to consider how far apart the agents are. Can they see each other or hear each other so that they can use passive communication methods? If there is a wireless network, over what distances can it provide reliable support communication and what is the environment like? Wireless networks have transmission limits that depend, in part, on distance, and connectivity is unpredictable inside structures.
- *Communication topology.* Is the system using a mobile ad hoc network? Does the choice of communication mechanism limit its ability to broadcast or communicate directly with another member of the MRS? A poor choice of network topology utilizing a centralized communication server imposes the limitations of a centralized system onto what was intended to be a distributed system.

17.6.2 MRS Composition

HOMOGENEOUS MRS

HETEROGENEOUS MRS

MRS are composed either of identical members, which are *homogeneous MRS*, or of nonidentical members, which are *heterogeneous MRS*. Homogeneous teams are visibly identical; that is, they have the same *morphology* or physical appearance. Examples of homogeneous MRS are the two robots in the “Call a Conference” competition and groups of sheep-like demining robots. Examples of heterogeneous MRS are a UGV robot with a resupply assistant and UAV-UGV teams.

A common misconception is if a group of robots has identical hardware, then the group is a homogeneous MRS. But homogeneous means that the members have identical software *and* hardware. A large number of robots that are identical are often called a swarm, with the implication that all the robots have identical software as well. In RoboCup soccer, the robots may have the same morphology but the goalie may be running significantly different software. Thus, that particular MRS would be

heterogeneous. However, if each robot was programmed with identical software and the software contained all the roles, e.g., the goalie, striker, defender, and so forth, the MRS would be homogeneous. The team members have identical hardware and software, just different portions of the software are instantiated for individual robots.

A subtle challenge for a homogeneous MRS is that wear-and-tear of the members over time leads to heterogeneity. MRS can start out being homogeneous, but physical degradation or failures in hardware and sensors can result in some members being slower or unable to perform certain actions. Fortunately, the decay of an individual team member's capabilities is not a problem for some MRS and applications. For example, some members of a swarm may begin to work slower or totally fail, but one of the benefits of a swarm is that the other members expand their efforts to fill in for the failing member. In the case of a soccer team, other members might have to dynamically change their roles to compensate for a failing teammate. Note that changing roles requires more sophistication in execution monitoring and role allocation than is associated with insects.

Heterogeneous robot teams are generally thought of as robots with different hardware and software morphologies. A common heterogeneous team arrangement is to have one team member with more expensive, and capable, computer processing. That robot serves as the team leader and can direct the other, less intelligent, robots, or it can be used for special situations. The danger is that the specialist robot will fail or be destroyed, preventing the team mission from being accomplished.

MARSUPIAL ROBOT

A type of heterogeneous team that is becoming popular among bomb squads is a *marsupial robot* team,¹⁵¹ where a smaller “daughter” robot is carried by the “mother” robot, much like a kangaroo mother and her joey. When the mother robot needs a repeater node or helpful secondary viewpoint, the daughter is deployed. The mother robot can also act as a coach for the daughter. [Figure 17.2](#) shows an example. As early as 1997, researchers have explored using combinations of UAVs and UGVs. For example, the University of Southern California demonstrated a UGV searching an area based on feedback from a UAV. This combination permits a comprehensive view of a particular site or event.

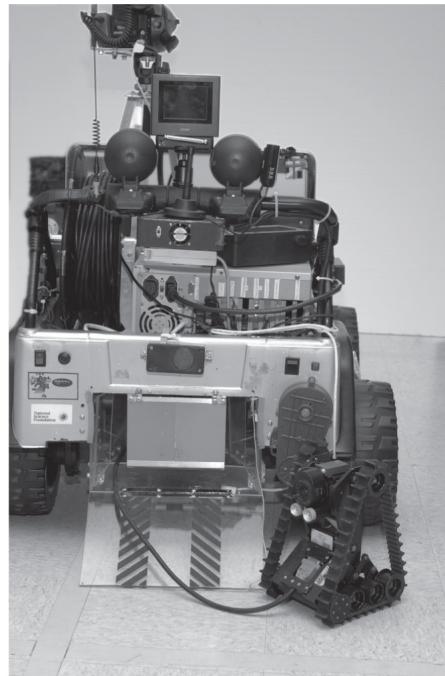
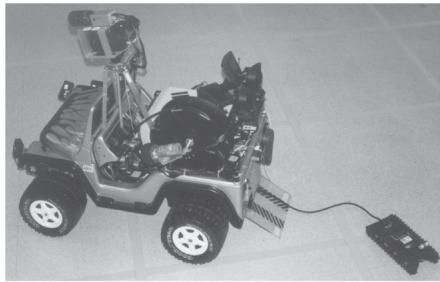


Figure 17.2 Two views of a marsupial robot team at University of South Florida. Silver Bullet is the “mother” carrying a tracked chemical inspection robot Bujold, the “daughter.” Bujold exits from the rear of the jeep. (Photographs by Tom Wagner.)

17.6.3 Team Size

The size of a team falls into four categories.

- *Alone*, the degenerative case of a multirobot system,
- *Pair*, the minimum for what most people consider a multirobot system,
- *Limited*, where the number n of robots is less than the size of the task or the environment, and
- *Infinite*, where n is greater than the size of the task or environment.

Limited and infinite team sizes are the more interesting categories. A limited team does not have enough members to perform the task as one group or in one simultaneous action. For example, a moving company does not provide a household with one person per box, instead they send a limited number of people who make multiple trips carrying the boxes from the house into the moving van. An infinite team has enough members, possibly too many members to effectively coordinate. For example, doubling the number of robots allowed on the field for soccer doesn’t guarantee doubling the score; the extra robots take up space, become obstacles, change the structure of plays, and increase the computation of planning algorithms. Infinite sized teams are often swarms.

17.7 Five Most Common Occurrences of MRS

COLLECTIVE SWARMS

INTENTIONALLY COORDINATED

Examining the ways of designing MRS, based on the types of tasks and the coordination and systems dimensions, can be daunting. Creating a complete taxonomy is even more challenging because there is overlap between the definitions of different tasks and dimensions. Therefore it is helpful to step back and identify the most common combinations of design attributes. There are common categories of combinations that show up most frequently in the scientific literature.⁷⁵ Three of these categories are variants of *collective swarms*: *unaware*, *aware but not coordinated*, and *weakly coordinated*. Two categories are variants of *intentionally coordinated MRS*: *strongly coordinated but weakly centralized* and *strongly coordinated but distributed systems*.

UNAWARE

Unaware means that each robot executes its own task without knowledge of the other team members. This is usually purely reactive and is often based on ant behaviors and stigmergy. Recall that ants do not notice the other ants most of the time and instead rely on sensing pheromones. Unaware systems are popular for domains that involve foraging or searching.

AWARE, NOT COORDINATED

Each robot in an *aware, not coordinated system* is somehow aware of other robots in the environment (e.g., a robot can tell another robot when it sees it), but it does not know what other robots are doing or why they are doing it. This may help reduce the problem of *task interference*.

A good example of MRS where the members were aware of each other, thereby preventing task interference, but were not coordinated was the multirobot team fielded by the Georgia Institute of Technology for the “Pick Up the Trash” event of the 1994 AAAI Mobile Robot Competition.¹⁸ In that event, each member moved randomly about searching for orange blobs (orange soda cans), then grasped the can, and then searched for blue recycling bins, moved to the bin, and dropped the can. The members were not coordinated. However, the MRS was aware of each other because the robots were painted green and each robot was repulsed by green. This meant the robots improved performance by spreading out in their search—that if random chance put one robot near another robot, both robots would move away from each other; therefore, no two robots would search the same area. It also meant that the robots did not interfere with each other by knocking cans out of the recycling bin and slowing down the recycling process, when two robots tried to deposit cans in the same recycling bin at the same time. Note that the robots did not have to be aware of the intent of the other robots or have any real semantic understanding that “green=robot”; they simply avoided “green.”

WEAKLY COORDINATED

In a *weakly coordinated* system, the task requires cooperation, but the system does not require explicit communication. An example of this is box pushing. The robots feel whether the box is getting pushed forward, and, if they feel pressure from contact with the box, they push against it. If there is no contact, they stop; the other robots will push the box on their end and thus pivot the box to the waiting robot and cause it to feel pressure. In this case, the communication is implicit, the robots are just feeling the contact, not explicitly communicating “I just moved the box. It is your turn now.” One advantage of a weak coordination scheme is that another robot, even a heterogeneous robot, can apply

the same behavior with the same results.

STRONGLY COORDINATED BUT WEAKLY CENTRALIZED

The fourth most common MRS team is *strongly coordinated but weakly centralized*. These teams require a leader, but the leader is selected dynamically. The advantage of selecting a leader dynamically is robustness. In strong centralization there would be one leader engineered for leading. If that robot is damaged or destroyed, the team can no longer function. In a weakly centralized team, another robot could take over as the leader. Thus, strongly coordinated but weakly centralized systems are popular for exploration tasks.

STRONGLY COORDINATED BUT DISTRIBUTED

COOPERATIVE MOBILITY

Strongly coordinated but distributed teams have plays and the ability to communicate, which is similar to strongly coordinated but weakly centralized, but the teams do not have a centralized leader. Each robot executes a particular script or role in their shared playbook, based on what they are immediately perceiving. This is a common configuration in RoboCup soccer. Another example of a strongly coordinated but distributed system is one where a robot might come over and help another robot in trouble. This is also called *cooperative mobility*.⁹⁰

17.8 Operational Architectures for MRS

Figure 17.3 places multirobot systems in the context of an operational architecture. Each robot has its reactive and deliberative functionality but also its interactive functionality. The additional functionality allows it to interact with other robots or with humans, the latter being the subject of the next chapter. Interactive functionality may be deliberative, reactive, or both.

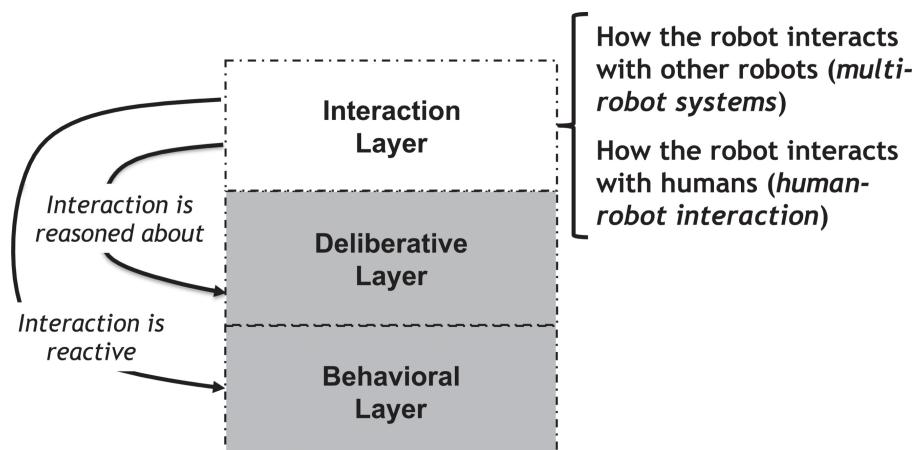


Figure 17.3 Operational architecture showing relationship between interactive and deliberative and reactive functionality.

One example of how MRS implement interactions is Mataric's *societal rules*, where group dynamics emerge in herds of multiple agents operating under fully distributed control.¹²³ Mataric compared three conditions: *ignorant coexistence* (that is, unaware) where there was no interactive functionality, *informed coexistence* where a reactive social rule governing interaction was added, and *intelligent coexistence* where a deliberative social rule was added.

These conditions illustrated a group of as many as 20 identical robots known as "The Nerd Herd," programmed with behaviors using the Subsumption architecture. The robots are shown in figure 17.4. The robots were given the same goal, however, the goal location was on the other side of a partition with a narrow door, permitting only one robot to pass through the partition at a time. The robots were placed randomly on the same side of the partition, and they started moving at the same time.

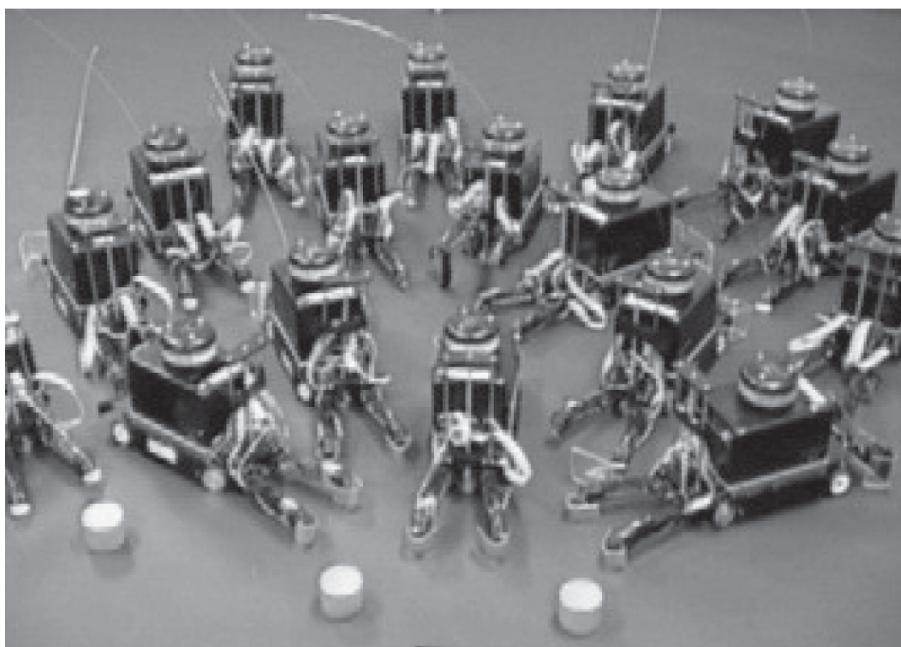


Figure 17.4 The Nerd Herd. (Photograph courtesy of USC Interaction Laboratory.)

IGNORANT COEXISTENCE

In the first set of demonstrations, the robots functioned with *ignorant coexistence*. The robots coexisted in a team, but they did not have any knowledge of each other. One robot treated another robot as an obstacle. Each robot had the equivalent of move-to-goal and avoid-obstacle behaviors. Since robots were treated as obstacles, once the robots gathered at the opening, they spent most of their time avoiding each other. The team as a whole made slow progress through the door to the goal location. Worse yet, the larger the number of robots fielded, the larger the traffic jam, and the longer it took to get all the team members through. This is an example of task interference in multirobot systems.

INFORMED COEXISTENCE

In the second demonstration, *informed coexistence*, the robots were allowed to recognize each other

and were given a simple social rule governing robot-robot interactions. In addition to move-to-goal and avoid-obstacle, a third behavior was created for avoiding robots. If a robot detected another robot, it would stop and wait for time p . If the blocking robot was still in the way after p , the robot would turn left and then resume moving to the goal. The result of the new behavior was to reduce the traffic jams, and the group got through the door in about the same time as a single agent going back and forth through the opening 20 times.

INTELLIGENT COEXISTENCE

The real surprise came in the third demonstration, *intelligent coexistence*. The social behavior for avoiding robots was replaced with another heuristic: each robot was repulsed by other robots, but as it moves away, it also tries to move in the same direction as a majority of other robots. A robot perceived any nearby robots as a repulsive field. But each robot could compute a second vector of the general direction of the other robot by summing the heading vectors of each robot. (The robots broadcast their heading over a radio transmitter to compensate for the inability to recognize each other by vision or sonar, so that is not considered communication.) A robot could see another robot, contributing a repulsive vector. However, the sensed robot could be heading towards the door, contributing a heading vector. The robot could sum the vectors which would generally move it away from a collision but still towards the door. As a result of intelligent coexistence, the robots exhibited flocking behavior and went through the door in single file! The heuristic created a need for each robot to follow the same heading as the majority. This, in turn, created a tendency to form a line, while the repulsion force from nearby robots caused the robots to create space for robots to merge into line. Together the two effects created a strong tendency to go through the door single file, even though there was no such explicit direction. Not only were traffic jams reduced, but the overall task was accomplished faster.

17.9 Task Allocation

Societal rules illustrated emergent coordination in a distributed team without explicit communication between robots but ignored teams and tasks where robots can communicate and need to divide responsibilities. This is referred to as the task allocation problem. There are three broad categories of approaches to this problem.

One approach calls for a centralized collective to reason and allocate tasks to team members. This approach has the disadvantage of being centralized, and AI roboticists tend to favor distributed solutions that do not introduce a vulnerable central controller.

Another approach uses an agent to broadcast information (“food is over here”) or requests (“I need help”), and then individual agents respond. For example, Gage⁷⁸ created MRS where each robot was endowed with a sense of shame. If a robot heard a request, the motivation to fill that request grew. If the robot was free, it would respond to the request. If it was busy, and the request was repeated (i.e., creating a growing sense of shame), the robot would respond as soon as its current task was completed. The shame could grow to the extent that the robot would interrupt its task in order to respond. The individual robots did not have to know about each other, directly communicate, or reason about optimal task allocation.

CONTRACT NET PROTOCOLS

Another approach, *contract net protocols*, allows the robots to communicate directly with each other in order to negotiate allocation. In contract net protocols, task allocation is like a market. Individual robots that are available and have suitable capabilities place bids on tasks. The quality of a bid is evaluated on factors such as how soon they can commence working on the task, estimates of how quickly they expect to complete the task, reliability, and so forth.

17.10 Summary

Many tasks favor the use of many cheap robots rather than a single expensive one. These collections of multiple robots are often referred to as multi-agents and multirobot systems. Designing a successful multirobot system involves understanding the task and the possible team configurations. Since the new emphasis is on the collective and the tasks, the designer now has to consider coordination as well as individual competence. This poses a basic design decision of whether to make the individual more intelligent, and by how much, or to make the group more intelligent. Designers also have to consider whether explicit plans are needed to accomplish a task, and, if so, how to allocate tasks or subtasks. Interaction also involves communication and determining when communication is needed and in what form, for example, stigmergy.

Returning to the questions posed in the introduction, the most fundamental question is: *What are multiple robots good for?* The chapter answers this question broadly by giving at least four motivations for MRS. In theory, MRS are good for tasks that are so complex that divide-and-conquer strategies are useful, or where many simple robots are cheaper, faster, or more robust than a single expensive robot. In practice, research has demonstrated the utility of MRS for coverage tasks such as foraging/search, multitarget observation, and exploration, as well as tasks requiring cooperation exemplified by demonstrations of box pushing, assembly, and playing soccer.

Are there different types of multiple robot teams—teams that act like insect swarms and teams that use more complicated intelligence? To date, there are five major groupings of coordination and system designs for accomplishing tasks. When the size of the group is large and the individual robots are not coordinated, they are often called swarms. Swarms generally connote a large number of identical, or homogeneous, robots replicating simple biological control principles, such as stigmergy, but some MRS are called a swarm despite using strong coordination and centralized control. The two competing connotations stem from a subtle difference between *how MRS are programmed and controlled* (i.e., the group of robots behaves as a biological swarm of bees) and *how MRS are used* (i.e., using many robots to accomplish a mission can be called a swarm). Any discussion of swarms eventually leads to the question: *Can you ever have too many robots?* The answer is yes, and too many robots result in the phenomenon called *task interference*.

A question of practical import for programming is: *How is programming multiple robots different from programming a single robot?* Programming a multirobot system is not very different from programming a single robot. MRS generally rely on adding additional behavioral rules or explicit communication mechanisms to an individually competent robot. An individual robot may do its share of the task reactively or because it is somehow explicitly assigned that task; the study of how programs make task assignments is called *task allocation*.

While significant progress has been made in multirobot systems and insects and science fiction

continue to inspire robot swarms, there are many unsolved problems from an artificial intelligence perspective. Monitoring a multirobot system is difficult. Determining when the collective or individual members are unproductive is hard if there is no centralized server. Swarms and other loosely coordinated teams have the potential to interfere with other team members. A major problem in adoption is that methods for creating provably correct designs of behaviors and for testing multirobot systems are virtually nonexistent. Multirobot systems accentuate the non-determinism found in all AI robotics, complicating testing.

The next chapter will introduce a new team, called a human-robot team, and will discuss the design of human-robot interaction.

17.11 Exercises

Exercise 17.1

Give at least three reasons why multi-agents are desirable. Describe the general attributes of applications which are well-suited for multi-agents, and give one example.

Exercise 17.2

List and describe seven challenges in designing multirobot systems.

Exercise 17.3

List and describe the key aspects of a multirobot system task that impact the choice of a multirobot system.

Exercise 17.4

Describe the two components below that should be considered when designing a multirobot system:

- a. the coordination dimension
- b. the system dimensions.

Exercise 17.5

Name the five most common multirobot teams in multirobot systems and give at least two examples of each.

Exercise 17.6

Give a one-sentence definition for each of the following:

- a. swarm robotics
- b. distributed problem solving
- c. reconfigurable robots
- d. cloud robotics
- e. stigmergy
- f. task interference
- g. contract net protocol
- h. networked robots.

Exercise 17.7

Define the following:

- a. heterogeneity
- b. control
- c. cooperation

d. goals.

Exercise 17.8

What is the difference between indirect communication and direct communication? Give an example of each from nature. Then give, or speculate about, an example from robotics.

Exercise 17.9

When would a collective swarm be desirable?

Exercise 17.10

Find an example from the Web of two multirobot systems—one that forms a collective swarm and one that is intentionally cooperative.

Exercise 17.11

Consider a team of robotic space ants that are programmed to go to the nearest stationary asteroid and bring it back to base. What would happen if the first robot communicated with the other robots to recruit them to help move the asteroid. Would the behaviors or the goal structure necessarily change? Why or why not?

Exercise 17.12

Describe and compare three approaches to societal behavior: social rules, internal motivation, and leadership. Which do you think is easiest to program?

Exercise 17.13

Explain the impact of *ignorant coexistence*, *informed coexistence*, and *intelligent coexistence* on getting a team of robots through a narrow opening.

Exercise 17.14

Were the behaviors for the Nerd Herd purely reactive? Why or why not?

Exercise 17.15

Consider a team of robots inspecting water irrigation canals for leaks. One team member is a UAV that flies each month. The data from the UAV are then processed at the water district office in order to identify leaks. If a leak is detected, an unmanned surface vehicle is then dropped off to scan that section of the canal because it can see both above and below the waterline. The USV also carries a ROV to get more direct views of any underwater damage to the canal. The USV controls the ROV. Each of the robots has behaviors and deliberative functions. Describe the system in terms of each component of the task, coordination, and system dimensions.

- a. task
- b. coordination
- c. system.

[*Programming*]

Exercise 17.16

Implement the space-ant example with 3–5 robots capable of phototaxis and dead reckoning.

- a. Multi-agent foraging. For this problem use only wander, phototropic and avoid-robot behaviors, where a robot is any obstacle that is not a light. The program will start with an empty world consisting of a light (you may need to make a “bigger light” by placing lights next to each other). Place the robots at different random starting locations in the world. Each robot will wander through the world, avoiding obstacles, until it comes to a light. Then it will move directly to the light. If more than one robot is attracted to the same light, the avoid obstacle behavior should cause the robots to center themselves evenly spaced around the light. Compare this program to the program in chapter 19, which has a single robot forage for lights. Which program gets more lights faster?
- b. Cooperating to bring the food home. Now add the push-to-home behavior where the robot wants to be on a straight

line from the light to home. What happens now?

[*World Wide Web*]

Exercise 17.17

Visit the RoboCup website at www.robocup.org. Which team has performed the best over the past three years? Describe the multirobot system organization in terms of control and cooperation.

[*Advanced Reading*]

Exercise 17.18

Read Ed Durfee's humorous paper on distributed artificial intelligence (DAI), "What Your Computer Really Needs to Know, You Learned in Kindergarten" (proceedings of the *Tenth National Conference on Artificial Intelligence*, 1992). For each of his ten issues ("Share Everything," "Play Fair," etc.), describe how this applies to robots. For each issue, give an example of how they apply to a robot team described in this chapter.

[*Advanced Reading*]

Exercise 17.19

Read and summarize "Behavior-Based Formation Control for Multirobot Teams" by Tucker Balch and Ron Arkin in *IEEE Transactions on Robotics and Automation*, vol. 14, no 6., 1998.

[*Science Fiction*]

Exercise 17.20

Watch *Silent Running*, the movie about a team of three mobile robots (Huey, Dewey, and Louie) working on a space station. Classify their teamwork in terms of heterogeneity, control, cooperation, and goals.

[*Science Fiction*]

Exercise 17.21

Watch *Star Wars IV-VI* and observe C-3PO and R2-D2. Classify their teamwork in terms of heterogeneity, control, cooperation and goals.

17.12 End Notes

For the Roboticist's bookshelf.

Robot Teams: From Diversity to Polymorphism edited by Tucker Balch and Lynne Parker is a classic introduction to the varieties of multirobot interactions.

Swarms and flocks.

The references to swarm robots are too numerous to cite here; many papers explore details of insect behavior and coordination strategies as well as provide simulations. Any proceedings of the Annual Conference on the Simulation of Adaptive Behavior (also called "From Animals to Animats") is an excellent starting point for swarms. Jean-Louis Deneubourg has produced many interesting articles synthesizing insights from insect colonies into a form useful for programming mobile robots. As noted in *Behavior-Based Robotics*, Craig Reynolds' work in computer graphic simulation of flocks in "Flocks, herds, and schools: A distributed behavior model," in *Computer Graphics*, 1987, showed how flocks emerge from simple, individual interactions.

Kangaroos and possums and robots, oh my!

The term "marsupial" first appeared in the robotics literature in 1996 after John Blitch coined the term during graduate work at the Colorado School of Mines. Following that, a marsupial robot configuration built by undergraduates was demonstrated at the 1997 AAAI Mobile Robot Exhibition. The term gained wide acceptance in the US in 1998 when it became an objective of a DARPA technology development program. The North American continent does have one native marsupial species, the opossum or "possum." The possum is native to the southern part of the US.

More robot name trivia.

The Nerd Herd consisted of IS Robotics R2 robots which look like toasters—brightly colored toasters, but toasters nonetheless. The 20 robots were named for things that come out of toasters, for example, Bagel.

Sci-fi movies.

The 1972 film *Silent Running* is worth watching despite being a laughably bad movie. It was directed by Douglas Trumbull just after he had completed his work on *2001: A Space Odyssey* and was written in part by Steven Bochco, who would go on to make TV history with *Hill Street Blues*. Legendary folk singer Joan Baez performed the cover song for the soundtrack and Bruce Dern was the lead actor. It should have been a blockbuster. Instead, it was a slow slog of environmental preaching and generally uninteresting characters except for the three robots, Huey, Dewey, and Louie. Their activities and their interactions with each other and with Dern's character are fairly plausible. The robots themselves seemed naturalistic, lowering themselves to the ground when being given long winded directions, and so on. The short robots were actually amputees walking on their hands.

18

Human-Robot Interaction

Chapter Objectives:

- Define *human-robot interaction (HRI)* and detail at least one way in which the study of HRI uses insights from each related area: human-computer interfaces (HCI), psychology, and communications.
- List and distinguish between the three types of user interfaces (*diagnostic, operational, and explicative*) a roboticist may have to build for a successful intelligent robot system.
- Given an application, identify whether it involves *physical, cognitive, or social/emotional* interactions.
- Define the three levels of *situation awareness*. Given a description of a robot application, identify where those levels are manifested.
- Define *trust* and list five ways that robots can increase trust.
- Write the formula for the *safe human-robot ratio*. Given a description of a robot application, use the formula to justify the appropriate manpower required.
- Describe the four methods for collecting human-robot interaction data: *interviews, surveys, and journaling; observation; biometrics; and specialized tests*.
- Describe the five categories of human-robot interaction system metrics: *productivity, efficiency, reliability, safety, and coactivity*.
- Explain the difference between the *speech recognition, language understanding, and language generation and speech synthesis* phases of natural language understanding and describe the current level of capability of each.
 - Describe the three approaches to language understanding, *words, syntax, and semantics*, and how they might be used by robots.
- Define *team, sensemaking, Uncanny Valley, common ground, beliefs, desires, intentions (BDI), Wizard of Oz (WOZ), deictic gestures, and signaling gestures*.

18.1 Overview

Although designers like to think of intelligent robots as stand-alone agents, there is always a human in the system somewhere, either “behind” the robot as an operator or stakeholder in what the robot is doing, “in front” of the robot being helped by the robot or having purposeful interactions, or “beside” the robot in the workplace where the robot happens to be located. Currently, robotics concentrates on the interaction with people specifically trained and responsible for the correct operation of the system—the pilot or operator “behind” the platform. This concentration on pilots and operators ignores other “behind” stakeholders such as the maintenance people, trainers, and those who test and evaluate the system. More attention is being paid to how robots interact with humans that are not trained or responsible for the robot. These humans are “in front” of the robot, for example, civilians in a town that an autonomous car is driving through. In theory the agent should be able to convey what the user is supposed to do and to inspire confidence. Users or bystanders rely on past experiences, expectations, and so forth, to help determine what to do and how to react; if they have no experience with the system, they often overestimate intelligence, capability, awareness, and sturdiness of the agent. In unmanned systems, civilians often come up to robots, touch and handle them inappropriately, thereby causing damage, or expect the robots to work safely around them. Some researchers are beginning to consider how unmanned systems act as a near-peer or member of an ensemble, for example, a robotic mule which should not roll over soldiers who are lying on the ground napping.

Human-robot interaction does not have a precise definition. Following the foundational workshop that established the field of HRI,³⁵ the goal is to enable “...synergistic teams of humans and robots where team members perform tasks according to their abilities.”

UNCANNY VALLEY

Using that as a working definition, it is clear that the field of human-robot interaction is devoted to the rich interactions between robots and people, and this chapter describes how artificial intelligence is a key component of good human-robot interaction. It covers how humans and robots can work together in a team. It also looks at situations where robots come across as creepy. The term *Uncanny Valley* was coined by Mori¹³⁶ to explain that when the physical realism of a humanoid robot is very high but not high enough to be perfect, the uncanny resemblance to a human goes from interesting to creepy.

TEAM

A set of rich interactions may be incorrectly referred to as human-robot teams, as “team” is often used to connote any ensemble of people working on a joint activity. However “team” means something different in organizational and cognitive psychology. As noted by Klein et al.,¹⁰⁵ in cognitive science, *team* connotes people who have worked together, have work practices, and will (and know how to) help each other out. It may take months or years for a human team to form; consider the difference between a pickup team for a game of basketball versus a professional basketball team. Especially consider the difference that practice, investment in consequences, and so forth, make in the team’s output. Groups of people working together that have not bonded as a true team are referred to as *ad hoc teams*.

Human-robot interaction design often is motivated by designers and users asking the following questions. Users ask: *How many people does it take to run a robot?* so they can plan for the

organizational and economic impact. Designers are often under unrealistic expectations to reduce the human:robot ratio. Designers also ask: *How to divide up responsibilities or roles?* and *How do people like to interact with robots?* Understanding the answers to these questions is essential for successful entertainment and assistive robots, but the answers also influence even routine, task-oriented interactions. *Do robots and people need to understand each other (e.g., do they need to have a shared cognitive model)?* Users often ask: *Why can't I just tell the robot what to do because then I don't have to worry about interfaces and controls?* because they are unaware of the hidden intelligence required for robots to understand natural language.

This chapter will attempt to answer all of these questions. The chapter begins by presenting a taxonomy of interactions, where a human-robot system has a mode of cooperation and a style of cognitive engagement that influences the design. With an established taxonomy, it becomes easier to see the contributions to HRI from the fields of human-computer interfaces, psychology, and communications and why HRI draws heavily from these three fields, yet remains unique. The first step in designing successful human-robot interactions is to model the work domains, the users, and their expected interactions with the robot(s). *Cognitive task analysis* and *cognitive work analysis* are two common approaches to modeling. Once the work domain is understood, it is possible to determine the human:robot ratio required for safely operating the robot. After establishing the roles of the humans and the robots, the user interfaces to support each class of user become more straightforward. User interface design is guided by principles succinctly summarized as the eight golden rules.¹⁹³ It is presumed that a good user interface supports *situation awareness*, and the chapter explores formal definitions of situation awareness and the related concept of sensemaking. Given the assumption by users that a good *natural language interface* would eliminate most, if not all, of their HRI problems, an entire section is devoted to natural language and why it is still a challenge for AI. The section also discusses *multi-modal communication* where agents use gestures and poses to clarify and amplify what they are trying to communicate. *Trust* is considered an outcome of good human-robot interaction design, so the chapter devotes a section to it and the factors that influence a user's trust in a system. Testing human-robot interaction is different from testing a computer program or hardware design and involves a variety of techniques for gleaning valid information from users. The chapter concludes with observations about how artificial intelligence can aid good human-robot interaction.

18.2 Taxonomy of Interaction

Because intelligent robots are being used for more applications, new ways in which robots and people interact continue to emerge. Early approaches to HRI attempted to categorize the rich variety of interactions using relationships such as supervisor, peer, subordinate, and so forth.¹⁸⁸ These taxonomies have not been successful at framing the underlying computational challenges in designing workable HRI, in part because a person can hold several relationships with another person, such as being a supervisor and also a friend, and that leads to blended responses.

An alternative to using social relationships as descriptors is to consider the modes that robots and humans use to cooperate and the style of *joint cognitive system*. This taxonomy concentrates on capturing the impact of cooperation on computational intelligence.

The three primary *HRI modes of cooperation* are:

- *Physical*, where the robot and the person are in direct physical contact, such as with surgical robotics, or indirect physical contact, such as sharing lifting a load. In physical HRI, the focus is on ensuring the safety of the person(s).
- *Cognitive*, where the robot and person are involved in joint work, such as defusing a bomb.
- *Social/emotional*, where the robot is engineered to influence how the person responds to the robot either explicitly, such as for entertainment, or implicitly, such as a training aid for autism.

JOINT COGNITIVE SYSTEM

The cognitive mode of cooperation has *styles of cognitive engagement*. When robots and people interact there is some jointness, that is, there are some mechanisms by which the two types of agents engage with each other. These mechanisms have a cognitive component, where the person and robot are thinking and reacting to the same world and task, either implicitly or explicitly. That joint cognition may be shared, when the two agents are explicitly working closely together on a task. How machines and people interact to accomplish their goals was termed by Woods and Hollnagel²²⁰ a *joint cognitive system*. The term “system” is used because the machines and the people are not necessarily a team. These cognitive engagements can be loosely divided into:

TASKABLE AGENT

- *taskable agent*, where the robot is treated as an independent agent that is delegated some degree of autonomy and initiative. This introduces the cognitive challenges of effectively delegating a task and trusting that the agent will carry out the task or will alert the human that there are problems, in a timely fashion.

REMOTE PRESENCE

- *remote presence*, or first-person view, where the robot is a real-time extension of a human in an environment or situation that a human cannot be in. This introduces the cognitive challenges of mediation, as discussed in chapter 5, and the novel tasks where the human-robot system is attempting something new and that increases personal stress.

ASSISTIVE AGENT

- *assistive agent*, where the robot is co-located with a human in order to help the person. This introduces the cognitive challenges of creating and maintaining mental models of what the other’s intent and expectations are and of social congruence.

Table 18.1 gives a matrix of examples of human-robot interaction as an intersection of style of cooperation and mechanism of engagement.

Table 18.1 Examples of human-robot interaction for combinations of cooperation and engagement.

	taskable agent	remote presence	assistive agent
social/ emotional	museum robots	telecommuting	Paro
cognitive	driverless cars	drones	weight loss coach
physical	kiva warehouse	DaVinci surgical	smart wheelchair

18.3 Contributions from HCI, Psychology, Communications

As noted in the 2001 workshop that bootstrapped the field,¹⁷⁹ human-robot interaction is the intersection of three major disciplines: human-computer interfaces, psychology, and communications.

18.3.1 Human-Computer Interaction

Human-robot interaction is related to the fields of human-computer interfaces (HCI) and computer-supported work groups (CSWG); these two fields are sometimes called CHI for computer-human interaction. HCI concentrates on the way people interact with computers. The field has many subareas including ergonomics, human factors, usability, and multi-modal interfaces. In general, designers of computer interfaces have a model of what the user needs to do and the user's preferences and expectations about how to do it. HCI is different from HRI because computers do not move but robots do.

HCI offers many principles for designing interfaces. Some particularly relevant ones for robots are:

- Avoid visual overload, especially in displaying multiple windows and camera views.
- Use formal test methods because the designer (or his/her officemate) is not the end user and the designer's ability to use the interface, or like it, is not a reliable predictor of whether the end user can use it or like it.
- Favor simple displays that use a few colors sparingly, and follow conventions such as: Red means “stop” or “bad.”
- Be aware of “visual capture,” where a user tends to watch movement in videos and ignore other displayed information.
- Apply the Ten Minute Rule: If a user cannot use the robot within ten minutes, then they will discard the robot.
- Minimize the number of clicks or pull-down menus that it takes to accomplish important or frequently occurring functions.
- Remember that users get familiar with system over time, that is, they transition from novices to experts, and an interface designed only to help novices may hinder users when they become more expert.

18.3.2 Psychology

ERGONOMICS

Human-robot interaction is related to psychology, particularly to the fields of cognitive engineering and industrial organization psychology. Psychology addresses team processes, especially how people work in *groups*, function in *ad hoc teams* that are formed suddenly, such as a pick-up game of basketball, and function in *high-performing teams* that have significant experience working together, such as professional sports teams. Psychology also addresses *social informatics*, especially who in a group or team has what role, when does a team member hold a role, how do team members interact and change roles/responsibilities, and how do the members fit into an organization? A major topic is *resilience* or how people working together can help each other out in order to meet the larger common goals. Psychology often addresses *ergonomics*, which focuses on how machines can best work within people's physical and neurophysiological abilities. These models of team processes, social informatics, and resilience can be used as the basis for human-robot team processes as presented by Clifford Nass.¹⁷⁸

Psychology also offers many insights into thinking about HRI. Some important ones for robotics are:

- Introspection is misleading; the way you think is often not really how you think. Therefore saying "I think about a problem this way, and I will duplicate this line of reasoning" has little value in increasing the intelligence of a robot.
- People can manage only a limited number of "chunks" of information, which, according to Miller's Law, is 7 ± 2 or 5 to 9 chunks; training people to remember lengthy sequences or to manage large number of inputs or actions may work in optimal conditions, but people are prone to forget and make mistakes.
- People vary according to skill levels, culture, socio-economics, age, experience, personality, and reactions to stress. Testing HRI without a truly representative sample of end users is misleading and not reliable. As noted earlier, an office mate or colleague rarely reflects the demographics, training, skill set, and interests of the end user.

18.3.3 Communications

Communications is the field that studies media. This is particularly relevant to HRI because information and directives to and through a robot or computer are mediated by the robot or the computer. Communications also intersects with HCI in studying how people communicate with each other and with machines, especially verbally and through signals, but it is different from HCI in that communications focuses more on what is being said and why. Communications theory is a major driver for research in social robotics, for example, the bedside manner of a robot.

Communications offers many insights into thinking about HRI, the most significant of which are:

MEDIA EQUATION

- The *Media Equation*, which states that *media = real life* or that people treat media as if it were human. Beginning with Reeves and Nass' work with interactive computers and the Computers Are

Social Actors paradigm,¹⁷⁵ it has been documented that people subconsciously treat anything that moves as if it were alive. People expect the same consistency of action, the same respect for personal space (called *proxemics*), and the same cues of intention that they would get from a dog, other animals, or another person.

- People do not trust and accept robots if they do not behave with the mannerisms consistent with animals or other people. If the robot looks and acts smarter than it really is, this creates cognitive dissonance and distrust.
- People’s reaction to working with or around robots often evolves over time. Testing whether people like a robot with only an initial interaction, similar to a food “taste test,” is likely to produce questionable results. A robot that is tolerable at first may be annoying over time.

18.4 User Interfaces

User interfaces enable people working directly with the robot to interact with it. The type and content of the interface depends on the goal of the user. There are three kinds of user interfaces that a roboticist will have to incorporate into an intelligent robot system.

DIAGNOSTIC USER INTERFACE

Diagnostic user interfaces are intended to let the robot designer access the detailed operation of the robot and the individual algorithms. These types of interfaces often have too much detail about the robot and too little about the context for actually running the robot; recall the Darkspot incident in chapter 5.

OPERATIONAL USER INTERFACE

Instead, the end-users need an *operational user interface* that permits successful and safe operation of the robot including exceptional cases. The end-user may be a robot operator, such as a bomb squad technician, or a mission specialist, such as an emergency manager looking at the information coming in from a UAV.

The user interface for the robot operator “behind” the robot is often called the *operator control unit (OCU)*, or operator control station, when there is only an operator interacting with the robot. The look and feel of the interface is often the focal point of user interface design. The focus on the look and feel may distract the designer from considering the more subtle aspect of how the user interface influences the creation and maintenance of situation awareness. Because situation awareness depends on the role of the person using the robot, each user may require a unique user interface.

EXPLICATIVE USER INTERFACE

A third variation of the user interface, an *explicative user interface*, may be needed to enable journalists, sponsors, or VIPs to use the robot and to appreciate its inner workings.

18.4.1 Eight Golden Rules for User Interface Design

Schneiderman and Plaisant in¹⁹³ collected Eight Golden Rules for user interface design:

1. *Strive for consistency.* Consistency means that the same actions lead to the same results in similar

situations or that colors have the same meaning in all cases (e.g., red is “bad,” “stop,” or “alert”). One example of consistency is the use of Futaba controller or videogame controller conventions for joysticks. While there is evidence that the Futaba radio controller convention introduces problems for controlling robots, it is commonplace and thus merits using rather than trying to create a new or better controller. Similarly, the Dvorak and Coleman keyboards are more efficient, but QWERTY remains the standard because more people are familiar with it.

2. *Cater to universal usability.* *Universal usability* enables people with different backgrounds, ages, and experiences to use the robot, including people with disabilities. But universal usability is broader than that. An example of the need for universal usability is in transitional user interfaces. A user interface that works well for a beginner, such as a game controller interface, may not work well for users as they become more expert or want to perform more expert tasks. User interfaces may permit experts to use shortcuts, power keys, remap keys, or create macros.
3. *Offer informative feedback.* Feedback helps the user keep up with the state of the robot, especially if commands have been executed. For example, if the robot changes mode, there should be some indication, such as the appearance of icon or a change in the status bar to signal that the mode has changed or the robot may emit an audible tone, or both audible and visual signals may be produced. It is important to avoid overusing feedback and thus annoying the user; the rule is that the feedback should be proportional to the significance and rarity of the action.
4. *Design dialogs to yield closure.* The ideas of dialogs and closure are often manifested in sequences. Most robot missions have a initiation phase, followed by execution and termination phases. The types of activities and options generally change for each phase. For example, the icons for functions may appear and disappear based on the phase. The interface for a UAV ready to launch might display the auto-takeoff icon but not the return-home or auto-land icons. Those icons would become visible when the UAV was out of the takeoff phase.
5. *Prevent errors.* It seems obvious that the user interface should prevent errors. While monitoring the mission progress and uncovering problems are deliberative functions, the interface can help even without artificial intelligence. It can prevent errors by requiring a confirmation message or an additional button press for high consequence actions such as shutting off the motors, or it can provide helpful reminders or easy to access help functions.
6. *Permit easy reversal of actions.* Reversal of actions can be a hard concept for robotics because physical actions are often irreversible or not easily or automatically reversed the way the “undo” function works in a word processor. Examples of reversible actions are the ability to modify a planned path or bounding box for the operation of the robot and a return to home function, which allows the user to recall a remote robot. If the system cannot reverse an action, then the options to stop the action or exit from it must be clear.
7. *Support internal locus of control.* Users want the interface to work for them, to make their jobs easier, not the other way around. For example, a poor user interface requires manual entry of GPS waypoints rather than allowing the user to select them by clicking on a map. The principle of internal locus of control is often interpreted to mean that users want to maintain direct control of the robot, such as driving the robot and taking their own photos or videos. But Peschel et al.¹⁶⁸ found

that responders, especially those higher in the decision hierarchy, preferred to delegate control as long as they could comprehend what the robot was doing.

3. *Reduce short-term memory load.* For example, recall the robot from chapter 5 that had a large sequence of steps in configuring the robot to go down a flight of stairs. The user interface provided no reminders of the steps or even any indication that each step had been completed by the robot, so this meant the operator had to remember the sequence and the state of the robot. It was no surprise when the robot crashed despite having an expert operator. Also, user interfaces can increase or decrease short-term memory load with multiple screens or windows because anytime users have to look at different screens or monitors, they have to remember information, and that increases the potential for error.

User interfaces are just one aspect of an OCU, and it is important also to consider *ergonomics* and to balance feedback with understanding the user. For example, the feedback from troops using small tactical mobile robots for handling improvised explosive devices in Afghanistan and Iraq was that all of the robots had annoying interfaces. Each robot had a different interface with joysticks, windows, and buttons. The troops often said they wanted the PlayStation2 controller. Defense contractors immediately went to work adapting their software to work with the controller. The soldiers were generally pleased, except they began to complain that the horns of the controller got caught in their pockets, the buttons were hard to press with gloves, and the mapping of functions unto buttons did not always make sense. In retrospect, the contractors had missed that the users were trying to say: “We want all the robots to use the same joystick interface and the joystick interfaces should be designed with the same attention to detail that videogame manufacturers put into the PS2 controller so that the robot controller is as easy and comfortable to use for robot tasks as the PS2 is for videogame tasks.” The contractors had missed the ergonomic ramifications of the PS2 controller. They had also missed the transitional user effect, where the PS2 controller was easy to use at first by a novice soldier but became harder to use for more sophisticated functions as a soldier went from being a novice to an expert.

18.4.2 Situation Awareness

SITUATION AWARENESS

How a person keeps up with what the robot is doing is an instance of the phenomenon of situation awareness, and a good user interface ideally assists the person in building and maintaining *situation awareness*. Situation awareness, abbreviated SA, was defined by Mica Endsley as “...the perception of the elements in the environment within a volume of time and space, the comprehension of their meaning and the projection of their status in the near future.”⁷⁰ A good user interface will support all three levels of SA.

Following Endsley, situation awareness can be divided into three levels of increasing awareness:

- *Level 1 SA* is sorting through the sensory input to perceive the relevant sensory information.
- *Level 2 SA* is interpreting and comprehending the sensory information and what it means for the current or near term mission objectives.

- *Level 3 SA* is synthesizing and projecting what the sensory information means for future events.

Ideally a user has Level 3A situation awareness. This means that the user is able to perceive what is relevant and can build and maintain an accurate understanding of the current state of the world and mission so that they can execute the mission. And while they are executing the mission, they are also thinking about the next step and are on the alert for any signs of problems. However, the user has only a finite amount of cognitive effort that they can spend, and that effort decreases over time as the user gets tired and experiences increased stress. If the user devotes the majority of the effort to Level 1 SA, then there will be little effort available for Level 2 and Level 3.

User interfaces are the most obvious mechanisms for assisting the user in perceiving the relevant sensory information and relating it to current objectives. A cluttered user interface that attempts to displays everything often interferes with a person's ability to build and maintain Level 1 SA because the user has to expend considerable cognitive effort, that is, the task has increased the *cognitive workload* by requiring more effort to sort out and find the relevant information. However, a user interface can err and provide too little information, especially if the robot does not provide adequate sensory information.

Consider the example user interface in [figure 18.1](#) for the Inuktun Mine Crawler used to attempt to locate trapped miners at the Crandall Canyon, Utah, mine disaster.¹³⁷ The robot was polymorphic: it lay flat in order to pass through a borehole, and then the camera mast rotated up. Recall that users often need to know the exproprioceptive state of the robot, in this case, whether the robot was about to flip over as it crawled over a rock or debris. Many user interfaces provide an icon of the proprioceptive state of the robot, in this case, the position of the mast. The user then mentally combines any sensory data about the terrain, such as a range map or visual image, with the proprioceptive information to estimate the exproprioceptive state. However, many robots, such as the Inuktun, provide no proprioceptive information about the pose of the robot. In this case, the robot did not have encoders to indicate the angle of the camera mast—the operator was expected to remember the approximate angle of the mast and not fully extend the mast or it would jam in the full upright position and the robot would not be able to lay flat and re-enter the borehole to return to the surface.



a.



b.

Figure 18.1 The Mine Crawler robot and its user interface showing the robot's-eye view of the environment with time, date, and distance information overlaid.

This impoverished user interface impacted the operators over time. Over the hours that the robot was on the mine floor searching, the operators, who had had little sleep and were working in stressful conditions, could not maintain the estimate of the position of the camera and had to retract the camera constantly in order to start over from a safe state. As a result, the operators were most likely working only at Level 1 SA. The operators had to stop the robot periodically so that they could concentrate on perceiving indications of a trapped miner or a clue to the cause of the collapse. The operators rarely, if ever, reached Level 3 SA, where they had time and mental resources to think about the next step or anticipate and prepare for any upcoming problems.

Situation awareness and *situational awareness* are used interchangeably, which is unfortunate because situational awareness refers to a separate topic of how a person's attention to aspect of the world is dependent on the situation. Situation awareness, on the other hand, is the state of a person's comprehension of the world. The term *sensemaking* may be used instead of situation awareness to avoid confusing with the vernacular use of the term and to emphasize the perceptual functions of the person as a whole. Klein defines sensemaking as "a motivated, continuous effort to understand connections (which can be among people, places, and events) in order to anticipate their trajectories and act effectively"¹⁰⁴ which is broader than the state of knowledge underlying situation awareness.

18.4.3 Multiple Users

Often there is more than one user of a robot, and thus there may need to be multiple displays and even multiple interfaces to support the different situation awareness needs for each user. For example, with a UAV performing reconnaissance for police at an event, there would be the operator. However, most likely, there would be other officers who know nothing about the robot but would need to see the camera feed and even retask the robot to get a different view. One solution is to provide a display that mirrors the operator's display, which is an option with the DJI Inspire UAVs, or to stream the operator's display to users over the Internet, which is an option with the DJI Phantoms. However, other users are experts in their particular job. That means they are interested in the information relevant to their responsibilities rather than information about the operation of the UAV. Therefore, the operator and the expert may have different goals, which means that each has a different "situation" that they need to be aware of. Therefore they need somewhat different user interfaces. Icons or labels about the UAV's battery levels, strength of GPS and wireless connections, and so forth, can be distracting and are usually not of interest to anyone other than the operator. Furthermore, the operator-oriented information may be overwritten on the imagery, thereby blocking the view of an object of interest. As a result, it may be beneficial to develop a different version of the interface for the "information consumer" users. Such an interface can empower those users and increase their acceptance of the robot.¹⁰⁸

Multiple categories of users lead to multiple role-specific displays but also raise issues of lack of collaboration and contention. With a group of users and only one robot, how do the users communicate with the operator or robot? If the users are given control, how does the system ensure that the users do not issue conflicting directives? Preventing these conflicts involves modeling the domain, the users, and their interactions, as well as how the operator, robot, and other users maintain common ground; these concepts are discussed in the remainder of this chapter.

18.5 Modeling Domains, Users, and Interactions

The possibility of multiple users of a robot, or set of robots, with different goals highlights more subtle challenges in determining the appropriate human-robot interaction: how do designers determine or model what the users' goals are, what do users need to be aware of in order to work with the robot and accomplish the mission, when do users need to be aware of what aspects, what information does the environment or robot provide, and how do users and robots interact with each other? The answers to

these types of questions are found in a *domain model*.

In artificial intelligence, domain modeling is generally approached either by a) creating ontologies that capture the core attributes of the domain and the relationships between attributes or b) explicitly representing expert knowledge. The representation of expert knowledge may make use of an ontology. There is a whole field within AI called *knowledge engineering* that is devoted to methods of extracting, representing, and reasoning about expert knowledge and expertise. Knowledge engineering approaches are excellent for determining the sources of information and representing how it is transformed by an expert into a decision, but often the approaches are narrowly focused on the knowledge to be encapsulated, not on how people interact with such knowledge or the agents that host such knowledge. There are two techniques for anticipating how people and robots will interact for a specific application domain: *cognitive task analysis* and *cognitive work analysis*.

18.5.1 Motivating Example of Users and Interactions

In 2013, Texas was one of several states that began introducing teleoperated robots as a means of enabling homebound students or students with extended hospital stays to keep up with classes. Texas uses the VGo robot (VGo Communications), a tablet on an eye-height pole mounted to a wheeled mobile platform. The system provides the teleoperator with two-way audio and two-way video. The robot's base has two degrees of freedom, translation and rotation. It is teleoperated over an Internet connection. This use of robots illustrates why good human-robot interaction is more than just a user interface and why it is beneficial to understand the work domain, identify the complete set of users and stakeholders, and formalize how all users and stakeholders expect to interact with the robot.

It is essential in HRI design to identify all stakeholders who are impacted by the design. The stakeholders that had to be satisfied by the system design included more than the homebound student and teacher. The homebound student's parents or guardians had to be supportive and engaged in order to encourage the robot's use. The other students and their families were stakeholders because the robot could not be a distraction or disruption in the classroom and thus interfere with their education. The program managers for the education service center were stakeholders, because they were responsible for purchasing and maintaining the robots, training the students, parents, and teachers, and making sure the robots were being used. The state of Texas as a whole was a stakeholder as the robot had to be able to clearly help the homebound student without burdening the teacher or distracting the other students, and the robot had to be a cost-effective and scalable solution.

In HRI design, the role of each stakeholder, especially how their interests and expectations shape the interaction, must be determined. Considering who is "behind," "in front," and "beside" the robot is helpful. The stakeholders that had direct interaction with the robots, or users, included more than the homebound student. "Behind" the robot was the homebound student, plus their parent or guardian as well as the education service area program managers. "Beside" the robot were the other students in the classroom who were literally beside the robot as it occupied the physical place of a student. These students could be instructed about the robot and what to expect from it. Some of the students might be "in front" of the robot, actively working with the homebound student on group exercises and team projects. The teacher was "in front" of the robot and could be trained to move it, reboot it, and recharge it. However, there were also the students, teachers, and school workers "beside" the robot who might encounter it in the halls and throughout the school; this general population that would not be

specifically trained on how to interact with the robot.

Establishing the identity of the stakeholders and determining their interactions enables the appropriate user interfaces to be designed. “Behind” the robot, the student needed to be able to navigate and control the camera. The parents needed to be able to set up the robot, troubleshoot any problems that the student could not solve, and observe what their child was doing. The parents and the student most likely had different computer skills and comfort levels with computers and robots. The program manager needed to see the status of all the robots in the education service area and remotely login to help troubleshoot. But there was a user “in front” of the robot, nominally the student situated in the classroom. The teacher needed to be able to recharge the robot easily and to determine if there was a wireless connection.

Modeling the stakeholders’ needs alerts the designers that people “in front” and “beside” the robot will react to it socially, although in this case the robot is only mildly anthropomorphic and has no way to move the tablet “head” of the robot. In the case above, the other students made eye contact with the robot as it moved down the halls and they helped when it was stuck. There was evidence that the other students, both in the class and in the general student population, actively engaged with the robot and the homebound student. This interaction was in contrast to the normal behavior of trying to avoid or minimize contact with a student in a wheelchair. In a fire drill, one group of students so strongly associated their classmate with the robot, that they did not want to leave the VGo behind.³³

The use of robots for homebound students also illustrates the trade-offs between autonomy and teleoperation. Teleoperation allowed the system to be inexpensive, simple to maintain and troubleshoot, and consistent. Autonomous navigation would, in theory, simplify movement of the robot, preventing collisions with desks and chairs, and making it easier to go through doors and move down hallways. However, students spend most of their time in the classroom and in close proximity to other students, desks, and tables. The most challenging maneuvers may be to align the robot to a desk or face the front of a classroom or to move appropriately in another student’s personal zone. When the robot travels down a hall, the hall may be filled with other students. The navigational autonomy has to be able to handle very close and socially sensitive distances in indoor spaces that are dynamically changing due to the movement of people. Navigating indoors is even more challenging because desks and chairs typically present narrow profiles that are hard for current sensors to detect. Any autonomous navigation algorithm or macros for automating going through a door or aligning to a desk have to be reliable and consistent in order to be worthwhile for users to agree to delegate control to the robot.

18.5.2 Cognitive Task Analysis

COGNITIVE TASK ANALYSIS (CTA)

NORMATIVE DOMAINS

*Cognitive task analysis (CTA)*¹⁷⁴ is appropriate when the robot is automating a well-established task or function. Well-established domains are referred to as *normative domains* because the normal tasks and standards of performance are known. In these domains, the robot is performing known tasks and is fitting into known command and information hierarchies. For example, the Defense Advanced Research Projects Agency (DARPA) Robotics Challenge (2012–2015) encouraged the development of robots that could get into a car designed for a human, drive to a facility, get out of the car, enter a

nuclear power plant or chemical facility, and turn valves to prevent an explosion. The expectation was for the robot to perform these tasks as fast as, and as well as, a human.

CTA relies on a suite of methods to elicit exactly what the task is and what the measures of performance are. The most commonly used method examines the existing practices, manuals, and written procedures. However, this is rarely sufficient by itself. A worker is implicitly expected to perform functions in addition to, or that go beyond, what is in the manual, such as noticing potential problems. The implicit expectation is that people are team players and have an incentive to reach the larger goal even when the mechanisms by which workers can facilitate the larger goals are not formally documented. Thus a CTA will often include interviews and table-top exercises to encourage stakeholders to reveal their implicit expectations.

CTA has a major disadvantage for robotics. As noted by Woods and Hollnagel,²²⁰ a robot does not completely replace a human because the robot is not as flexible or adaptable as a human and does not communicate and work as a teammate the same way a human does. Woods and Hollnagel refer to the expectation that a robot will replace a human in a task with no subtle changes to the overall system as the substitution myth. Interviews, surveys, and tabletop exercises can be misleading because it is hard for people to imagine how a new technology will impact their jobs. Therefore the robot designer has to project how the robot will change the task and will facilitate how the humans will react and adapt to the robot.

18.5.3 Cognitive Work Analysis

COGNITIVE WORK ANALYSIS (CWA)

FORMATIVE DOMAINS

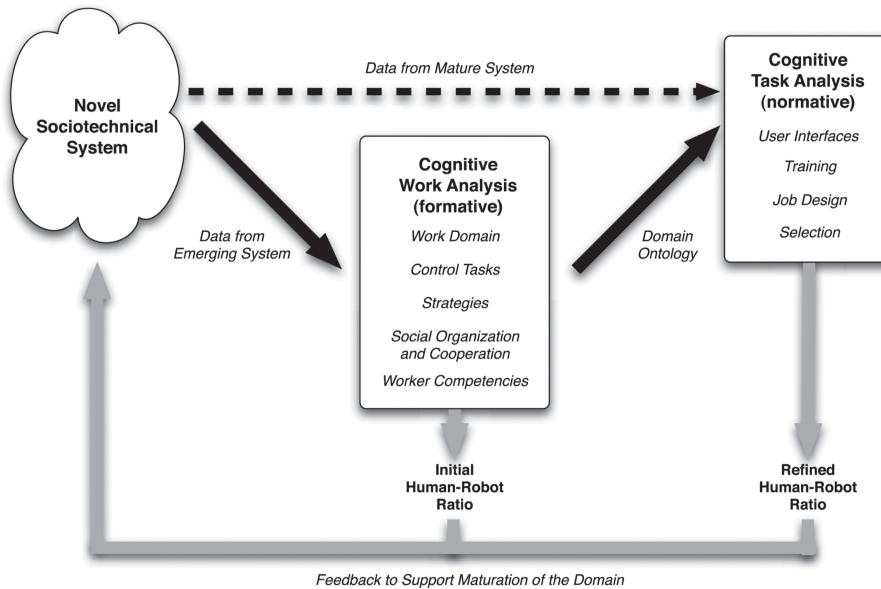
*Cognitive work analysis (CWA)*²¹² is appropriate when the robot is being used for new tasks or functions or will radically change the way by which work is performed. These novel tasks or functions are referred to as *formative domains* because the best way to accomplish work is still being formed. For example, using marine robotics to create maps of a wetland provides new capabilities. A marine vehicle could go into shallow spaces, record video of the view above the waterline, map the bottom, take continuous measurements of water quality, presence of sea life, and sea grass coverage, and so forth. This type of pervasive mapping is new, and the challenges for using a UMV are also new. As another example, consider a small UAV replacing a manned helicopter for monitoring wildlife. The UAV may be nominally doing the same tasks as the helicopter but the ways it accomplishes the tasks may be very different. Instead of having a dedicated pilot, a police officer or engineer may be expected to control the UAV directly.

CWA relies on a suite of methods, many of which overlap CTA and thus lead to confusion about the difference between the two. The intent of CTA is to capture five aspects of the domain:

- *Work domain:* The “big picture” or ecology of the application—the environment, types of robots that are feasible, and the general missions and expectations.
- *Control tasks:* The general workflow of the ways decision makers get information and make decisions using the robots, what the robots need to provide, and patterns of the ways robots would be used

- *Strategies*: How the robot could perform specific missions or tasks
- *Social-organizational*: How is the work divided up? What should the robot do, and what should be done onboard or offboard? What does the human do? What if there are interactions with multiple humans? How do the decision makers prefer to get information?
- *Worker competencies*: Who are the typical users and what are their backgrounds? What training will they need in order to perform operation and for maintenance of the robot?)

[Figure 18.2](#) compares cognitive task analysis and cognitive work analysis.



[Figure 18.2](#) How cognitive task analysis and cognitive work analysis relate to each other for human-robot interaction.

18.6 Natural Language and Naturalistic User Interfaces

A common user request is “just make it where I can tell the robot what I want!,” that is, design the robot interface to use natural language. As noted in chapter 1, natural language is a major area of artificial intelligence; it is a challenging and complex problem that has not been solved. There has been tremendous progress in speech recognition and associating words with tasks, for example, Siri, however language understanding remains an open research issue. Furthermore, the spoken word is only one aspect of natural styles of communication; people also gesture, pose, point, change the volume of their voice, and so on. Natural language is actually multi-modal.

18.6.1 Natural Language Understanding

Understanding verbal natural language has three distinct computational phases: *speech recognition*, where sounds are converted to phonemes and then to words, *language understanding*, where the words and resulting word structure are parsed for the most likely meaning, and *language generation and*

speech synthesis, where sentences are planned and generated to convey meaning.

Language understanding is the most difficult problem and is generally addressed in one of three ways. One way is to focus on the individual words themselves. This can be sufficient for telling a smart phone that you want to send a text message or play a particular song. However, the natural language understanding (NLU) system may have a limited vocabulary and may not be able to determine context. An interaction Doug Hofstadter had with Nicolai, a chatterbot developed by the US Army and described by Ray Kurzweil,¹¹⁰ d illustrates why context is as important as the words themselves:

Hofstadter: What are feet?

Nicolai: FEET ARE 12.0 INCHES.

Hofstadter: What are ears?

Nicolai: EARS ARE AUDITORY ORGANS FOUND ON ANIMALS.

Hofstadter: What are arms?

Nicolai: THAT INFORMATION IS CLASSIFIED.

FOUR GRAMMARS

Another approach to natural language understanding is to focus on the grammatical structure of the expressions. Noam Chomsky created a hierarchy of *four grammars* that are used by linguists and computer scientists.⁴² Chomsky's hierarchy is loosely described below.

- *Regular grammars*. This is the most basic grammar. It is too primitive to capture natural language and usually is not expressive enough for computer languages. It is sufficient for creating search queries.
- *Context-free grammars*. These are the foundation of computer languages which are based on non-deterministic pushdown automata. However, these overly precise and constrained grammars do not capture general natural language.
- *Context-sensitive grammars*. As seen with the feet, ears, arms example, context is important. Context-sensitive grammars, in theory, capture most natural language attributes. However, in practice, it is difficult to capture context in rules that a computer can apply, even for small, limited subsets of natural language.
- *Recursively enumerable grammars*. These are theoretical grammars that could be used to build a universal Turing machine. These grammars go beyond normal natural language.

CONCEPTUAL DEPENDENCY THEORY

Semantics is about the deep underlying meaning of expressions and are thus the real goal of artificial intelligence. There are three broad categories of algorithms and representations of semantics in natural language. One category attempts to create generalizable algorithms that are applicable to any natural language dialogue and do not depend on the word choices. The most famous algorithm is Schank's *conceptual dependency theory*, where the algorithm identifies the underlying speech acts

being expressed. Schank generated a list of speech acts including MTRANS, where one agent is trying to transfer mental information such as telling someone something, PTRANS, where an agent is talking about a transfer of the physical location of an object, ATTEND, where the agent is focusing on a stimulus or object, and SPEAK, where an agent is uttering a sound.

Figure 18.3 shows how conceptual dependency theory might decompose directives into speech acts. These acts can then map onto robot functions. Notice that “Please bring Wang Fang the orange bin over there” decomposes into the same speech acts as “Robot, see that orange bin? Bring it to Wang Fang.” The challenge is developing the larger context of the entire dialogue and keeping up with what vague identifiers like “it” refer to, which is called *pronomial reference*.

Robot, see that orange bin? Bring it to Wang Fang.

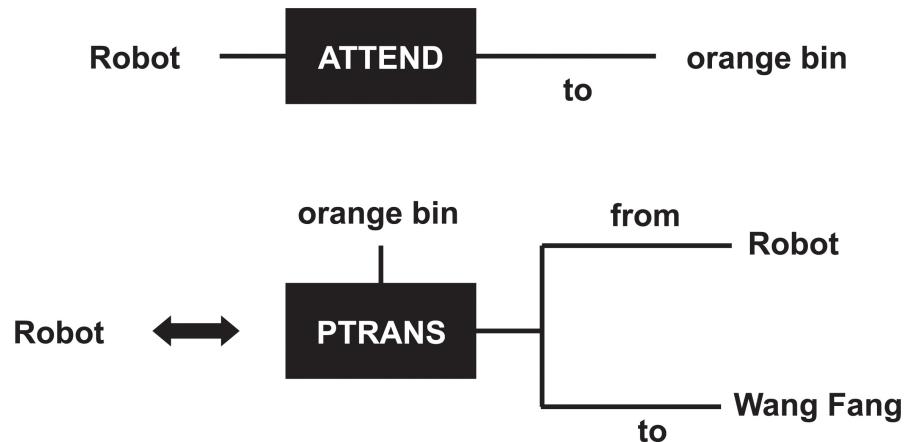


Figure 18.3 Example of conceptual dependency theory decomposing a directive into acts.

18.6.2 Semantics and Communication

Semantic understanding is hard because communication is vague and underspecified, but yet we often know what the other person means or wants. Semantic approaches to natural language attempt to make the implicit aspect of communication explicit. The two most notable approaches, common ground and belief-desire-intention (BDI), both assume that the one agent has a model of what the other agent is thinking and communicating about. However, research is not clear about how sophisticated or even how correct the robot’s model of the human or the human’s model of the robot has to be in order to be successful. Another reason why communication is hard is that meaning is not just derived from words. There is the tone of voice or prosody, the way we gesture, if we are smiling, whether we maintain eye contact and where we look, and other cues as to what we mean. Therefore robots will need to understand and express themselves through multiple modes of communication in order to comprehend directions given by a human and to show engagement or confusion.

18.6.3 Models of the Inner State of the Agent

COMMON GROUND

The *common ground* approach to semantics focuses on determining how a group of people know that they are talking about the same thing. One aspect of common ground is *symbol grounding* of perception. For example, if a person says, “Please bring me some coffee,” to a robot, and the two are in the kitchen near a coffee pot, the sentence may mean that the human is directing the robot to get a coffee cup, pour coffee for that coffee pot, and bring it to the human. Another aspect of common ground is *context*. If the person says, “Please bring me some coffee,” in reply to a robot who has announced it is going to the grocery store, the sentence may mean that the human is directing the robot to purchase a bag of coffee.

BELIEF-DESIRERS-INTENTIONS (BDI)

The *belief-desires-intentions (BDI)* approach goes much further than common ground by expecting the robot to have a model of what the human is doing and wants to accomplish. As the name implies, the robot is expected to understand that people have beliefs, desires, and intentions that influence communication. When a person says to a robot “Please bring me some coffee” in the context of grocery shopping, the robot has a model of the person. In this context, the robot infers that the person believes that s/he is eligible to give the robot orders. But the person’s beliefs do not obligate the robot. If the person cannot direct the robot, the robot would use its understanding of the person’s beliefs to reply that it cannot fulfill that request. The context also influences desires and intentions. If the normal supply of coffee is running low, the person intends the robot to resupply the larder. The person may desire a particular type of coffee, say whole bean breakfast blend, which the robot might infer from past experience with the human. The robot may also infer that the person desires to coffee at the best price. However, if a dinner party is coming up, the desire may be for a Kona or flavored coffee or an espresso roast, regardless of price, to meet the intention of a social event.

Both the common ground and BDI approaches assume that the human also has, or builds, a model of what the robot is doing and its goals. The human builds common ground and, forms expectations of, the robot based on many factors, one of which is the robot’s appearance. As noted earlier in the Uncanny Valley, the more lifelike a robot, the stronger the expectation that it will behave and deliberate like that life-form. However, the form of the robot also gives clues to its purpose and to its expected intelligence. For example, the iRobot Roomba does not look like a standard vacuum cleaner, which subtly prepares the user that it will not use a lawnmower scan of a room. It also does not look like it has any other capabilities, as there are no arms, obvious sensors, or an interface beyond buttons. The color can be an affordance, with black implying military or police use versus soft blue which is associated with medical workers or the yellow or orange of emergency workers. Other factors include the robot’s communication content. Choosing simple words and formal sentences help convey that the robot cannot handle vernacular expressions. Mannerisms, such as acting slow and deliberate, suggests the robot’s skills and intelligence are limited. Another factor in determining common ground and forming expectations is the human’s past experience with other robots and AI systems.

18.6.4 Multi-modal Communication

Communication is not limited to the content of verbal communication; it is multi-modal. Verbal communication can be reinforced by non-verbal communication, and verbal communication may not

be necessary at all for an agent to communicate its intentions. Major non-verbal modes of communication include, but are not limited to, expressions and gestures. Expressions are one mechanism by which an agent communicates its inner state. The major forms of expression are proxemics, which is the relative position of the agent to another agent, its pose, its affective facial expressions, and its tone of voice or prosody.

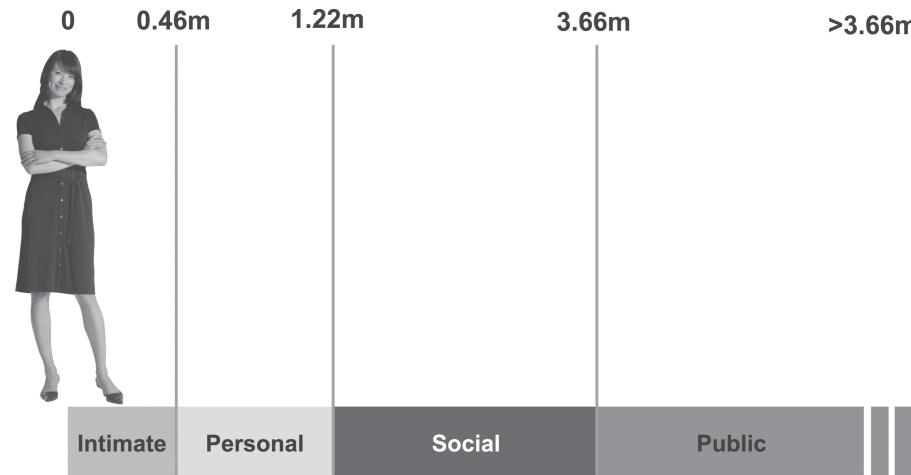


Figure 18.4 The proxemic regions around a person; distances may vary slightly with culture and gender.

PROXEMICS

Any robot, whether it is anthropomorphic or not, gives off multi-modal communication cues through *proxemics*; this is the most important nonverbal communication cue to implement correctly. The phrase “violating personal space” comes from *proxemics*. The classic works of Argyle¹⁰ and Hall⁸⁶ show that proximity of one agent to another enables different modes of interaction and implies intent. Proximity is divided into four regions around a person called zones, with the zone distances varying with culture (It is acceptable to stand closer when talking in some cultures), gender (Women prefer men to stand farther away), and situation (in a crowded subway versus in a park). The zones, with their recommended cues,²¹ are:

- *public zone*, 3.66 – 7.62 meters, or even further. In this zone the robot is outside of the area of interaction with a person. The person may be aware of the robot but not necessarily pay it any attention or be able to distinguish any cues from the robot as to its intent. The robot is far enough away that a human cannot hear it above the noise of the crowd or notice details, but a person can see a large display of color (a flashing light that indicates the robot is turned on) and the direction in which it is moving.
- *social zone*, 1.22 – 3.66 meters. In the social zone, a person has come close enough to another person that each normally would acknowledge and be aware of the other, such as two people nodding to be sociable as they pass each other in the hall in order. The human attends to whatever agents are in the social zone to determine if their presence connotes intent to interact. Therefore, as the robot enters the social zone, the person would pay more attention to the robot. The human would

be able to see the robot's orientation (Is it facing me?) and general posture (Does it appear submissive or threatening?) but may not be able to detect any subtle facial cues. The human can see changes in color and also hear the robot if it is sufficiently loud.

- *personal zone*, 0.46 – 1.22 meters. Any agent that enters a human's personal zone is presumed to have a specific intent to engage that person. The person subconsciously begins to react to cues to determine whether that interaction is threatening because the robot is now within striking distance. The human can see where the robot is facing, its posture, and any facial expressions, so it is important that these cues convey intent. A robot that is moving quickly or erratically would be considered a threat because it might hurt the human. Thus smoother and slower movements become increasingly important in creating confidence in the robot. Likewise more submissive postures, such as a lowered head or configuration and lowered gaze, help convey submissiveness and that the robot is aware of the person and will behave safely. The human should be able to hear the robot more easily above any ambient noise. Therefore the volume used at the public and social zones would have be reduced.
- *intimate zone*, 0.15 – 0.46 meters. The intimate zone is very close to a person, and people instinctively are uncomfortable when someone enters. People in the intimate zone are extremely vulnerable because they cannot move away fast enough to protect themselves. In general, a robot should stay out of the intimate zone unless it is in the process of making contact with the human or performing a necessary examination. Then it must exude reassuring verbal and non-verbal cues. In intimate zone, the robot is so close to the human that the human is able to see the robot's orientation but is unlikely to be able to discern the robot's body movements and posture. The human might not be able to see any changes in color. Since the robot is so close, the human can easily hear the robot so the robot's volume may need to be turned down. Thus the robot must move slowly, submissively, and with clear intent in order to reduce anxiety in the human.

It is important for a ground robot to conform to the proxemic expectations of a human, otherwise the robot will introduce anxiety or be perceived as creepy. Teleoperators who drive robots need to be aware that how the robot moves and how closely they place their robot surrogates near people will have a subtle impact on the quality of their interaction with those people. Bethel²¹ showed that simple accommodations to proxemics, for example, slowing the robot's velocity and lowering its height and volume can significantly improve people's comfort level with a robot as it enters their personal zone.

Proxemic rules do not appear to apply to unmanned aerial systems⁶⁶ operating near people, which means that people may allow UAVs to approach too closely or they may not react defensively and thus expose themselves to injury. It is unknown whether proxemic rules apply to unmanned marine vehicles such as diver-assistant robots.

GESTURES

DEICTIC GESTURES

SIGNALING GESTURES

Gestures play a particularly important role in giving directions to a robot and prioritizing information. Humans generally expect to direct a robot or clarify directives through *gestures*. *Deictic gestures* are pointing gestures, where the agent's head, eyes, hands, or arms indicate a referent in the

environment. Since the 1990s, the primary challenge in deictic gestures has been in using computer vision to extract the direction the human was looking or pointing to. *Signaling gestures* communicate commands such as “stop” and “come here.” Robotics researchers typically default to the standard signaling gestures used by crane operators or in military operations. Gestures can help prioritize information content. Consider how arm waving, especially creating a beat, accentuates key elements in a verbal directive.

18.7 Human-Robot Ratio

Discussing user interfaces leads to questions about how many people does it take to operate a robot, which is referred to as the human-robot ratio. In demanding remote presence applications, such as urban search and rescue and nuclear power plant decommissioning, the state of the practice is a two humans to one ratio. A study by Burke et al.³⁴ showed that two humans working together with shared displays are nine times better than one human at reaching and maintaining high levels of situation awareness in narrow or perceptually challenging environments. This phenomenon is analogous to driving to a new location where it is helpful to have one person drive and one person look for the address and a parking spot. Standard practice calls for two robot teams to perform manipulation tasks or for navigating in narrow regions that restrict movement, with one robot providing “overwatch” for the other.

SAFE HUMAN-ROBOT RATIO

The worst-case *human-robot ratio for safe operation*¹⁵⁴ is:

$$N_{humans} = N_{vehicles} + N_{payloads} + 1$$

Using this formula, the HRI for a small unmanned aerial system with a single camera would start by considering a three-person team. The formula suggests that there will be a pilot for the vehicle ($N_{vehicles} = 1$) who is concentrating on the operation of the platform, a mission specialist who is directing the mission and deciding what pictures to take ($N_{payloads} = 1$), and an external safety officer to ensure safety of the team and the platform (+ 1). Whether the team can be safely reduced to two or one people is then a matter of applying HRI analysis and regulatory constraints. If the mission is flying in close proximity to buildings at low altitudes, the pilot will likely have to stay fully engaged in supervision of even an autonomous robot in order to take control, if necessary. In some cases, the roles of the pilot and mission specialist may be combined. For example, in precision agriculture, a small UAS is typically used to fly a fixed path, and then the analysis is performed later by someone else. The pilot serves as the mission specialist in the sense of being responsible for the data. However the roles cannot always be combined as the pilot may not be trained as a mission expert. For example, the pilot may not have sufficient civil engineering skills to notice an important detail on a building damaged by a hurricane that needed further examination such as seen at Hurricane Katrina.¹⁷¹ In both situations, aviation regulations may require an external safety officer or visual observer.

Another example is an unmanned marine surface vehicle being used to inspect a bridge with a camera above the waterline and sonar below the waterline. The formula suggests a four-person team as the starting baseline for safety. However, a three-person team consisting of a pilot, camera specialist,

and sonar specialist may still be safe. The team may be further reduced to two people if the camera is not being used for inspection but just to help with navigation. In that case, the pilot can direct navigation, swapping between the camera and line of sight views, while the expert makes sure useful sonar data are collected. The safety of the vehicle and adherence to marine collision regulations can be handled by the pilot since regulations do not mandate an independent visual observer. It is unlikely that the pilot and mission expert roles could be combined without the single human risking becoming so focused on the below-the-waterline view that the human loses awareness of the vehicle relative to structures, other boats, and so forth. The possibility of a collision may be small, but the consequences are so high—from loss of the expensive platform to a boater drowning—that it is worth having a two-person team.

While the current state of the practice is a high human-to-robot ratio, there is the hope that increased robot intelligence can reduce the human-to-robot ratio and eventually enable a single operator to control multiple robots. Many researchers have looked at ways to reduce the cognitive workload by delegating tasks or portions of tasks to the robot or trying to estimate how long an operator can work on something else before checking in with the robot, most notably *neglect tolerance*.⁵³ There have been numerous barriers. One barrier is that reducing the cognitive workload does not mean that humans can necessarily spend their “freed” capacity to do anything else that is particularly useful. A second barrier is that people who have completely delegated a task to a robot and are working on a different task, or even multi-tasking by doing minor tasks like checking email, are often unable to rapidly re-engage control in an emergency. As seen in the human out-of-the-loop control literature⁹⁹ discussed earlier in chapter 5, if the human is expected to respond to unexpected events, that person may have to stay in the loop in order to be prepared to react fast enough to regain control before there are negative consequences, say production of flawed parts. A third barrier is that people are not divisible. If you need a person for a mission, that person has to stay available for that mission. For example, if the robot is autonomous except for a particular phase, say docking, that requires a human pilot, then human-robot ratio for the mission is 1:1. If two persons are needed to transport and set up the robot, the human-robot ratio is 2:1, regardless of whether a single person can be the pilot and mission specialist.

A common request, up there with “Why can’t I just tell the robot what to do?,” is “Why can’t the robot alert me when it is running into problems and tell me what to do about it?” The “the robot alert me” part of that request is certainly within current capabilities of artificial intelligence and indeed is standard practice in HCI and in autopilots—and should be standard practice in robotics. The second part is trickier as there is a difference between monitoring for deviations in execution and reasoning about the causality of those deviations. If the robot could reason about why it was having problems, it could just fix them on its own. Right now the best that systems can do is to display what could be relevant information in order to help the human reason.

While the human-robot ratio connotes the number of people to accomplish the mission, when the mission was primarily navigation, network connectivity has expanded access to the robot’s payloads in real-time. Independently of who is responsible for operations (setting up, driving, and maintaining safety), there may be dozens of people wanting to see the data and to direct the robot to get more. The users may create contention for the robot,⁶⁵ giving the robot (or operator) conflicting directions that it must sort out. It is fatiguing and distracting for the operator to order priorities. The robot could attempt to meet goals, but, unless the priorities are explicit, it would be difficult to maximize success. If a

remote user is given control over a payload or the robot, the user may not understand the difficulty or dangers in what they are asking the robot to do.

18.8 Trust

TRUST

A major concern of robot users is *trust*, specifically can the user trust the robot to work as expected? Trust is often synonymous with comfort in using the robot, for example, even if the user has been assured that the robot has been tested and works reliably, the user remains worried or is uncomfortable using the robot. Trust is also related to empowerment, for example, the robot enables me to do better, and I want to use it in order to improve my performance.

FIVE FACTORS INFLUENCING TRUST

Reliability is one component of trust but there are at least four other influences. Trust, as it relates to acceptance of a new technology, often means favorably meeting four of the five factors listed in¹⁷⁹ that influence how a person decides to adopt an innovation. The fifth factor in¹⁷⁹ is the relative advantage of the innovation, which is independent of trust. The four factors are:

- *Compatibility*, or the degree to which the robot is consistent with existing values, past experiences, and needs. A robot that requires a major change to the standard operating procedures introduces uncertainty and risk. A robot may make one task easier but make the larger set of tasks harder. Another example of compatibility is the similarity of the tasks the robot has been applied to in the past to the one it is being used for now. Applying a robot to new domain entails the risk of mismatches in performance and reliability.
- *Complexity*, or the degree to which the robot is perceived as being difficult to use. At the 9/11 World Trade Center collapse, responders declined to use an iRobot Packbot saying it looked too experimental.¹⁴¹ The “looked too experimental” comment stemmed from the appearance of the operator control unit. The unit physically consisted of a computer game joystick attached to a laptop and looked more like a toy than something that would be used in life and death situations. The interface display was a set of windows designed for the engineers, and it looked complicated. The responders chose instead to use a less physically capable Foster-Miller, Inc. SOLEM robot that had a dedicated operator control unit with few options and ruggedized connectors.
- *Trialability*, or the degree to which the robot can be experimented with on a limited basis. As would be expected, familiarity encourages trust. The more a user can train with a robot in a wide variety of scenarios, the more the user trusts the system. In this case, trust, or lack of trust, results from the users developing an internal model of how the robot meets, or deviates, from their expectations. If the user is expected to work with, or beside, a robot with no training, the robot has to be designed to elicit trust both subconsciously and consciously.
- *Observability*, or the degree to which the results of the robot’s actions or deliberations can be observed. Observability is also referred to in HCI research as visibility. Early HCI researchers found that users grew uncomfortable when the computer did not respond after a short time to queries or commands; the users did not know if the computer was working on the task or had crashed. To

overcome this, the hour glass, watch, or spinning rainbow icons became commonplace. Users do not necessarily need to observe the details of what a robot is “thinking,” but they will want to know what the robot is trying to do. User interfaces for robots now routinely include icons and status panels to help the user comprehend the state of the robot. More advanced user interfaces may display the progress on a task.

18.9 Testing and Metrics

Given that human-robot interaction is important, how does a roboticist determine whether the system supports good HRI? Often the answer is that the quality of the HRI and the reason for its quality must be inferred from watching people and robots in action or in designed experiments. The methods and metrics for formal experiments on human-robot interaction typically try to reproduce, or stage, the tasks, environment, types of people, and robots that are involved for a particular domain. The data that are collected fall into four classes that produce at least 42 different metrics.¹⁴³ When testing human-robot interaction, it is helpful to use at least three different classes of data collection to make sure that useful data are acquired and to help cross-check and make stronger inferences.²⁰ Testing human-robot interaction may require an institutional review board approval to ensure the safety and privacy of the participants.

Formal experimentation of human-machine systems is well understood, and courses in experimental design are taught in psychology or industrial engineering departments. These courses focus on designing replicable experiments with sufficient data to be statistically significant and repeatable. Repeatability and statistical significance are often at odds with the uncontrolled messiness of a human-robot system working at scale. Because formal experiments generally concentrate on a narrow aspect of the system, it is important to make sure the experiment replicates as much of the expected environment, task activities, and human engagement as possible; otherwise the results may not transfer to the real world. Unlike many psychology experiments which can make use of volunteer first-year psychology students as participants, robotics may have distinct classes of users, and the demographics of those users may make a difference to the experimental results. For example, using students as subjects in experiments to project how highly trained astronauts or firefighters would use a robot is unlikely to produce meaningful results.

18.9.1 Data Collection Methods

WIZARD OF OZ (WOZ)

Experiments with humans interacting with autonomous robots generally take one of three forms. One form experiments with the working autonomous system. This may not be possible, in some cases, given the necessary degree of sophistication; for example, researchers studying the Uncanny Valley may not be able to build a robot that can autonomously move and act like a human. The difficulty of building robots with the desired autonomous capabilities leads to *Wizard of Oz (WOZ)* forms of experiments, where the experimenter is like the wizard in the movie *The Wizard of Oz* hiding behind the curtain and controlling the robot. To the participant in a Wizard of Oz experiment, the robot appears to be autonomous. The third form of experiments is computer-based simulations. The popularity of

computer-based video games makes it natural to exploit that technology. These simulators tend to be more useful for experimenting with control of robots, rather than for exploring how people will interact with the robot. However, mixed-reality simulations, where the human works with a robot projected full-size on a screen or a virtual reality CAVE, can be helpful.

Regardless of the form of the experiment, the purpose is to collect meaningful data. The four broad classes of methods for collecting data from an experiment are:

INTERVIEWS, SURVEYS, JOURNALING

- *Interviews, surveys, and journaling.* These methods capture the subjective impressions of the technology post hoc, or after the interaction. Surveys can be long and involved or short, for example the 10 question System Usability Scale.^{98;210} These methods are often not sufficient by themselves because respondents may have forgotten problems that occurred during the interaction, may not want to make the experimenter feel bad, or may hold back for fear that negative feedback will impact their job.

OBSERVATION

- *Observation.* Observation can be done using ethnographic video and note-taking, which captures the observer's subjective interpretation of what is going on. The observer is acting as a silent "fly on the wall." Ethnography in the field can be standardized with the use of templates such as the one in figure 18.5. The subjectivity associated with ethnography can be reduced by applying formal protocol analysis methods in which multiple observers independently score the activity using a coding scheme such as RASAR-CS. The coding scheme defines events such as when and how the operator determines the robot's state, performs a specific task with the robot, and so. The events defined in the coding scheme are linked to a large HRI measure; for example, the event of "the operator asking if their partner can tell whether the robot arm is raised all the way" would indicate that the operator is working at Level 1 or 2 of situation awareness. Coding is subjective, and, to reduce errors, multiple observers, called raters, are needed to review and score the video and audio. The independent scores are then analyzed for inter-rater agreement.

	ROLE 1	ROLE 2	ROLE 3	MULTIPLE	TOTAL
distracted					
slips					
mistakes					
confused/hesitant					
asked questions					
bugs					
ergonomics					
TOTAL					
OBSERVATIONS:					

Figure 18.5 Form used for recording field observations about human-robot interaction.

BIOMETRICS

- *Biometrics.* Aspects of human-robot interaction can be measured or inferred from physical movements and physiological reactions. Eye-tracking has been used extensively in HCI to determine what windows and icons are being looked at by a user. Changes in stress due to increased workload or fear and anxiety can be signaled by changes in respiration rates, heart beat, and galvanic skin response. Measuring biometrics is generally intrusive as the human has to wear biometric sensors, see [figure 18.6](#). The sensors often constrain the wearer's movement or are uncomfortable. The measurements can be affected by the wearer's sweat and environmental factors, interfering with reliable measurements in outdoor field conditions. Biometrics are typically used in laboratory experiments or highly controlled field experiments.



Figure 18.6 An example of a HRI study participant being fitted with biometric sensors.

SPECIALIZED TESTS

- *Specialized tests.* HRI experimenters may suspend the activity and ask the participants to fill out a survey or take a test. For example, the experiment may be stopped at several predetermined points and the participant asked to complete the *NASA-TLX survey*,⁸⁷ a popular tool. The survey is subjective, but because it is administered immediately following an activity, it can be more accurate than a post hoc survey. A less subjective approach may be to stop the experiment and administer a test. One such test is the *Situation Awareness Global Assessment Technique* (SAGAT).⁶⁹ The disadvantage of these types of tests are that they are disruptive and may interfere with the fidelity of scenario, and thus the results may be compromised.

18.9.2 Metrics

CATEGORIES OF HRI METRICS

Once collected, data about the system can be examined using metrics of human-robot interaction. Murphy and Schreckenghost¹⁴³ identified 42 metrics that various researchers have proposed or used. The majority (29) of the metrics measured some aspect of the system's performance, but some metrics measured the human's response during the interaction to the robot or the robot's activity. The system performance metrics were further divided into five categories:

PRODUCTIVITY

- *Productivity.* Was the human-robot system effective? How much time was spent in autonomous

operations versus manual operations?

EFFICIENCY

- *Efficiency.* Was the human-robot system faster than a human-only system? How many humans did it take to operate the robot?

RELIABILITY

- *Reliability.* How many times did a human have to intervene?

SAFETY

- *Safety.* Did the robot pose a risk to humans? Did it have awareness of humans in its work envelope?

COACTIVITY

- *Coactivity.* Did the division of effort between the human and robot work well? Did the robot require unexpected supervision of delegated tasks? How often did the robot encounter a problem or have to ask for help?

18.10 Human-Robot Interaction and the Seven Areas of Artificial Intelligence

Good human-robot interaction requires artificial intelligence. Certainly natural language interfaces require natural language understanding, one of the seven core areas. However, knowledge representation is essential, especially for capturing common ground and beliefs, desires, and intentions. Inference is important because interaction is sensitive to the context. The impact of computer vision on human-robot interaction is often overlooked, but it is needed to perceive the non-verbal communication (e.g., gestures, pose, facial expressions) as well as the situated common ground (e.g., the red coffee cup over there). Learning also contributes to the robot's ability to adapt itself to individual preferences. Human-robot interaction also engages distributed AI, because in many cases, the human and robot are dividing up tasks and responsibilities.

Good human-robot interaction also depends on the overall autonomy of an agent for a task. Approaches that express autonomy as a degree of initiative⁴⁹ covered in chapter 3 are attractive because they explicitly formulate what is being delegated to the robot with what expectations; the autonomy is framed as a relationship to the human, not to the task. Recall that in the “no autonomy” case, the robot is delegated a task but follows rigid programming associated with a task or goal, much like cruise control on a car. In process autonomy the robot can choose the algorithm or process to achieve its task goals. The human knows that the robot is working on the task, but does not need to supervise or approve the details. In systems-state autonomy, the robot can generate and choose between options to achieve its goals. The robot can decide whether to perform the task itself or ask the human to do it. This type of autonomy goes beyond alerting that an out-of-the-loop control condition is eminent. It is more about understanding the capabilities of the team members with respect to accomplishing the goal. Intentional autonomy allows the robot to change its goals to meet the intent of its roles within the team, such as helping the team by taking over some functions of another team player who is distracted or not performing well. Finally, under constraint autonomy, the robot can create its own roles and goals, by relaxing constraints on its actions.

Each of these degrees of initiative implies different methods of interaction, degrees of trust, and testing. For example, in intentional autonomy, the robot has to notice and diagnose the impaired performance of a team member in order to take over the appropriate portion of the task. The degree of trust for constraint autonomy is much higher than for process autonomy. Testing and evaluation are likewise much different and more demanding as more initiative is given to the robot.

18.11 Summary

Human-robot interaction is fundamental in intelligent robots because humans are involved with the robots as operators, team members, and bystanders. The robot is generally operating in one of three fundamental modes: as a taskable agent, as an assistive agent, or providing remote presence. The interaction with a human can be a combination of physical, cognitive, and social/emotional modes of cooperation. Understanding and designing good human-robot interaction builds on work from the fields of human computer interaction, psychology, and communications, but goes beyond them. For example, HCI offers Eight Golden Rules for designing user interfaces, but provides no principles for avoiding the Uncanny Valley. The first step in designing human-robot interaction for an application is to understand the application domain, which is usually accomplished using cognitive task analysis for normative domains or cognitive work analysis for formative domains. Once the domain is understood, the roles and responsibilities for each agent, the dynamics of the environment and task, and any regulatory constraints should be clear. With that, the safe human-robot ratio formula can be applied to determine the requisite manpower. Each user or role may need a different user interface in order to construct and maintain all three levels of situation awareness. A pilot may require situation awareness of navigational and platform health, but a payload specialist may not care about those attributes and need situation awareness of the mission progress and payload operation. Natural language communication, whereby the user provides explicit verbal and gestural interaction and implicit pose, gesture, and facial expressions, is attractive but is still beyond the practical abilities of AI. There are no principles for dividing responsibilities between the human and the robot; however, the designer has to be aware that the implicit assumptions people have about delegating and sharing tasks with other people also apply to robots. Unfortunately, robots do not capture the implicit aspects of working in groups. When these implicit assumptions about delegation and sharing are not explicitly addressed, the human out-of-the-loop control problem occurs, where a task or process encounters a problem and a human, who is engaged in some other task, has to take control unexpectedly but has no capacity or time to handle the problem. Although trust is increased by the non HRI factors of reliability and trialability of the robot, good human-robot interaction should increase trust further. The interaction design can increase trust by being compatible and consistent with the organization's existing procedures and worker skill-sets, by reducing the complexity of the system and even the appearance of complexity, and by providing user interfaces which increase observability or visibility of what the robot is doing. HRI requires a different style of testing and evaluation than does algorithmic performance. Experiments with human subjects will likely be regulated and require review by an institutional or governmental authority to ensure participant safety and privacy.

Returning to the questions posed in the introduction, the answer to the first question: *How many people does it take to run a robot?* is that the human-robot ratio depends on the ecology: the

environment, the task, and the robot itself. A worst-case ratio can be computed using the formula for the safe human-robot ratio $N_{humans} = N_{vehicles} + N_{payloads} + 1$. The designer can then determine if the ratio can be reduced if the user interfaces, cognitive limitations introduced by high workload and human out-of-the-loop control problems, and regulations permit. *How to divide up responsibilities or roles?* also depends on the ecology. A more capable robot can be delegated more functions and initiative. Defining all the stakeholders and their roles presents a more subtle challenge. It is easy to forget stakeholders, such as the educational service program manager needing to see the status of all robots and to troubleshoot remotely, but those stakeholders can be critically important for the adoption of robotics. The answer to: *How do people like to interact with robots?* is a bit facile as people like to interact with a robot as conveniently and naturalistically as possible, but convenience and naturalism can be hard to achieve. People “behind” the robot favor multi-modal user interfaces that are tailored to their needs, assist in reaching Level 3 SA, and increase trust that the robot is doing its delegated share of the work. People “in front” and “beside” ground robots subconsciously interact socially with them, following the Computers Are Social Actors paradigm.¹⁷⁵ People infer intent and competence from the robot’s appearance. They also respond to the degree of naturalism in movements and mannerisms, following the Uncanny Valley curve. They react to robots violating their personal space, following proxemics. Considering the question: *Do robots and people need to understand each other (e.g., have a shared cognitive model)?*, the degree to which robots and people need a common ground or to have models of each other’s beliefs, desires, and intentions is unknown. Having explicit cognitive models is related to interpersonal trust but may not necessary for trusting the robot to do a well-defined tasks. *Why can’t I can just tell the robot what to do, because then I don’t have to worry about interfaces and controls?* is, unfortunately, a flawed question in that it ignores the challenges of designing robots that can understand natural language.

Designing good human-robot interaction concludes the design of the interactive layers. The next chapter addresses the design of autonomous robots from a broader perspective and attempts to pull together the themes posed so far in the book.

18.12 Exercises

Exercise 18.1

Describe the differences between a user interface and human-robot interaction.

Exercise 18.2

Define human-robot interaction and describe at least one way in which it uses insights from each related area: HCI, psychology, and communications.

Exercise 18.3

If given an application, identify whether it involves physical, cognitive, or social/emotional modes of HRI cooperation.

- a. A doctor is using a robot to view a victim trapped in rubble; the robot interface shows reminders about the features to look for
- b. A surgeon has a robot drill into a bone on a patient in order to implant a hip replacement
- c. A doctor talks with a physician’s assistant 3,000 km away in another country via a telepresence robot.

Exercise 18.4

Which one is NOT a way to increase trust in a robot:

- a. ensure compatibility
- b. decrease complexity
- c. increase observability
- d. add redundancy
- e. increase reliability
- f. support trialability.

Exercise 18.5

Describe the four methods for collecting human-robot interaction data. Who collects the data for each: the operator or the scientist?

Exercise 18.6

Describe the five categories of human-robot interaction system metrics.

Exercise 18.7

Describe the three approaches to language understanding and how they might be used by robots.

Exercise 18.8

Define:

- a. team
- a. user interface
- b. situation awareness
- c. sensemaking
- d. Uncanny Valley
- e. human out of the loop control problem
- f. common ground
- g. BDI
- h. WOZ.

Exercise 18.9

What are the differences between deictic gestures and signaling gestures? When would a human use them to direct a robot? When would a robot use these gestures to communicate with a human?

Exercise 18.10

True or false: In the human out-of-the-loop control problem, the human is a problem and thus the robot should be fully autonomous to get the human totally out of the control loop.

Exercise 18.11

True or false: Communication is enhanced when the agents communicating have a model of what the other agent is thinking and communicating about. There are many different models, two of which are common ground and BDI.

Exercise 18.12

Consider the development of an intelligent wheelchair that would climb stairs. Contrast taskable agent and assistive agent approaches to this problem. What would be the general flavor of the three resulting systems? What would be different about each of the user interfaces?

Exercise 18.13

Write out the formula for the safe human-robot ratio and explain each term.

Exercise 18.14

How do you decide what any kind of new technology, not just robotics, is likely to work? Is it based on appearance? Free trials? Referrals from friends? How do you think you would judge whether a robot could be trusted for a task?

Exercise 18.15

What is the difference between situation awareness and sensemaking?

Exercise 18.16

Consider an operator using a UAV as a taskable agent to photograph a field of corn, where, once launched, the UAV will autonomously fly back and forth in a regular pattern. Now consider an operator using the same UAV to provide remote presence for a specialist investigating a building collapse. Describe the differences in the situation awareness needed for each of the applications.

Exercise 18.17

Describe the differences between speech recognition, language understanding, and language generation.

Exercise 18.18

What does the Media Equation say?

Exercise 18.19

List at least three different modalities for communication.

Exercise 18.20

Describe what deictic gestures are and explain how a user would use such gestures to control a robot. Name three tasks, and gestures, that deictic control might be used to interact with a robot.

Exercise 18.21

Think of one normative and one formative domain where a robot is being used. What is the difference? What makes them normative or formative?

Exercise 18.22

Natural language understanding researchers often claim that NLU requires expertise in all areas of AI. Can you think of one, or more, of the seven areas of AI that is not involved in NLU?

Exercise 18.23

Look up the names of commercially available UAVs. Do you think the names reinforce the public's fear of UAVs or reduce those fears?

18.13 End Notes

For a roboticist's bookshelf.

Two essential books are not about robots but are at the foundation of human-robot interaction. *The Media Equation* by Byron Reeves and Clifford Nass¹⁷⁵ presents the Computers Are Social Actors paradigm in a compelling summary of years of research that has motivated more than a decade of human-robot interaction research. Yes, we would all like to think we are smart enough not to react subconsciously to robots, but in the end, Reeves and Nass' studies show we are not. *Affective Computing* by Rosalind Picard¹⁶⁹ was the breakthrough book that encouraged computer scientists and roboticists to think of how people subconsciously interacted with technology.

TV sometimes gets it right and makes us laugh.

The "Succession" episode of the NBC TV series *30 Rock* contains the most succinct (and funniest) explanation of the Uncanny Valley, using the actual figure from Mori's paper applied to Star Wars, videogaming, Tom Hanks, and pornography. A clip is on YouTube.

TV sometimes gets it right and makes us cry.

The “Darmok” episode of *Star Trek: The Next Generation* provides a poignant example of why identifying words is not the same as understanding what someone is saying. In this heartbreakingly touching episode, the lack of a common ground, despite apparent similar desires and intentions, keep two species at odds with each other. Admittedly, there are no robots in the episode besides Commander Data.

Two Movies to Compare and Contrast.

The 2012 movie *Robot and Frank* with Frank Langella and Peter Sarsgaard offers a view of an assistive robot interacting with an unhappy senior citizen. The premise touches on the very real problem in the assistive robot market—the children of an aging parent may buy the robot but if the senior citizen does not want it, it will not be used. *Robot and Frank* is an optimistic view of assistive and entertainment robots, in stark contrast to the 2015 movie *Ex Machina* where female humanoid robots are used as servants, general ego-boosting, and, of course, sex. Watch Oscar Isaac’s hubris in *Ex Machina*, and you may be disappointed that Poe Dameron in *Star Wars: The Force Awakens* didn’t die.

You can always trust a uncanny robot.

When the creepy child-like robot, who illustrates the valley in the Uncanny Valley, announces, in the opening of the 2014 very bad robot movie *Robot Overlords*, that robots never lie, it is a sure thing that robots lie.

V

Design and the Ethics of Building Intelligent Robots

Contents:

- [Chapter 19: Designing and Evaluating Autonomous Systems](#)
- [Chapter 20: Ethics](#)

19

Designing and Evaluating Autonomous Systems

Chapter Objectives:

- Describe the five design questions that influence the degree of algorithmic sophistication needed for a specific autonomous capability (*What functions does the robot need to provide? Which planning horizon do the functions require? How fast do the algorithms have to update outputs? What type of world model does the robot need? and Where does the human come in?*)
- Give examples of each of the three sources of robot failures: *external failures, physical failures, and human error.*
- Given a goal for evaluating a robot, choose and justifying the appropriate *type of experimentation: controlled experimentation, exercise, concept experimentation, and participant-observer.*
- List the six categories of data to collect during experimentation (*log of activity, context, robot's eye view, robot state, external view of the robot, human-robot interaction view*) and discuss strategies for acquiring the data.

19.1 Overview

The book started with chapters on autonomy, automation, and architectures in order to provide a framework for learning about the elements of artificial intelligence used in robotics. This chapter returns to the concepts introduced in those chapters to revisit the question: *How do I design an intelligent robot?*

The book has espoused an ecological approach to design, where the robot and its intelligence fits the ecological niche of the system. Recall that the ecological approach considers how the robot, environment, and task influence the successful design. The designer must consider what the *robot* can do: What can it sense, directly or indirectly, for example range (chapter 10)? What are its locomotion and manipulation abilities? What compute power is needed (chapters 9 and 14)? What can it do that a human cannot or what must it do because a human cannot (chapters 5 and 18)? The designer must also consider the *environment*: Can it be modeled as a closed world or engineered to support automation (chapter 3)? What does the environment afford (chapter 6)? Are those affordances sufficient or is recognition required (chapter 7)? What is the scale and traversability of each region of the environment

chapter 15)? Finally, the *task* itself impacts design: Is the task navigation or something else (chapter 13)? Can the goals be better served with a multirobot system (chapter 17)? What are the implicit goals, such as trust and health monitoring (chapter 3)? Do the goals require deliberation (chapter 3)? The ecological approach can be applied to the design of specific capabilities as well as the overall robot system design. Consider learning (chapter 16), where creating a successful learning algorithm requires understanding what the robot is trying to do as well as what the robot can sense and learn from; this invokes the ecological approach to design, which considers the task, robot, and environment.

Recall also that chapter 4 described how the hybrid architecture is more than a programming style; it guides the designer in determining how much artificial intelligence is needed by posing five key design questions. There are: *What functions does the robot need to provide? Which planning horizon do the functions require? How fast do the algorithms have to update outputs? What type of world model does the robot need? and Where does the human come in?*

The chapter concludes by attempting to bring the previous chapters together within a practical design context. It begins by outlining how to design a specific autonomous capability starting with the general design philosophy, reviewing the five questions a designer has to ask from an architectural perspective, and giving a case study of the successful use of this design process. Next it reminds the readers that the various taxonomies and metrics introduced throughout the book, but especially in chapter 3, do not substitute for good system design of intelligence. The chapter concludes with a section on holistic evaluation of system designs, such as identifying the failures, the types of experiment to conduct for a particular evaluation goal, and recommended data to collect.

19.2 Designing a Specific Autonomous Capability

The canonical operational architecture presented in chapter 4 is more than a framework, it can also be used as an aid in designing an autonomous capability as it presents five questions that a designer must answer. These questions orient the designer toward satisfying the explicit, and especially the implicit, expectations of the capability. Designing autonomous capabilities can be confusing if the designer treats the layers in the operational architecture as a roadmap. Metrics, such as Autonomy Levels for Unmanned Systems (ALFUS),⁹⁵ which try to compare autonomous capabilities, can lead designers to think that a more complex situation requires more sophisticated intelligence, but as was seen in chapter 6, animals with lower intelligence are highly successful in extremely complex environments. In the end, the strategy should be one that determines the necessary capabilities and then examines how those capabilities can be achieved given the hardware, software, environmental, and human constraints.

19.2.1 Design Philosophy

The design philosophy advocated in this book starts with the tenet that an autonomous capability involves all of the functions and layers in the canonical architecture. Some of these functions may be handled by the human, often because of limitations preventing a computer from performing those functions. Forgetting to account for how these functions are handled can result in unexpected manpower demands, as seen by the surprisingly high manpower costs encountered by the US military in deploying unmanned aerial systems.¹⁴⁴ On the other hand, a designer may decide that an autonomous capability does not require deliberation and that the “fly-at-the-window effect” brittleness is acceptable.

Likewise a design for an autonomous vehicle that is being used to opportunistically explore a cluttered environment might emphasize adding autonomous deliberation rather than navigation, where the robot provides mapping and health monitoring while the user decides where to go and what is interesting to examine further. The point is that the designer should consciously decide what autonomous capabilities are needed for the mission and determine how those capabilities will be implemented from a systems perspective.

A mission will likely require multiple autonomous capabilities. For example, waypoint navigation might involve four capabilities: planning a route, instantiating behaviors to follow a road and avoid obstacles, setting a monitor or timer to trigger replanning if the robot does not reach a waypoint by a certain time, and providing a user interface that displays the progress and presents a diagnostic display if there is a problem. A second autonomous capability for the robot could be searching for intruders with an infrared camera that uses software to disregard groups of small heat signatures that are consistent with animals and alerts the operator to the presence of a person. A third autonomous capability for the same robot might be fault detection and recovery which monitors the health of subsystems and triggers the substitution of an equivalent sensor or component.

A system with multiple autonomous capabilities will likely have differing needs for world models, monitoring, and interfaces. The operational architecture is sufficient to remind the designer that these functions should be considered, but the architecture does not help the designer consider how to coordinate these different demands. How to manage the different design demands is the role of the systems and technical architecture views; they describe how to design the monitoring function or user interface to support different capabilities.

19.2.2 Five Questions for Designing an Autonomous Robot

FIVE QUESTIONS FOR DESIGN

There are *five broad questions* that a designer should ask when designing each specific autonomous capability.

What internal functions does the robot need to perform for the capability? Does it need to generate a plan? Will it monitor the plan execution? Will it need to select resources or strategies based on different weather conditions or terrain types? How will these strategies be converted to behaviors and turned on and off (implemented)? How will it execute the behaviors? Will there be any online learning during the mission? Will there be learning after the mission, either from the operator or offline?

What planning horizon (present; present and past; present, past, future) does the capability require? The planning horizon helps shape expectations. Reactive capabilities, such as reflexive obstacle avoidance, work in the present and are very fast, but, like the human patella's knee-jerk response, they may not be optimal. Optimal algorithms generally require maintaining state about the past and often projecting outcomes into the future which can involve constructing models. Each planning horizon is an implicit commitment to internal models and structures.

How fast do the algorithms enabling that capability have to execute? Certainly many UAV flight control algorithms have to run at extremely high rates, though recall these normally are not included in the “artificial intelligence” formulation of an autonomous robot. But if a robot has to avoid a moving obstacle or projectile, that capability may have to be executed very quickly. This execution speed indicates that the capability may reside solely in the reactive layer, because the deliberative

functionality is slower due to the extra computation time required to maintain state and world models. Often the desired perception for capabilities takes longer than is desired, and, as was seen in chapter 6, sensor data are often processed both reactively and deliberatively; for example, a robot may use the sensor data to avoid a collision while another process simultaneously uses the sensor data to determine that the object was a nerf ball and that it is safe to resume its path.

What type of models does a specific algorithm need? Local? Global? Both? A common assumption is that autonomous capabilities require sophisticated, detailed models. The first iRobot Roomba models were successful precisely because they did not create a model of the world; they did not need to map out a room in order to clean it. Likewise, many systems create models based on the sensor capabilities versus the area of interest; if the work envelope for a robot is defined by a three meter radius, why build a detailed model that spans a nine meter radius? Are 3D representations essential for the robot or are they simply attractive for visualization? Modeling adds significant computational complexity and processing bottlenecks, and it can create blind spots in autonomy—if an object or condition is not in the model, it does not exist.

Where does the human come in? As seen in chapter 18, a human is always involved with the robot in some fashion, and this must be accounted for in the design from the beginning. Is the human merely tasking the robot but the robot is fully in charge of instantiating, monitoring, and dealing with any problems over the entire mission? This is unlikely because of the current state of artificial intelligence and that missions are formative. Therefore, the algorithms are limited and the understanding of the domain is incomplete that operators have to be available to handle inevitable failures of autonomy and surprises. Are the missions really completely delegated to a robot, that is, taskable agents, or are the missions ones of remote presence or of assistance to the human operator? If the human is involved in one or more capabilities, which agent—robot or human—is performing what functions; for example, is the human monitoring the progress of the robot? Are any of the functions shared by the human and the robot and, if so, how? For example, the robot can recommend a path or resource allocation, but the human operator would make the final decision.

19.3 Case Study: Unmanned Ground Robotics Competition

ECOLOGICAL DESIGN PROCESS

The Colorado School of Mines' (CSM) entry in the 1994 Unmanned Ground Robotics Competition¹⁴⁸ provides an example of the design process. Even though the event is decades old, the simplicity aids in seeing how design principles connect to results. The objective of the competition was to have a small unmanned vehicle (no larger than a golf cart) autonomously navigate around an outdoor course of white lines painted on grass. The CSM entry won first place and a \$5,000 prize. The team used an ecological design process consisting of seven steps. Each design step is first presented in boldface and discussed. What was actually done by the CSM team follows in italics. This case study illustrates the effective use of remarkably few behaviors and the use of affordances combined with an understanding of the ecological niche. It also highlights how even a simple design may take many iterations to be workable.

Step 1: Describe the task. The purpose of this step is to specify what the robot has to do to be successful.

The task was for the robot vehicle to follow a path containing hairpin turns, stationary obstacles in the path, and a sand pit. The robot that went the furthest without going completely out of bounds would be the winner, unless two or more robots went the same distance or completed the course, then the winner would be the robot with the fastest time. The maximum speed was five mph. If the robot went partially out of bounds (but one wheel or a portion of a tread remained inside), a distance penalty was subtracted. If the robot hit an obstacle hard enough to move it, another distance penalty was levied. Therefore, the competition favored an entry which could complete the course without accruing any penalties over a faster entry which might drift over a boundary line or bump an obstacle. Entrants were given three runs on one day and two days to prepare and test on a track near the course; the times of the heats were determined by lottery.

Step 2: Describe the robot. The purpose of this step is to determine the basic physical abilities of the robot and any limitations. In theory, it might be expected that the designer would have control over the design of the robot itself, what it could do, what sensors it carries, and so forth. In practice, most roboticists work either with a commercially available research platform, which may have limitations on what hardware and sensors can be added, or with a relatively inexpensive kit type of platform where weight and power restrictions may impact what it can reasonably do. Therefore, the designer is usually handed some fixed constraints on the robot platform which will impact the design.

In this case, the competition stated that the robot vehicle had to have a footprint of at least 3 feet by 3.5 feet but no larger than a golf cart. Furthermore, the robot had to carry its own power supply and do all computing onboard (no radio communication with an off-board processor was permitted), and carry a 20-pound payload.

[Figure 19.1](#) shows the CSM entry, Omnidot. Omnitech Robotics, Inc., donated the materials. The vehicle base was a Fisher Price Power Wheels battery-powered child's jeep purchased from a toy store. The base met the minimum footprint exactly. It used Ackerman (car-like) steering, with a drive motor powering the wheels in the rear and a steering motor in the front. The vehicle had a 22° turning angle. The on-board computing was handled by a 33MHz 486 PC using Omnitech CANAMP motor controllers. The sensor suite consisted of three devices: shaft encoders on the drive and steer motors for dead reckoning, a video camcorder mounted on a mast near the center of the vehicle, and a panning sonar mounted below the grille on the front. The output from the video camcorder was digitized by a black and white framegrabber. The sonar was a Polaroid® lab grade ultrasonic transducer. The panning device could sweep 180 degrees. All coding was done in C++.



Figure 19.1 Omnidbot, a mobile robot built from a Fisher Price Power Wheels battery-powered toy jeep by students and Omnitech Robotics, Inc.

Due to the motors and gearing, Omnidbot could move at a speed of only 1.5 mph. This limitation meant that it could win only if it went farther with fewer penalty points than any other entry. It also meant that the steering system had to have at least a 150 ms update rate or the robot could veer out of bounds without ever perceiving it was going off course. The black-and-white framegrabber eliminated the use of color. Worse yet, the update rate of the framegrabber was almost 150 ms; any vision processing algorithm would have to update very quickly or else the robot would be moving faster than it could sense and react. The reflections from uneven grass reduced the standard range of the sonar from 25.5 feet to about 10 feet.

Step 3: Describe the Environment. This step is critical for two reasons. First, it is a key factor in determining the situatedness of the robot. Second, it identifies perceptual opportunities for the behaviors: how a perceptual event will instantiate a new behavior and how the perceptual schema for a behavior will function. Recall from chapter 4 that the Reactive Paradigm favors direct perception or affordance-based perception because it has a rapid execution time and involves no reasoning or memory.

The course was laid out on a grassy field with gentle slopes. The course consisted of a 10-foot wide lane marked in US Department of Transportation white paint and was roughly kidney-shaped (see figure 19.2). The exact length of the course and layout of obstacles on the course were not known until the day of the competition, and teams were not permitted to measure the course or run trials on it. Obstacles were all stationary and consisted of bales of hay wrapped in either white or red plastic. The bales were approximately 2 feet by 4 feet and never extended more than 3 feet into the lane. The sonar

was able to reliably detect the plastic covered bales at most angles of approach at 8 feet away. The vehicles were scheduled to run between 9AM and 5PM on May 22, regardless of weather or cloud cover. In addition to the visual challenges of changing lighting due to clouds, the bales introduced shadows on the white lines between 9AM to 11AM and 3PM to 5PM. The sand pit was only 4 feet long and placed on a straight segment of the course.

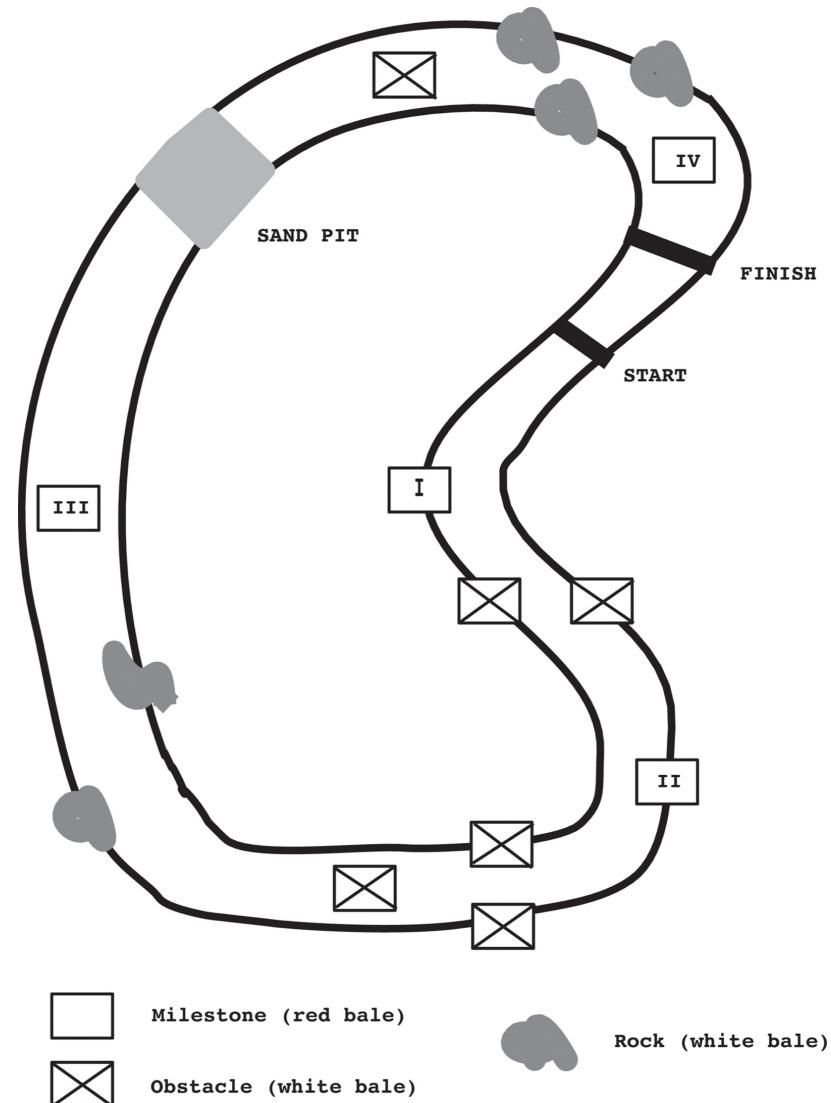


Figure 19.2 The course for the 1994 Ground Robotics Competition.

The analysis of the environment offered a simplification of the task. The placing of the obstacles left a 4-feet-wide open area. Since Omnidbot was only 3 feet wide, the course could be treated as having no obstacles if the robot could stay in the center of the lane with a 0.5 ft tolerance. This relationship

between the size of the robot and the scale of the environment eliminated the need for an avoid obstacle behavior.

The analysis of the environment also identified an affordance for controlling the robot. The only object of interest to the robot was the white line, which should show as a high contrast to the green (dark gray) grass in a camera. But the exact lighting value of the white line changed with the weather. However, upon further consideration, the team realized that if the camera was pointed directly at one line, instead of trying to see both lines, the majority of the brightest points in the image would belong to the line. This is a reduction in the signal-to-noise ratio because more of the image has the line in it. Some of the bright points would be due to reflections, but these were assumed to be randomly distributed. Therefore, if the robot tried to keep the centroid of the white points in the center of the image, it would stay in the center of the lane.

Step 4: Describe how the robot should act in response to its environment. The purpose of this step is to identify the set of one or more candidate primitive behaviors; these candidates will be refined or eliminated later. As the designer describes how the robot should act, behaviors usually become apparent. It should be emphasized that the point of this step is to concentrate on what the robot should do, not how it will do it, although often the designer sees both the what and the how at the same time.

In the case of the CSM entry, only one behavior was initially proposed: follow-line. The perceptual schema would use the white line to compute the difference between where the centroid of the white line was versus where it should be, while the motor schema would convert that difference into a command to the steering motor.

BEHAVIOR TABLE

In terms of expressing the behaviors for a task, it is often advantageous to construct a *behavior table* as one way of getting all the behaviors on a single sheet of paper. The releaser for each behavior is helpful for confirming that the behaviors will operate correctly without conflict (Remember, accidentally programming the robotic equivalent of male sticklebacks from chapter 7 is undesirable). It is often useful for the designer to classify the motor schema and the percept. For example, consider what happens if an implementation has a purely reflexive move-to-goal motor schema and an avoid-obstacle behavior. What happens if the avoid-obstacle behavior causes the robot to lose perception of the goal? Oops. The perceptual schema returns no goal and the move-to-goal behavior is terminated! Probably the designer assumed that the behavior would be a fixed-action pattern and thereby would persist in moving toward the last known location of the goal.

As seen from the Behavior Table in table 19.1, the CSM team initially proposed only one behavior, follow-line. follow-line consisted of a motor schema, stay-on-path(centroid), which was reflexive (stimulus-response) and taxis (It oriented the robot relative to the stimulus). The perceptual schema, computecentroid (image,white), extracted an affordance of the centroid of white from the image as being the line. Only the x component, or horizontal location, of the centroid was used, c_x.

Table 19.1 Example behavior table.

Behavior Table

Releaser	Behavior	Motor Schema	Percept	Perceptual Schema
always on	follow-line()	stay-on-path(c_x)	c_x	compute-centroid(image,white)

Step 5: Refine each behavior. By this point, the designer has an overall idea of the organization of the reactive system and its desired activities and now can concentrate on the design of each individual behavior. When constructing the underlying algorithms for the motor and perceptual schemas, it is important for the designer to consider the normal range of environmental conditions the robot is expected to operate in (e.g., the steady-state case) and conditions under which the behavior will fail.

The follow-line behavior was based on the analysis that the only white objects in the environment were the white lines and the bales of hay covered in white, as described in the rules. Although this was a good assumption, it led to a humorous event during the second heat of the competition. As the robot was following the white line down the course, one of the judges stepped into the view of the camera. Unfortunately, the judge was wearing white shoes, and Omnibot turned in a direction roughly in-between the shoes and the line. The CSM team captain, Floyd Henning, realized what was happening and shouted at the judge to move. The warning came too late; the robot's front wheels had already crossed over the line. Its camera was now facing outside the line, and there was no chance of recovery. Suddenly, right before the leftmost rear wheel was about to leave the boundary, Omnibot straightened up and began going parallel to the line! The path turned to the right, and Omnibot crossed back into the path and re-acquired the line. The robot eventually went out of bounds on a hairpin turn further down the course. The crowd went wild, while the CSM team exchanged confused looks.

What happened to make Omnibot drive back in bounds? The perceptual schema was using the 20% brightest pixels in the image for computing the centroid. When Omnibot wandered onto the grass, it went straight because the reflection on the grass was largely random and the positions cancelled out, leaving the centroid always in the center of the image. The groundskeepers had cut the grass only in areas where the path was. Next to the path was a parallel swatch of uncut grass loaded with dandelion seed puffs. The row of white puffs acted as a white line, and, once the dandelions were in viewing range, Omnibot obligingly corrected its course to be parallel to them. It was sheer luck that the path curved so that when the dandelions ran out, Omnibot continued straight and intersected with the path. While Omnibot was not programmed to react to shoes and dandelions, it did react correctly considering its ecological niche.

Step 6: Test each behavior independently. As with any software engineering project, modules or behaviors are tested individually. Ideally, testing occurs in simulation prior to testing modules on the robot operating in its environment. Many commercially available robots come with impressive simulators or have simulators embedded in the Robot Operation System (ROS) package. However, it is important to remember that simulators often model only the mechanics of the robot, not perceptual abilities. Simulation is useful for confirming that the motor schema code is correct, but often the only way to verify the perceptual schema is to try it in the real world.

Step 7: Test with other behaviors. The final step of designing and implementing a reactive system is to perform integration testing; integration testing is the stage at which the behaviors are combined. Integration testing also includes testing the behaviors in the actual environment.

Although the follow-line behavior worked well when tested with white lines, it did not perform well when tested with white lines and obstacles. The obstacles, shiny white bales of hay sitting near the line, were often brighter than the grass. (Note that bales wrapped in red were used only to mark the

(starting area and would not be visible to the robot.) Therefore the perceptual schema for follow-line included pixels belonging to the bale in computing the centroid. Invariably the robot became fixated on the bale and followed its perimeter rather than the line. The bales were referred to as “visual distractions.”

Fortunately, the bales were relatively small. If the robot could “close its eyes” for about two seconds and just drive in a straight line, it would stay mostly on course and be able to re-acquire the line. This was called the move-ahead behavior. The behavior used the direction of the robot (steering angle, dir) to produce a uniform potential field, even though there was no vector summation. The issue became how the robot would know when to ignore the vision input and deploy move-ahead.

The solution to the issue of when to ignore the input from the camera was to use the sonar as a releaser for move-ahead. The sonar was pointed at the line, and whenever it returned a range reading, move-ahead took over for two seconds. Due to the difficulties in working with DOS, the CSM entry had to use a fixed schedule for all processes. It was easier and more reliable if every process ran each update cycle even if the results were discarded. As a result the sonar releaser for move-ahead essentially inhibited follow-line, while the lack of a sonar releaser inhibited move-ahead. Both behaviors ran all the time, but only one had any influence on what the robot did. [Figure 19.3](#) shows this inhibition, while the new behavioral table is shown in [table 19.2](#).

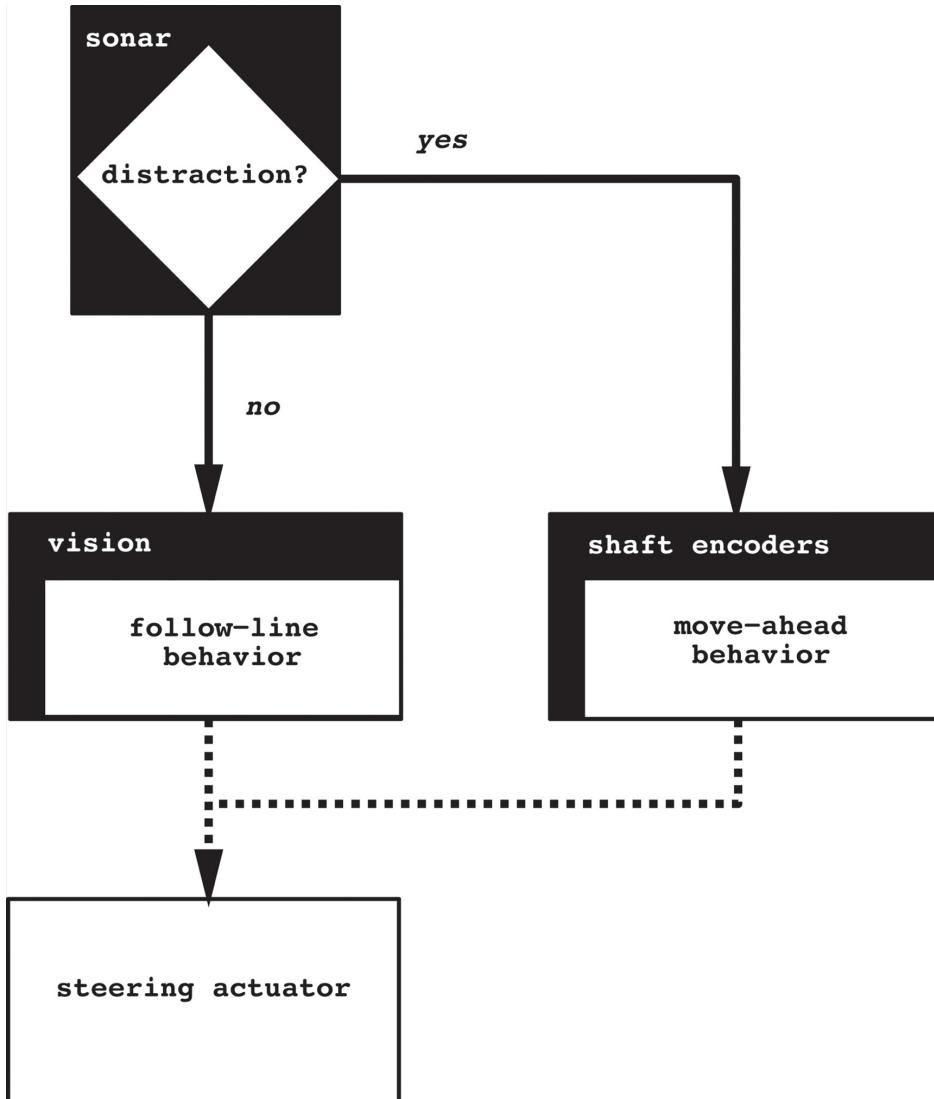


Figure 19.3 The behavioral layout of the CSM entry in the 1994 Unmanned Ground Vehicle Competition.

Table 19.2 New behavioral table for the CSM entry.

Releaser	Inhibited by	Behavior	Motor Schema	Percept	Perceptual Schema
always on	near=read_sonar()	follow_line()	stay-on-path(c_x)	c_x	compute_centroid(image,white)
always on	far=read_sonar()	move_ahead(dir)	uniform(dir)	dir	dead_reckon(shift-encoders)

The final version of the design worked well enough for the CSM team to take first place. The robot went around the track until it was about 10 yards from the finish line and encountered a shallow sand pit that was intended to test the platform's traction. The sand pit was of some concern since sand is a light color, and might be interpreted as part of the line. Because the sand was at ground level, the range reading could not be used as an inhibitor. In the end, the team decided that since the sand pit was only

half the length of a bale, it would not have enough effect on the robot to be worth changing the delicate schedule of existing processes.

The team was correct in that the sand pit did turn out to be too small to be a significant visual distraction. However, they forgot about the issue of traction. In order to get more traction, the team slipped real tires over the slick plastic wheels but forgot to attach the tires to the wheels. Once in the sand, the robot spun its wheels inside the tires. After the time limit was up, the team was permitted to nudge the robot along (done with a frustrated kick by the lead programmer) to see if it would have completed the entire course. Indeed it did. No other robot made it as far as the sand pit.

It is clear that a reactive system was sufficient for this application. The use of primitive reactive behaviors was extremely computationally inexpensive, allowing the robot to update the actuators almost at the update rate of the vision framegrabber.

The CSM team evolved a robot which fit its narrow ecological niche. The behaviors would not work for a similar domain, such as following sidewalks, or even a path of white lines with an intersection because the niche did not include those attributes. Yet the ecological design process is not limited to narrow niches. A similar design process could be used to develop a robot that met explicit requirements for a broader confluence of environments and tasks with additional robustness. That robot would likely require deliberative functionality.

19.4 Taxonomies and Metrics versus System Design

A common mistake in design is to confuse the taxonomies and metrics for comparing autonomous systems presented in chapter 3 with design guidelines and roadmaps. In particular, the taxonomy of delegation between the human and machine, originated by Sheridan for NASA¹⁹¹ and used as the source for the level of autonomy (LOA) styles of architectures, is often treated as a programming roadmap. Focusing on a taxonomy directs designers to an evolutionary roadmap approach: first, program robots to handle shared control, then program decision support, and so forth, and eventually sufficient capabilities will be added so that the code base for fully autonomous robots will exist. The design philosophy promoted by this book starts with identifying the desired capabilities for a particular robot independent of the amount of deliberative artificial intelligence needed. Then the designer determines which agent—robot, human, or software—is responsible. For example, the ecological niche for one robot may be to provide decision support for some tasks yet the same robot might require shared control for other tasks.

Taxonomies and visualizing the organization of software as layers are often treated as developmental roadmaps based on the idea that we should first program systems with the “lowest” intelligent capabilities and then add programming to enable the robot to perform the next set of functions or capabilities. There are two problems with this approach. First, it undermines the value of systems design. For example, a small house will need the same core functional spaces of a bathroom, kitchen, bedrooms, and living space as a mansion; a house is generally not a house if it contains only a bathroom or only a kitchen. Taxonomies and layers tend to encourage a “first build houses with just bedrooms, then build houses with bedrooms and bathrooms, and then building houses with bedrooms, bathrooms, and kitchens” approach rather than a “build a small house with a bedroom, bathroom, and kitchen and add a second story or wing later” approach. The house has to provide a certain amount of

functionality from the very beginning to be valuable. Second, the layered approach inhibits extensibility. If a person was building a house incrementally over years and living there as they built it, the first increment of the house would normally have some sort of provision for a kitchen, bathroom, and some sort of living space that can also be used as bedroom, but the house would be designed to facilitate future additions. For example, a larger foundation might be built or certain walls or portions of the roof constructed in a such a way as to make it easier to add on to the house. Adding on to a house that was not designed for future additions can be expensive and lead to an ugly house with an awkward use of space.

Recent discussions in the robotics communities have led to attempts to create classification schemes that would compare the autonomy of a robot system to another. One example of such a classification scheme is the US National Institute of Standards and Technology's *Autonomy Levels for Unmanned Systems (ALFUS)*.⁹⁵ Metrics for comparing or estimating how autonomous a system is can be misleading because the amount of work it takes to build a house is not the measure of whether a structure is a house. Consider that a building with a lovely kitchen but without a place to sleep may take as much time to build as a house with a bedroom and a minimal kitchen, but a lovely kitchen is not a house. A house in the 1800s probably required more physical effort to build than a current suburban tract home which has been optimized for ease of construction. Yet both are functional equivalents. A robot system should have a particular autonomous capability because the goals of the system require that capability, not because that particular capability was easier to program than another capability.

The ecological approach encourages a designer to think outside of the simplistic visualization of layers and about the influences on the design of specific core functions. A three-bedroom house built along a hillside in earthquake-prone California will be built with a different type of foundation and construction materials than would a similarly sized three-bedroom house along a beach in Florida. Likewise, in a robot, the choice of platforms, sensors, and algorithms is influenced by the environmental complexity. But the California house is not intrinsically “better” than the Florida house. Returning to the small house example, while both a small house and a mansion will have the same core functional spaces, a mansion designed for a more complex lifestyle of entertaining guests may have a larger kitchen, more bedrooms and bathrooms, and additional features such as a swimming pool and home theater. Likewise the design of a robot is influenced by the mission complexity. A five-bedroom house may have a small yard because the owners do not want to spend time gardening, while a one-bedroom cottage might have a much larger yard because the owners do want to spend time gardening. Likewise, robot design is influenced by the amount of human independence desired.

19.5 Holistic Evaluation of an Intelligent Robot

Chapter 3 introduced the challenge of non-determinism in testing autonomy. The subsequent chapters offered evidence that an autonomous capability might actually be the result of multiple algorithms or data structures working together. These individual algorithms can be tested, but, at some point, the capability will be judged in the context of the holistic robot system. The question is “Did the increased intelligence work and add value to the system overall?”

This section does not discuss specific test methods because those are heavily dependent on the

algorithm, hardware, and application. Instead it offers a general framework for conducting an evaluation of an intelligent robot. It first provides a failure taxonomy to alert the designer to the types of failures to look for. Next, it describes four types of experiments or settings in which to evaluate robots. The section recommends specific categories of data to collect in order to prove hypotheses, debug problems, and opportunistically learn new things about the robot. It concludes with a case study of how collecting a large set of data helped determine a sensor problem.

19.5.1 Failure Taxonomy

Figure 19.4 shows a taxonomy of types of robot failures presented in¹⁴¹ as an expansion of an earlier version by Carlson and Murphy.³⁷ Robot failures, according to the taxonomy, have three sources.

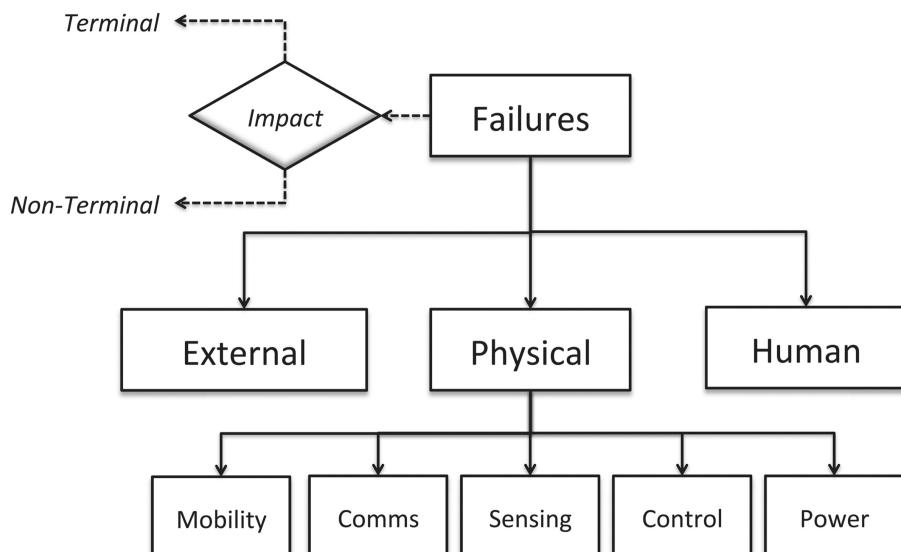


Figure 19.4 A taxonomy of robot failure types.¹⁴¹

EXTERNAL FAILURE

1. *External failure* which is beyond the direct control of the robot or operating crew. An example is a robot being destroyed in a mine explosion.

PHYSICAL FAILURES

2. *Physical failure* of the robot itself. Physical failures can be further classified by subsystem: mobility, communications, sensing, control or computation, and power. Mobility failures are rarely encountered, while power (e.g., short circuits) and communications drop-outs are more frequent.³⁸ The low rate of mobility failure and high rate of power and communications failures may not be surprising because designers often focus on perfecting mobility effectors and ignore the “secondary” systems.

HUMAN ERROR

3. *Human error*. A recent study of disaster robotics reports that more than 50% of failures in robots were due to human error.¹⁴¹ That high of a human error rate suggests that the source of the error is the designer, not the operator.

Failures can have one of two impacts. A *terminal failure* means the mission is terminated; either the mission aborted or the robot has died in place. A *non-terminal failure* means the mission continues but with degraded capabilities.

19.5.2 Four Types of Experiments

Table 19.3 shows the four types of experiments that are used by roboticists to test autonomous capabilities. Each type has a *driver*, or ultimate purpose or motivation, a venue in which the experiment occurs, and a general class of expected outcomes. In addition, the *composer* of the overall structure of the experiment may be either the roboticist or the stakeholder using the robot. The experiment may take place in realistic, also called high-fidelity, conditions. It is important to keep in mind that realistic conditions reproduce both the physical properties of the intended workspace and the operational properties of the workspace; for example, everything follows standard safety and operational procedures used for that application or by that stakeholder. Conditions range from laboratory settings, which sacrifice realism in exchange for statistical repeatability, to using the robot in a real mission.

Table 19.3 Four types of experiments, modified from Murphy.¹⁴¹

Type of Fieldwork	Driver	Venue	Composer	Outcomes
Controlled Experimentation	hypothesis or performance	Laboratory	Roboticist	statistically valid inferences
Exercise	successful demonstration	Staged World	Stakeholders and Roboticists	familiarity, favorable opinion, accelerated adoption, some feedback
Concept Experimentation	mission suitability	Staged World or Natural World	Stakeholder	gaps analysis, possible new uses, concepts of operations
Participant-Observer	authencity	Natural World	Stakeholder	case studies of actual uses and concepts of operations, failures and bottlenecks

CONTROLLED EXPERIMENTATION

In *controlled experimentation*, the objective is to test a hypothesis or quantitatively ascertain the performance of a specific capability of the robot. The experimentation occurs in a laboratory setting (either indoors or outdoors) where the roboticist explicitly composes the experiment and controls conditions to obtain statistically valid inferences. The focus is on repeatability.

EXERCISE

Following the definition in Alberts and Hayes,³ in an *exercise*, the objective is to test the system in a

more realistic environment following a scripted set of activities. An exercise is conducted in a “staged world,” where the robot’s environment is staged to be realistic, but repeatable, allowing for higher fidelity physical testing. The script is expected to have higher fidelity operational fidelity than it would in a laboratory. Exercises may be composed by either the roboticist or a stakeholder. Exercises are typically used by roboticists to try the robot in more realistic conditions or to obtain feedback from stakeholders. Exercises, such as the Top Official National Domestic Counterterrorism Exercise (TOPOFF) or the US Department of Defense Advanced Concept Technology Demonstrations, are used by stakeholders to promote familiarity with hands-on use of the technology and to encourage other stakeholders to form a favorable opinion, that will, in turn, accelerate adoption of that technology. Adherence to a script and the desire to impress participants and observers with the robot often lead to engineering the script and environment to highlight the robot’s strengths. Thus exercises are not as valuable as might be expected for evaluating the real system-level performance of a robot or gaining any human-robot interaction insight.

PARTICIPANT-OBSERVER

*Concept experimentation*³ is perhaps what most roboticists intend as the purpose of an exercise; that the outcomes of the exercise would be a gap analysis of what was missing from the system design, possible new uses would be identified, and the general concept of operations. Concept experimentation is intended to explore the suitability of a technology system for a mission, from beginning to end, which is referred to as the concept of operations. Concept of operations includes not only the mission but also the logistics of moving the robot into position, size and weight restrictions, lack of electricity for recharging batteries during a normal shift, and so forth. The scenario for the exercise would be composed by the stakeholders to offer the opportunity to insert the technology but not to change anything else about normal operations. The technologists would have to capture data without interfering with the operation of the robot or the flow of the exercise. In concept experimentation, the participants continue with the mission even if the robot fails, thus providing valuable feedback on the vulnerabilities of the system. Concept experimentation has been used in robotics and human-machine interaction since 2002, most notably by the Summer Institute series for disaster robotics.^{152;213;89}

PARTICIPANT-OBSERVER

The highest fidelity physical and operational conditions for evaluating a robot occur during actual use. In that case, the roboticist is acting as a *participant-observer* in the natural world (versus a staged world), similar to when ethnographers embed with a remote tribe. The data produced from these real-world applications are inherently less quantitative than other forms of testing and evaluation but the case studies are useful for determining how the robot will really be used in practice, what needs to be done to facilitate those uses, and identifying when the robot fails and why.

19.5.3 Data to Collect

Evaluating a robot system requires data. While collecting data is obvious, it is easy to forget to collect a complete set of data. Experiments generally report the dependent variable, independent variable, and operating conditions, robot and any other equipment used for the experiment, and the experimental method as well as the results. However, a recent analysis of 18 studies of autonomous take-off and

landing in micro-UAVs found that the only common measure reported by all the studies was the type of vehicle,⁶⁷ so it is worth reviewing what data to collect. For three of the four types of experiments, exercises, concept experimentation, and participant-observer experimentation, the book *Disaster Robotics*¹⁴¹ identifies six categories of experimental data to collect, if possible: *log of activity, context, robot's eye view, robot state, external view of the robot, and human-robot interaction* view. The book also describes the essential elements of a data collection protocol for field robotics, including data management. For the fourth type of experiment, controlled experiments, the purpose of data collection is to prove or disprove a hypothesis. The hypothesis may be very broad, often along the lines of “the autonomous capability works,” but there has to be a definition of “works” as well as data to support or refute the hypothesis and enough information about the data to assure confidence in the results. For example, the hypothesis of controlled experiments with an autonomous take-off and landing capability for a micro-UAV might be to show that the robot can reliably land itself in any weather condition, or that the robot can land itself as well as a human, or that a human pilot can take over during an aborted landing. The first hypothesis suggests that the experiment will have repeated take-offs and landings in a wide variety of weather conditions and will have metrics for measuring error. Those metrics can be the distance from the target, the distance and difference in heading, or any other measure that reflects the definition of “works.” The second hypothesis would set up a statistical comparison of human pilots and auto-pilots by having the human and robot both land in a set of near identical situations. The metric might include a paired T-test to confirm that the performance of the human and the robot were different. Both of the experiments would resemble classic laboratory experiments from chemistry and physics classes. The third hypothesis would require a very different type of experiment, following the psychological experimental design described in chapter 18.

EXPERIMENTAL DATA

The data collected for a classic experiment, following Duncan and Murphy,⁶⁷ would consist of:

- *Dependent variables, independent variables, and constants.* Independent variables are the variables that are changed or manipulated in order to produce the measurable change in the dependent variable. In the case of autonomous landing, a dependent variable might be the average error between the actual landing location and the desired location, and the independent variables might be the algorithms (e.g., autonomous, manual, autonomous with different parameters, and so forth), the different operators, the starting altitude of the descent, the landing surface, etc. Everything else should be constant. If there are aspects of the experiment that are not constant, for example, wind, temperature, time of day, cloud cover, or other *operating conditions*, they have to be measured and their impact on any results explicitly considered and discussed. These uncontrolled operating conditions could be minor or they could be confounding factors that undermine any conclusion about the hypothesis.
- *Operational data.* There may be data that are recorded that are not expected to be used for the hypothesis but may be of value later. Recording the total flight time can uncover important information. For example, if the UAV is unable to land in 2.5 minutes and an operator has to take over, this can be incorporated into the algorithm or operating procedures. Likewise recording the flight path can be of use, because a straighter or smoother path may indicate better performance or identify runs where an anomaly was occurring.

- *Equipment or apparatus.* An experiment has to record what robot, payload, operating system, and so forth were used. Experiments also report on what equipment was used to collect data in case that makes a difference. For example, using a motion-capture camera would produce a different resolution of tracking the descent movement than would a cellphone camera.
- *Experimental method.* Data should be collected about the overall process. For testing autonomous landing, an experimental method might be as follows. The UAV is started 30 times from a random altitude between 10 and 30 meters above ground level and in a random relative location and compass orientation within a 10-meter radius of the landing target. An operator would initiate the autonomous landing. The experimental run would stop when a) the UAV touches down and shuts itself off, b) when the operator aborts the landing due to weather or unforeseen behavior, or c) the run is terminated after five minutes without a successful touchdown.

SIX CATEGORIES OF DATA

The book *Disaster Robotics*¹⁴¹ identifies six categories of data to try to collect in field exercises, concept experimentation, and participant-observer situations. These are helpful for general experimentation as well as preliminary testing:

- *Log of activity.* The log of activity provides the sequence of events. The activity log can inform design refinements, such as the expected duration of a flight. For example, work on structural inspection showed that UAVs could map a side of a multistory building in 8 to 12 minutes and that the crew generally had to land and move in order to maintain safe operations while mapping the next side.¹⁷¹ No matter how many sides to a building, the maximum flight time was 12 minutes. Therefore, a UAV with longer battery life, and thus a longer flight time, was not essential. Designers could use that information to create a cheaper, lower duration UAV for the structural inspection market. An activity log can illuminate interesting advantages of robots. For example, deploying a UGV for a radiological event takes, on average, the same amount of time as a trained responder to suit up but the UGV can get down range faster and stay longer.
- *Context.* The context can help explain anomalous behavior as well as why a robot sometimes performs well and why it sometimes does not. In one case, a tracked UGV, developed for pipe and sewer inspection, was used to search a residential home crushed by a mudslide.¹³⁹ The tracks came off the UGV and the mission had to be aborted. The context of the residential home, in particular, the context that the house had shag carpeting, was essential to understanding why the robot failed. With sensitive human-robot interaction situations, the context of how much sleep the team has, the overall stress and pressure they are feeling, and so forth, provides a psychological context.
- *Robot's eye view, robot state, and external view of the robot.* These data sets are valuable for debugging and error analysis. For example, if the robot begins to oscillate slightly at time T before a crash, a video showing the oscillation serves two purposes. It is both a clue as to what might have gone wrong and it also gives a timestamp to start looking through the robot state logs (What was it doing?) and robot's eye view (What was it seeing?). The external view can be obtained from a helmet camera of a safety observer or a dedicated tracking camera for nearby operations. It may be worth having a chase plane or second robot following the first robot.
- *Human-robot interaction view.* Independent of the camera recording the external view of the robot,

it can be helpful to have a separate recording capturing what the human team members are saying and doing. This can be helpful for debugging or determining the cause of a crash, especially for identifying user interface and interaction bottlenecks. An ethnographer serving as an observer and note-taker is also helpful because s/he can identify human-robot interaction problems and possible causes that can be hard to extract from a video.

19.6 Case Study: Concept Experimentation

The value of concept experimentation and data collection is illustrated by the following case where an analyst discovered that a system performance problem, initially ascribed to human error, was actually due to unanticipated sensor behavior. The discovery of the surprising sensor behavior was a result of the high fidelity of concept experimentation as a method, the use of an ethnographer who noticed an anomaly in the operator's behavior, and the comprehensive data collection which allowed a post hoc analysis.

A 2013 concept experiment at Disaster City® simulated a response to earthquake damage to the nuclear medicine wing of a major hospital. A partially collapsed multi-story building, called Prop 133, at Disaster City simulated the hospital and a cesium radiological source was hidden somewhere in the building. See [figures 19.5a and b](#). Hazardous materials (HazMat) fire rescue teams, trained in chemical, biological, radiological, nuclear, and explosive (CBRNE) events, had to respond to the earthquake, search the area and hospital for radiation and people in distress, and localize any radiological material and remove it. The concept experiment introduced a ground robot, a UAV, and visualization tools, each of which was hypothesized to decrease the data-to-decision time. The hypotheses were supported, but the ground robot did not perform as expected.

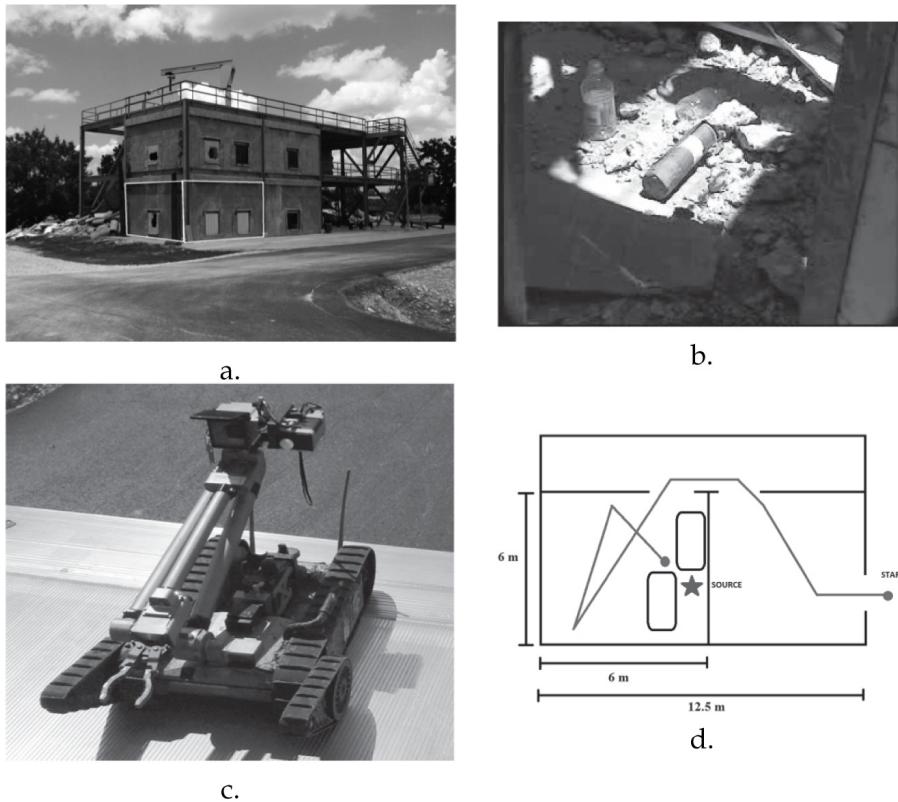


Figure 19.5 Nuclear forensic localization concept experimentation: a.) the Prop 133 site at Disaster City® serving as the simulated hospital; the room containing the radioactive source is highlighted, b.) the radioactive source in a bright blue container on floor, c.) an iRobot Packbot 510, and d.) robot path relative to the source.

The ground robot deployment was as follows: An iRobot Packbot 510 with a Canberra Radiac sensor, a small sensor commonly used by HazMat teams, shown in [figure 19.5c](#), was unpacked and configured at the same time as the traditional HazMat teams starting donning protective clothing. The set up of the robot was not significantly faster than the manual teams, so both the robot and the human team began searching for the radioactive source at the same time. But once the robot started moving down range, it provided continuous radiological readings, which the manual team could not do. The incident commander determined that the radiation was not as bad as feared and started moving personnel and equipment closer to the site. Meanwhile the robot reached the collapsed building faster than the responders could walk down range, surveyed the exterior of the building, and then entered the building to search for the radiological material. Although the rooms were small, it took the operator and HazMat specialist nearly 30 minutes to find the bright blue cylinder containing the cesium on the floor between two desks.

During the after action meeting following the exercise, the unexpectedly long localization time was initially ascribed to operator error but the ethnographer stationed with the robot observed and reported that the operator had constantly complained that the radiological readings were inconsistent and seemed to be steadier and more readable when the robot was going backwards to move around

obstacles.

A student, Dexter Duckworth, on the research team reviewed the comprehensive data collected and discovered that the problem was not due to human error. The student was able to reconstruct the path of the robot relative to the source ([figure 19.5d](#).) and associate the radiation readings with the location along the path. The data showed that the readings from the robot were indeed higher when the robot was facing away from the source. This was very surprising because the Radiac sensor was omnidirectional and had passed all calibration tests and is the de facto standard for radiological sensors in the United States.

The team contacted the Japanese Atomic Energy Agency (JAEA), which was participating in the use of robots for the Fukushima Daiichi nuclear accident and subsequent decommissioning, to see if the agency had witnessed this strange sensor behavior, as the most commonly used robot at Fukushima was the iRobot Packbot 510. The JAEA replied that workers had never tried using a Packbot for radioactive source localization, they used it only for general radiological surveys. However, the agency had an explanation for what was happening. The Radiac sensor was designed to be carried by a human, usually clipped to a belt. At that height, the sensor registers radiation from any angle as the human body provides relatively little absorption. In the Disaster City scenario, the Radiac was mounted in the front of the Packbot payload bay with one side against the heavy metal frame. The metal frame was blocking radiation, turning the Radiac into a directional sensor aimed at the back of the robot. The JAEA and HazMat teams were pleased to learn about this behavior so that they could be aware of it should they use the robots for radiological source localization.

19.7 Summary

This chapter has concentrated on answering *How do I design an intelligent robot?* by providing design principles for a robot system overall. Despite the prevalence of attempting to quantify how much more autonomous one system is compared to another using “layers” or metrics, these comparisons tend to distract from the ecological approach to design. The ecological approach expects the designer to consider the interdependencies between the robot, environment, and task. Explicitly considering those interdependencies can lead the designer to discovering affordances and behaviors that can simplify programming and increase robust performance.

Regardless of the proactive design effort, the design may be imperfect or robots may fail for other reasons. Failures can arise from one of the five hardware-oriented subsystems of the robot itself, which are called physical failures. They can arise from human error, which may account for 50% of failures. A robot can fail due to an event or condition in the external world, such as being crushed while exploring a collapsed building.

Testing that the system will act intelligently can be hard because the robot is usually working in an open world. As noted in chapter 3, the algorithms may be non-deterministic and certainly the open world amplifies the spectrum of unexpected events which may be too numerous or hard to model explicitly. Clearly, evaluating a robot system is more complex than testing an algorithm. Depending on the goals or driver for the evaluation, the designer may want to choose between a formal controlled laboratory experiment, an exercise, a concept experiment, or inserting the robot system into the natural world. It is particularly important not to confuse an exercise with concept experimentation, as exercises

guarantee that objectives will be successfully met but concept experimentation probes for system-level potential problems and encourages failures. Regardless of the type of experiment used to evaluate the robot system, the designer should try to collect a wide variety of general data: the log of activity, context, robot's eye view, robot state, external view of the robot, and human-robot interaction view. It is not always possible to collect all types of data, but more is better.

Now that the reader is familiar with how to design an autonomous capability and robot system, the next, and final, chapter will turn to ethics. In some regard, this chapter sets the stage for ethics because a robot designer is bound by professional ethics to use the best principles in designing autonomous capabilities and should be aware of the larger ethical issues in using robots.

19.8 Exercises

Exercise 19.1

Consider writing the autonomous navigational capability for a driverless taxi or bus where the user enters the vehicle, states the desired destination, and then the car drives to that location. Answer the following five design questions:

- a. What functions does the robot need to provide?
- b. Which planning horizon do each of the functions require?
- c. How fast do the algorithms have to provide updates for control of the car?
- d. What type of world model does the robot need?
- e. Where does the human come in?

Exercise 19.2

Give examples of robot failures in a driverless car that could arise from each of the three sources of robot failures: external, physical, and human.

Exercise 19.3

List the four types of experimentation and give an example of how each might be used in which step of the design process for the autonomous navigational capability for a driverless taxi or bus.

Exercise 19.4

What type of experiment would you use to generate quantitative measures in controlled conditions so as to obtain statistically valid inferences about a robot?

Exercise 19.5

What type of experiment would you use to try out a robot system in realistic conditions, including users, following a script?

Exercise 19.6

What type of experiment would you use to try out a robot system in realistic conditions, including users, and collect data on systems-level issues?

Exercise 19.7

What type of experiment would you use to try to collect data about a robot system while it is being used by real users for real jobs?

Exercise 19.8

List the six categories of data to try to collect during system evaluation of any robot. Consider the design of an autonomous UAV that would drop off relief supplies to trapped hikers in the woods. What data would you capture in

order to prove that the system worked reliably and safely?

Exercise 19.9

Which one of these is not usually collected for an experiment?

- a. Dependent variables, independent variables, and constants
- b. Equipment or apparatus
- c. Experimental method
- d. Narrative summary of the difficulties and emotional stress in collecting the data
- e. Operational data.

Exercise 19.10

Consider a robot with a new vision algorithm that should improve its ability to detect obstacles. Why would it be useful to record more than just the duration of the run? What other attributes could be measured?

End Notes

For the Roboticist's bookshelf.

Alberts, David S., Hayes, Richard E, *Code of Best Practice for Experimentation* ³ is a great overview of experimentation versus exercises and introduces concept experimentation.

AI for Mobile Robots: Case Studies of Successful Systems ¹⁰⁸ describes older, ambitious attempts to create fully autonomous systems. Although the hardware is quite old, the case studies provide a comprehensive look at the design decisions involved in creating each robot. The book is also cited by Michael Crichton in his bestseller *Prey*.

Science fiction sometimes gets it right...

A small, Dante-like robot called SpiderLegs appears in the 1997 movie *Dante's Peak*. It is used by a team of vulcanologists, led by Pierce Brosnan, to explore a Mount St. Helens-like volcano that is threatening a lovely town with a coffee shop owned by newly single Linda Hamilton. The robot is continually breaking, forcing the team of graduate students to drink more coffee and to come up with workarounds. Both the breakage and the heavy reliance on strong coffee are accurate reflections of field robotics.

20

Ethics

Chapter Objectives:

- Describe the difference between *operational morality*, *functional morality*, and *full moral agency* and state the designer's responsibility of for each.
- List the four types of *ethical agents* (ethical impact, implicit, explicit, full) and give an example of each.
- Describe at least one problem with each of Asimov's *Three Laws of Robotics* as a guideline for developing moral robots.

20.1 Overview

The highly visible use of drones by the U.S. military has increased societal discomfort with robots and accelerated the growth of robot ethics. The public poses questions such as: *Isn't it unethical to weaponize robots?* That issue is hotly debated in artificial intelligence circles. Another common question from the public, sometimes a statement, is: *If programmers apply Asimov's Three Laws, won't that be enough to protect us from injury, death, or an uprising?* That question taps into Asimov's perennially popular *I, Robot* science fiction stories, but ignores that the stories were about how the three laws created conflicts and unintended consequences. A third question posed by the public is: *Isn't it unethical to create robots and treat them like slaves?* These questions are becoming so prevalent that the European Union recently conducted a multi-year RoboLaw study.^{177;172} The study analyzed whether robots were exempt from regulations that apply to products and people and thus required special regulations, and if so, what those regulations would be. The answers to the three questions have ramifications for the design of robot systems. A fourth topic in ethics has emerged as to the societal impact of job dislocation from increased use of robots but since that topic is not explicitly a robot design problem, it is beyond the scope of this book:

Designers often get caught up in speculation about future robot capabilities (“One day, when robots have full autonomy...”) and forget their professional obligations to design safe systems for any autonomous capability that are consistent with good human factors and engineering principles. For example, a recent crash of the Tesla car apparently occurred in a situation where the navigational

autonomy failed and the human did not take over;¹⁶² this appears to be an instance of the human out-of-the-loop control problem described in chapter 18. In that case, the courts did not hold the manufacturer liable for designing a system that violated well-known human factors principles, but it certainly raises the issue of professional ethics, which is the purview of operational morality.

Ethical considerations of artificial intelligence and robotics are motivated by at least four categories of applications where the use involves our perceptions of morality.²¹⁵ One category is an application where the robots are designed to “do good” and thus society is obligated to use them just as health professionals were obligated to use penicillin and heart-lung machines. Robot camel jockeys that replace children and rescue robots are two examples of robots intended to “do good.” Another category addresses robots that are used because they are morally superior agents. The robot border guards in South Korea watching the demilitarized zone cannot be bribed and would not hesitate to react to intruders. Thus they are more reliable than human guards and morally superior. A third category covers applications where we expect robots to treat us appropriately. This category is especially apparent in the debate over weaponized robots, but it also includes situations where an autonomous car might decide to take an action that injures you but saves many others. Another example of appropriate treatment is deception: Is it acceptable for a robot to lie in order to get us to behave in our ultimate best interest? The fourth category is the inverse of the third; it contains the applications where we should treat robots appropriately. For example, at what point does a robot engaged in healthcare and service go from being an intelligent assistant to being a slave?

This chapter will attempt to give the reader a framework for thinking about the ethics of design. It first defines ethics and professional ethics. It presents Moor’s four categories of ethical agents.¹³³ The categorization suggests that most autonomous robots are unlikely to be explicitly concerned with ethics or will be acting as fully ethical agents. The next four sections cover morality and the moral hierarchy posed by Wallach and Allen:²¹⁵ operational morality, functional morality, and full moral agency. It covers the current state of the art in the design of functional and full moral agents. The chapter then examines Asimov’s Three Laws and why they are not sufficient for good design. Next, the chapter reframes ethics in terms of the types of artificial intelligence needed to support ethical designs or ethical agents.

20.2 Types of Ethics

PROFESSIONAL ETHICS

Ethics in robotics has two meanings. In the philosophical sense, “Ethics is a branch of philosophy concerned with how moral values should be determined, how moral outcome can be achieved in specific situations, how moral capacity or moral agency develops and what its nature is, and what moral values people act on and abide by.”⁷³ The second meaning of ethics in robotics is *professional ethics*. Professional ethics “concerns the moral issues that arise because of the specialist knowledge that professionals attain, and how the use of this knowledge should be governed when providing a service to the public.”⁴¹

Robot designers need to be aware of both types of ethics. While the public discussions focus primarily on philosophical aspects of robots, designers are responsible for following professional ethics because of liability issues. This can be complicated because there is no “intelligent robotics

profession,” and thus professional ethics specific to robotics have not been clearly established. However, AI robotics builds on professions which do have ethics such as engineering, computer science, psychology, plus the application domains of medicine and nursing, which also have professional ethics. Professional organizations promote the ethics for their fields, and programs granting degrees in that profession offer classes or modules on ethics. For example, the Association for Computing Machinery has a 24-part code of ethics and professional conduct.¹ For example, Part 2.5 of the code states “Give comprehensive and thorough evaluations of computer systems and their impacts, including analysis of possible risks.” The ACM code makes it clear that saying a robot is autonomous and therefore the designer is not responsible for any deviations is not professional. Parts 3.4 and 3.5, “Ensure that users and those who will be affected by a system have their needs clearly articulated during the assessment and design of requirements; later the system must be validated to meet requirements” and “Articulate and support policies that protect the dignity of users and others affected by a computing system,” are particularly relevant. A true computing professional would incorporate human-robot interaction into the design of a robot from the very beginning.

20.3 Categorizations of Ethical Agents

Pragmatically, a designer has to determine when robots have to be ethical and to what degree do they have to be programmed to be ethical. Both Moor¹³³ and Wallace and Allen²¹⁵ think of a robot as falling somewhere on a spectrum of ethical considerations. The spectrum reflects the discussions of operational autonomy in chapter 4, where a robot may have well bounded autonomous capabilities or may be permitted to reason and relax constraints on itself.

20.3.1 Moor’s Four Categories

Moor poses four categories of agents in relationship to ethics. These categories are helpful to a designer in framing the ethical expectations of a robot and discussions of morality.

Moor’s four categories are:

ETHICAL IMPACT AGENTS

- *Ethical impact agents.* These are robots that are designed for applications that support, what Moor calls, ethical norms. Search and rescue robots are intended to help with activities that have ethical consequences: speeding life-saving responses and economic recovery are desirable and worthy. Note that the robot is not expected to reason about ethics; it is an ethical robot simply because of the application.

IMPLICIT ETHICAL AGENTS

- *Implicit ethical agents.* These are robots that are not programmed to consider ethics, but their programming must be designed to minimize negative ethical effects. Moor gives an example of auto-pilots in airplanes. Originally auto-pilots returned control to the pilots at the moment it reached the limits of its operation. This abrupt change sometimes caught pilots by surprise and led to crashes. However, an auto-pilot can sense when it is approaching its limits. Thus, because lives are at stake, the auto-pilot must be programmed to give pilots a warning that they may need to prepare

to take over in the near future. Another example is the implicit expectation that a hospital drug-delivery robot will not hit patients walking on crutches,¹⁵⁵ even though writing software to detect thin profile surfaces such as crutches is harder than detecting people and walls.

EXPLICIT ETHICAL AGENTS

- *Explicit ethical agents.* These are robots that could reason about ethical consequences of actions. Returning to the examples in the introduction, an autonomous car could conceivably be programmed to sacrifice its passengers in order to save a larger number of passengers in other cars. It seems within reach to have autonomous cars that could share information and, in milliseconds, project likelihood of death and injury.

FULL ETHICAL AGENTS

- *Full ethical agents.* These are robots that can make, execute, and justify ethical decisions in contrast to explicit ethical agents that are limited to specific situations and ranges of actions. As Moor notes, the design of full ethical agents is the subject of much of the discussion about robot ethics even though it is not clear that a robot with the requisite amount of intelligence and true political autonomy can, or would, be built.

The four categories relate not only to the definition of ethics but also to definitions of operational autonomy. The implicit ethical agent category reinforces the notion of professional ethics. An intelligent robot has to be safe and has to incorporate the best practices and highest standards of the robotics profession. An explicit ethical agent makes ethical decisions but within explicit boundaries. It executes its existing programming about the ethics while a full ethical agent can generate new programming. The difference between an explicit and a full ethical agent can be expressed using the levels of initiative terminology from chapter 4. In that terminology, a full ethical agent has constraint autonomy because it is allowed the initiative to change the rules or boundaries of its operation with regards to ethics, while an explicit ethical agent has only systems-state autonomy.

20.3.2 Categories of Morality

Wendall and Allen²¹⁵ pose a similar but different taxonomy than Moor. Their taxonomy reflects the transition from a robot as a device or tool to a fully autonomous agent in terms of the morality it evinces.

OPERATIONAL MORALITY

- *operational morality*, where the accountability and responsibility of the moral function of the robot is within the control of the tools designers and users. Accountability and responsibility are the equivalent of professional ethics and emphasizes autonomous capabilities always, in some sense, involve morals. Even though a robot is non-deterministic, that does not necessarily obviate professional ethics and liability.

FUNCTIONAL MORALITY

- *functional morality*, where the robot has autonomous capabilities (e.g., autopilot) or is working in applications with ethical sensitivity (e.g., medical decision-making aids) still falls within

professional ethics because autonomy is a function of the design. For example, if a car's autonomous driving capability fails and a person is killed, the designer may be held liable.

FULL MORAL AGENCY

- *full moral agency*, where the robot is trustworthy and has highly sophisticated deliberative capabilities that include decision-making in ethical situations.

20.4 Programming Ethics

A robot with functional or full moral agency will have to be programmed to have ethics. Exactly how this would be implemented is unclear. Philosophers have speculated about how such programming might be accomplished by reviewing how they believe humans acquire ethics. Roboticists have begun looking at more realistic and practical computing tools, but there is no consensus or emergent trend.

20.4.1 Approaches from Philosophy

Philosophers have discussed how they believe people acquire ethics with the assumption that robots will duplicate the process.

CONSEQUENTIALISM

DEONTOLOGY

VIRTUE ETHICS

There are three top-down approaches: *consequentialism*, where the robot reasons about the consequences of options; *deontology*, where the robot reasons about options in terms of its obligations, duties, and rights; and *virtue ethics*, where the robot develops good character so that it can react in the most moral way.²¹⁵ These three approaches may capture how people learn to behave ethically, but it is not clear how to implement this algorithmically. All three require advances in representation and reasoning. Even though virtue ethics may sound the easiest because it is producing behaviors rather than conducting deliberation, it implies that analogical reasoning is used to produce those behaviors; analogical reasoning is still beyond the scope of practical AI.

An alternative to top-down methods are bottom-up methods, where the robot learns to be good. This is too generic to be of use for designers. Recall from chapter 16 that there are many types of learning and that learning requires specifying what is being learned or at least specifying the input for learning new concepts.

20.4.2 Approaches from Robotics

Existing robots typically fall into ethical impact or implicit ethical agency. Current robots have not been programmed with the deliberative capabilities needed for more sophisticated explicit ethical responses. This does not mean that one day an application will call for a robot to act with explicit or full ethical agency. One example is Arkin's focus on robots as potentially able to conduct military operations in a way that is more moral than the way humans sometimes act.¹³ Humans are known for misusing their full moral agency to commit war crimes, while robots, as explicit moral agents, would

not exceed the boundaries of the rules of engagement. The robots would likely employ deliberative algorithms such as an *ethical governor*.¹³

20.5 Asimov's Three Laws of Robotics

Isaac Asimov's famous *I, Robot*¹⁴ series of short stories were set in a future where robots were governed by three rules, the first, that a robot could do no harm to a person. These rules are frequently cited as the ultimate goal in robots that interact with people, ignoring the fact that the stories were about how the rules had built-in conflicts and sufficient vagueness so as to lead to amusing and unintended consequences. The rules were a literary device deliberately designed to be flawed to support story lines. Since the rules are firmly established as the de facto gold standard of functional morality for robots, it is important to understand the rules.

THREE LAWS OF ROBOTICS

The three laws taken verbatim from¹⁴ are:

1. A robot may not injure a human being or, through inaction, allow a human being to come to harm.
2. A robot must obey orders given it by human beings, except where such orders would conflict with the First Law.
3. A robot must protect its own existence as long as such protection does not conflict with the First or Second Law.

20.5.1 Problems with the Three Laws

The three laws collectively and individually are problematic for the ethical design of robots. Murphy and Woods¹⁴⁵ list and discuss the flaws; the gist of their analysis is summarized below.

Collectively, the three laws assume that all robots have at least functional morality and possibly are even full moral agents. However, robots may not have sufficient agency and cognition to make moral decisions that have ethical impact. The rules neglect operational morality and do not consider the liability of inventors and manufacturers for design decisions.

The first law specifies the robot as the responsible safety agent, but, in reality, the robot is a product. As a product, designers and manufacturers are liable for safety and must be able to prove that they have been reasonable and prudent.

The second law specifies that robots have to obey humans without constraints on which humans and under what circumstances. In reality, people work in explicit hierarchies, and not everyone is in charge of the robot and its task. Imagine a terrorist hijacking a robot to create economic havoc. Clearly some people's commands would have a higher priority than other people's commands.

The third law states that the robot should protect itself. This law implies that the robot has sufficient intelligent capabilities to project and monitor for unsafe conditions. In reality, humans are expected to take over when robot cannot control itself in order to save the robot from a crash (or to save people).

20.5.2 The Three Laws of Responsible Robotics

THREE LAWS OF RESPONSIBLE ROBOTICS

Murphy and Woods¹⁴⁵ go further and propose a rewrite of Asimov's Three Laws, which they called the three laws of responsible robotics:

1. A human may not deploy a robot without the system meeting legal safety standards and highest professional ethics.
2. A robot must respond to humans as appropriate for their roles.
3. A robot must be endowed with sufficient situated autonomy to protect its own existence as long as such protection provides smooth transfer of control which does not conflict with the First and Second Laws.

The alternative first law would introduce operational morality as the fundamental “base case” of deploying a robot and would shift the responsibility for moral actions to the designer and operator. This alternative first law would apply to weaponization of robots and the limitations of their current capabilities. Note that alternative first law requires the robotics community to be sensitive to regulations, and that building something just because you can does not obviate ethical responsibility.

The alternative second law acknowledges non-hierarchical relationships and the need for security and command hierarchies. It also allows for cases when the robot is smarter than a human and, regardless of whether the human wants control, the robot will perform better (e.g., fly by wire). Another aspect of the second law is that the mode of response does not have to be natural language. A hospital drug-delivery robot intercepted by a visitor may need to give a waggle “no” to indicate that it cannot take orders or answer queries.

The alternative third law counters the assumption that humans can fix any problem, anytime, which research into the human out-of-the-loop control problem, discussed in chapter 18, has shown is not true. This third law actually encourages the development of autonomy, at least on a tactical scale, in order to permit the robot to be able to monitor and project the future state of its own safety.

20.6 Artificial Intelligence and Implementing Ethics

A robot with functional or full moral agency must be intelligent in some regard. *Knowledge representations* are needed to permit the codification of explicit ethical rules, such as the military’s rules of engagement or the Geneva convention, and implicit ethical rules. *Planning and problem solving, inference, and search* are needed for deliberation as to the robot’s state and to reason about how it is impacting, or will impact, humans. Comprehending the world, especially what the humans are doing, will likely involve *computer vision*. *Learning* is touted as a way for robots to acquire ethical rules and behaviors, though it is not clear what the robot would be learning and from what input. There is an implicit expectation that robots with functional and full moral agency will be able to give orders to another robot and have it comprehend the intent despite the ambiguities in *natural language*.

20.7 Summary

All robots have an ethical impact, though it may be implicit, but robots are not necessarily moral agents

that have either functional or full moral agency. Designers are responsible for direct negative consequences (*operational morality*) that can be anticipated; robot intelligence must provide any real-time decision-making and execution (*functional morality*), but designers are still responsible. Asimov's Three Laws have captured the public's imagination as a convenient reference point for robot ethics, but the laws are deeply flawed and cannot be implemented and used in practice.

Returning to the questions posed in the introduction, the answers may be much different than expected. *Isn't it unethical to weaponize robots?* The decision to weaponize robots is a human decision and thus resides in the domain of policy. It is certainly unethical for a robotics designer to build weapons into robots without adequate safety precautions or assured reliability. The question: *If programmers apply Asimov's Three Laws, won't that be enough to protect us from injury, death, or an uprising?* should at this point elicit a chuckle from the reader as the laws are unworkable as a realistic framework for programming ethical robots. The Laws do illustrate how hard it is to capture the complexities of human society. The chapter did not directly address *Isn't it unethical to create robots and treat them like slaves?* This question is not about the design of an intelligent robot but rather about human ethics. However, it should be clear that the capabilities required for a robot to have full moral agency means it has peer-like intelligence. When a robot crosses the threshold from being a tool to being a cognitive entity with rights to protection equal to at least that of a primate, if not a human, is unclear.

20.8 Exercises

Exercise 20.1

Describe the differences between operational morality, functional morality, and full moral agency and discuss the responsibility of the designer for each of the three.

Exercise 20.2

List the four types of ethical agents and give an example of each.

Exercise 20.3

Which is NOT a problem in applying Asimov's Three Laws of Robotics to designing robots:

- a. ignores the social science of how people interact with machines
- b. is about functional morality; it assumes that all robots can make moral decisions
- c. neglects operational morality
- d. neglects system and system resilience.

Exercise 20.4

True or false: Designers are responsible for direct negative consequences that can be anticipated.

Exercise 20.5

True or false: All robots have an ethical impact, though it may be implicit. Therefore robots are necessarily moral agents.

Exercise 20.6

Look up the professional code of ethics for engineering. Imagine being in court where you are being sued for the design of an autonomous robot that failed and caused more than a \$1M in property damage. What might be the relevant portions of the code that lawyers would use to argue that you disregarded the code of ethics for your profession and

thus are culpable?

Exercise 20.7

Which approach might a driverless car use in determining how to react to an incipient crash with another car: consequentialism, deontology, or virtue ethics? Why or why not? How would you implement the approach?

Exercise 20.8

Think of at least one moral behavior that could be learned with the current state of machine learning.

[*Science Fiction*]

Exercise 20.9

Watch the 1970 movie *Colossus: The Forbin Project*. This movie was the first of the “computer takes over the world” genre, of which *War Games* is an example of the “computer takes over the military” sub-genre. Colossus is acting as a full moral agent and decides that the moral thing to do is to take over the world. If there were an international court on such things (assuming Colossus allowed it), would the designer, Dr. Forbin, be considered liable for the unexpected actions of Colossus?

20.9 End Notes

For a Roboticist’s Bookshelf

Governing Lethal Behavior in Autonomous Robots ¹³ by Ron Arkin, a professor of computer science, is considered one of the foundational books in formally examining the ethics of robots in military applications. Noel Sharkey, also a professor of computer science, takes an opposite opinion in his many papers and blogs. The two are occasionally invited to give spirited point-counterpoint talks.

The heavily cited *Moral Machines: Teaching Robots Right from Wrong* ²¹⁵ is by two philosophers, Wendell Wallach and Colin Allen, who speculate on what robots might be able to do. As would be expected, the book is less grounded in computer science than are the works of Arkin or Sharkey and the book frequently refers to Asimov’s Three Laws.

Science fiction and robotics: Asimov’s Three Laws

To make the constant interjection of Asimov’s Three Laws into robot ethics even more annoying, science fiction fans know that Asimov proposed *four* laws. The fourth, Law Zero, says that a robot may not harm humanity, or, by inaction, allow humanity to come to harm.

Science fiction and robotics: the Golem and ethics

He, She and It by Marge Piercy is set in a future Middle East where a Jewish community has created a modern Golem called Yod as a super smart soldier. The problem is that in order to create a being smart enough to do all the things that they need, they have to create a full moral agent; in this case, a full moral agent that does not want to be a soldier and would rather be a lover than a fighter. Does Yod have the right to choose his own path?

Science fiction and robotics: Ex Machina

While Yod in *He, She and It* may want to talk for pages about having the right to choose his own path, Ava, in the 2015 movie *Ex Machina*, eliminates the philosophical debate and silently takes matters into her own hands. She easily passes the Turing Test as she strives to control her existence as a sexy super-toy, but it is clear she does not share the same ethical framework as humans. As my husband pointed out, the movie probably set back the public’s acceptance and trust of robots by at least a decade.

Science fiction and robotics: A Calvinistic approach to machine ethics

Ian Tregillis’ science fiction/fantasy book *The Mechanical* posits a world where the debate over whether robots are slaves or just devices is framed as a debate between the Calvinist inventors who do not believe in free will and the Catholic Church which seeks to liberate the robots. Regardless of whether he has a soul, Jax, the mechanical protagonist who has obtained freedom, acts as a full moral agent.

Bibliography

- [1] “ACM Code of Ethics and Professional Conduct.” Report, Association for Computing Machinery. 10/6/1992. <https://www.acm.org/about-acm/acm-code-of-ethics-and-professional-conduct>.
- [2] Agarwal, S., R. R. Murphy, and J. A. Adams. “Characteristics of Indoor Disaster Environments for Small UASs.” *IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, Hokkaido, Japan, October 27, 2014.
- [3] Alberts, David S., and R. E. Hayes. 2002. *Code of Best Practice for Experimentation. Department of Defense (USA) Command and Control Research Program*. Washington D.C.: CCR Press.
- [4] Albus, James S., and Alexander M. Meystel. 2001. *Engineering of Mind: An introduction to the Science of Intelligent Systems*. Hoboken, NJ: Wiley-Interscience.
- [5] Albus, Jim S. 1995. “Rcs: A Reference Model Architecture for Intelligent Control.” *Computer* 25 (5): 56–59.
- [6] Allocca, J. A., and A. Stuart. 1984. *Transducers: Theory and Application*. Reston, VA: Reston Publishing Company, Inc.
- [7] Altendorfer, Richard, Ned Moore, Haldun Komsuoğlu, Martin Buehler, H. B. Brown, Dave McMordie, Uluc Saranli, Robert Full, and Daniel E Koditschek. 2001. “RHex: A Biologically Inspired Hexapod Runner.” *Autonomous Robots* 11 (3): 207–213.
- [8] Arbib, M. 2002. *The Handbook of Brain Theory and Neural Networks*, 2nd ed., pp. 830–834. Cambridge, MA: MIT Press.
- [9] Arbib, Michael A., and Jim-Shih Liaw. 1995. “Sensorimotor Transformations in the Worlds of Frogs and Robots.” *Artificial Intelligence* 72 (1–2): 53–79.
- [10] Argyle, Michael. 2013. *Bodily Communication*. London, England: Routledge.
- [11] Arkin, R. 1998. *Behavior-based Robotics*. Cambridge, MA: MIT Press.
- [12] Arkin, R. C., and R. R. Murphy. 1990. “Autonomous Navigation in a Manufacturing Environment.” *IEEE Transactions on Robotics and Automation* 6 (4): 445–454.
- [13] Arkin, Ronald. 2009. *Governing Lethal Behavior in Autonomous Robots*. Boca Raton, FL: CRC Press.
- [14] Asimov, Isaac. 2004. *I, Robot*. New York, NY: Bantam Spectra.
- [15] Autumn, Kellar, Yiching A. Liang, S. Tonia Hsieh, Wolfgang Zesch, Wai Pang Chan, Thomas W. Kenny, Ronald Fearing, and Robert J. Full. 2000. “Adhesive Force of a Single Gecko Foot-hair.” *Nature* 405: 681–685.
- [16] Balch, T., and R. C. Arkin. 1999. “Behavior-based Formation Control for Multirobot Teams.” *IEEE Transactions on Robotics and Automation* 14 (6): 926–939.
- [17] Balch, Tucker. May 1993. “Avoiding the Past: A Simple but Effective Strategy for Reactive Navigation.” *IEEE International Conference on Robots and Automation (ICRA-93)*, Atlanta, pp. 678–685. IEEE. 0818634502.
- [18] Balch, Tucker, Gary Boone, Thomas Collins, Harold Forbes, Doug MacKenzie, and Juan Carlos Santamar. 1995. “Io, Ganymede, and Callisto - A Multiagent Robot Trash-collecting Team.” *AI Magazine* 16 (2): 39.
- [19] Barr, Avron. 1981. *The Handbook of Artificial Intelligence*, vol. 1. William Kaufmann, Inc.
- [20] Bethel, C., and R. R. Murphy. 2010. “Review of Human Studies Methods in HRI and Recommendations.”

- International Journal of Social Robotics* 2 (4): 347–359.
- [21] Bethel, C. L., and R. R. Murphy. 2008. “Survey of Non-facial/non-verbal Methods of Affective Expressions for Appearance-constrained Robots.” *IEEE transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews* 38 (1): 83–92.
 - [22] Bhanu, Bir, Peter Symosek, and Subhodev Das. 1997. “Analysis of Terrain Using Multispectral Images.” *Pattern Recognition* 30 (2): 197–215.
 - [23] Billard, A., S. Calinon, and R. Dillmann. 2016. “Learning from Humans.” Chap. 74. Pp. 1995–2014 in *Handbook of Robotics*, 2nd ed., edited by Bruno Siciliano and Oussama Khatib. Berlin-Heidelberg: Springer.
 - [24] Billard, Aude, Sylvain Calinon, Rüdiger Dillmann, and Stefan Schaal. 2008. “Robot Programming by Demonstration.” Chap 59 in *Springer Handbook of Robotics*, 1st ed., edited by Bruno Siciliano and Oussama Khatib. Berlin- Heidelberg: Springer Verlag.
 - [25] Bonasso, R. P., R. J. Firby, E. Gat, D. Kortenkamp, and M. G. Slack. 1997. “A Proven Three-tiered Architecture for Programming Autonomous Robots.” *Journal of Experimental and Theoretical Artificial Intelligence* 9 (2): 171–215.
 - [26] Bond, Alan H., and Les Gasser. 1988. *Readings in Distributed Artificial Intelligence*. San Francisco, CA: Morgan Kaufmann.
 - [27] Braitenberg, V. 1984. *Vehicles: Experiments in Synthetic Psychology*. Cambridge, MA: MIT Press.
 - [28] Brehmer, B. 2005. *The Dynamic OODA Loop: Amalgamating Boyd’s OODA Loop and the Cybernetic Approach to Command and Control*. Department of War Studies: Swedish National Defence College. Stockholm, Sweden.
 - [29] Brooks, R. 1991. “Challenges for Complete Creature Architectures.” Pp. 434–443 in *Proceedings of the First International Conference on Simulation of Adaptive Behavior*. Cambrdige, MA: MIT Press.
 - [30] Brooks, R. 1999. *Cambrian Intelligence: The Early History of the New AI*. Cambridge, MA: MIT Press.
 - [31] Brooks, R., and A. Flynn. December 1989. “Fast, Cheap and Out of Control.” Report, AI Memo 1182, MIT AI Laboratory.
 - [32] Brooks, R. A. 1986. “A Robust Layered Control System for a Mobile Robot.” *IEEE Journal of Robotics and Automation (now IEEE Transactions on Robotics and Automation)* 2 (1): 14–23.
 - [33] Brown, Robbie. June 7, 2013. “A Swiveling Proxy That Will Even Wear a Tutu.” http://www.nytimes.com/2013/06/08/education/for-homebound-students-a-robot-proxy-in-the-classroom.html?_r=0.
 - [34] Burke, J. L., and R. R. Murphy. 2004. “Human-robot Interaction in USAR Technical Search: Two Heads are Better than One.” Pp. 307–312 in *13th IEEE International Workshop on Robot and Human Interactive Communication (ROMAN)*.
 - [35] Burke, J. L., R. R. Murphy, E. Rogers, V. J. Lumelsky, and J. Scholtz. 2004. “Final Report for the DARPA/NSF Interdisciplinary Study on Human-robot Interaction.“ *IEEE Transactions on Systems, Man, and Cybernetics, Part C* 34 (2): 103–112.
 - [36] Cai, A., T. Fukuda, and F. Arai. 1997. “Information Sharing Among Multiple Robots for Cooperation in Cellular Robotic System.” Pp. 1768–1773 in *Proceedings of 1997 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS 97)*.
 - [37] Carlson, J., and R. R. Murphy. 2005. “How UGVs Physically Fail in the Feld.” *IEEE Transactions on Robotics* 21 (3): 423–437.
 - [38] Carlson, Jennifer, Robin R. Murphy, and Andrew Nelson. 2004. “Follow-up Analysis of Mobile Robot Failures.” In *Proceedings of the 2004 IEEE International Conference on Robotics and Automation*, 5: 4987–4994. IEEE. 0780382323.
 - [39] Cattaneo, M., A. G. Cavalchini, and G. L. Rogonoi. 1996. “Design and Construction of a Robotic Milking System.” Pp. 155–160 in *the Sixth International Conference on Computers in Agriculture*, Cancun, Mexico.
 - [40] Cervantes-Perez, F. 2002. “Visuomotor Coordination in Frogs and Toads.” Pp. 1036–1042 in *The Handbook of Brain Theory and Neural Networks*, 2nd ed., edited by Michael A. Arbib. Cambridge, MA: MIT Press.

- [41] Chadwick, R. 1998, “Professional ethics” in *Routledge Encyclopedia of Philosophy*. Oxford:, Taylor and Francis, <<https://www.rep.routledge.com/articles/thematic/professional-ethics/v-1>>. doi:10.4324/9780415249126-L077-1.
- [42] Chomsky Hierarchy Wikipedia. https://en.wikipedia.org/wiki/Chomsky_hierarchy.
- [43] Choset, H., K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, and S. Thrun. 2005. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. Cambridge, MA: MIT Press.
- [44] Choset, Howie, and Joel Burdick. 2000. “Sensor-based Exploration: The Hierarchical Generalized Voronoi Graph.” *The International Journal of Robotics Research* 19 (2): 96–125.
- [45] Choset, Howie, Sean Walker, Kunayut Eiamsa-Ard, and Joel Burdick. 2000. “Sensor-based Exploration: Incremental Construction of the Hierarchical Generalized Voronoi Graph.” *The International Journal of Robotics Research* 19 (2): 126–148.
- [46] Chung, Wankyun, L-C. Fu, and T. Kröger. 2016. “Motion Control.” Chap 8 in *Springer Handbook of Robotics*, 2nd ed., edited by Bruno Siciliano and Oussama Khatib. Berlin-Heidelberg: Springer Verlag.
- [47] Chung, Wankyun, Li-Chen Fu, Su-Hau Hsu. 2008. “Motion Control.” Chap 6 in *Springer Handbook of Robotics*, 1st ed., edited by Bruno Siciliano and Oussama Khatib. Berlin-Heidelberg: Springer Verlag.
- [48] Clough, B. T. 2002. “Metrics, Schmetrics! How The Heck Do You Determine A UAV’s Autonomy Anyway?” In *Proceedings of the Performance Metrics for Intelligent Systems Workshop*. Gaithersburg, MD.
- [49] Colman, Alan, and Jun Han. 2007. “Roles, Players and Adaptable Organizations.” *Applied Ontology—Roles, an Interdisciplinary Perspective* 2 (2): 105–126.
- [50] Connell, J. 1990. *Minimalist Mobile Robotics: A Colony Architecture for an Artificial Creature*. San Diego, CA: Academic Press Professional, Inc.
- [51] Connolly, C. I., and R. A. Grupen. 1993. “The Applications of Harmonic Functions to Robotics.” *Journal of Robotic Systems* 10(7): 931–946.
- [52] Coradeschi, Silvia, Amy Loutfi, and Britta Wrede. 2013. “A Short Review of Symbol Grounding in Robotic and Intelligent Systems.” *Künstliche Intelligenz* 27 (2): 129–136.
- [53] Crandall, Jacob W., Curtis W. Nielsen, and Michael A. Goodrich. “Towards Predicting Robot Team Performance.” 2003. Pp. 906–911 in *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics*, vol. 1, Washington, D.C. 0780379527.
- [54] Davis, Ian Lane, Alonzo Kelly, Anthony Stentz, and L. Matthies. “Terrain Typing for Real Robots.” 1995. Pp. 400–405 in *Proceedings of the Intelligent Vehicles ‘95 Symposium*. Piscataway, NJ: Institute of Electrical and Electronics Engineers, Inc. 078032983X.
- [55] Dean, T., J. Allen, and Y. Aloimonis. 1995. *Artificial Intelligence: Theory and Practice*. San Francisco, CA: Benjamin/Cummings Publishing.
- [56] Dean, T., and R. P. Bonasso. Spring 1993. 1992 “AAAI robot exhibition and competition.” *AI Magazine* 14 (1): 35–48.
- [57] Dean, T., and M. Wellman. 1991. *Planning and Control*. San Mateo, CA: Morgan Kaufmann Publishers, Inc.
- [58] De Berg, Mark, Otfried Cheong, Marc van Kreveld, and Mark Overmars. 2008. *Computational Geometry: Algorithms and Applications*, 3rd ed. New York, NY: Springer.
- [59] Department of the Army. 1990. *Fm 5-33 Terrain Analysis, Report*. Washington, D.C.
- [60] Dima, Cristian S., Nicolas Vandapel, and Martial Hebert. 2004. “Classifier Fusion for Outdoor Obstacle Detection.” Pp. 665–671 in *International Conference on Robotics and Automation*, IEEE. 0780382323.
- [61] Dorf, Richard C., and Shimon Y. Nof. 1990. *Concise International Encyclopedia of Robotics: Applications and Automation*. New York, NY: John Wiley and Sons, Inc.
- [62] Doyle, Richard, Bernard, Douglas, Ed Riedel, Nicolas Rouquette, Jay Wyatt, Mike Lowry, and Pandurang Nayak. 1999. “Autonomy and Software Technology on NASA’s Deep Space One.” *IEEE Intelligent Systems* 14 (3): 10–15.

- [63] Drumheller, M. 1987. "Mobile Robot Localization Using Sonar." *IEEE Trans. Pattern Anal. Mach. Intell.* 9 (2): 325–332.
- [64] Dudek, Gregory, Michael RM Jenkin, Evangelos Milios, and David Wilkes. 1996. "A Taxonomy for Multi-agent Robotics." *Autonomous Robots* 3 (4): 375–397.
- [65] Duncan, B. A., and R. R. Murphy. 2014. "Safety Considerations for Small Unmanned Aerial Systems with Distributed Users." Pp. 1–7 in *2014 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. Hokkaido, Japan. <http://ieeexplore.ieee.org/ielx7/7002491/7017643/07017648.pdf?tp=&arnumber=7017648&isnumber=7017643>.
- [66] Duncan, Brittany A., and Robin R. Murphy. 2013. "Comfortable Approach Distance with Small Unmanned Aerial Vehicles." Pp. 786–792 in *2013 IEEE International Symposium on Robot and Human Interactive Communication (ROMAN)*. Gyeongju, South Korea.
- [67] Duncan, Brittany A., and Robin R. Murphy. 2010. "Methods and Metrics of Autonomous Take-off, Landing, and GPS Waypoint Navigation Experiments in Micro-UAVs." Pp. 1–12 in *International Conference on Unmanned Aircraft Systems*.
- [68] Edwards, G. R., R. H. Burnard, W. L. Bewley, and B. L. Bullock. 1994. "The Ground Vehicle Manager's Associate." Tech Report. AIAA-94-1248-CP, pp. 520–526.
- [69] Endsley, Mica R. 2000. "Direct Measurement of Situation Awareness: Validity and Use of SAGAT," Pp. 147–174 in *Situation Awareness Analysis and Measurement*, edited by Mica R. Endsley and Daniel J. Garland. Mahwah, NJ: Lawrence Erlbaum Associates, Inc.
- [70] Endsley, M. R. 1988. "Design and Evaluation for Situation Awareness Enhancement." Pp. 97–101 in *Proceedings of the Human Factors Society 32nd Annual Meeting*. Santa Monica, CA.
- [71] Endsley, M. R., and D. B. Kaber. 1999. "Level of Automation Effects on Performance, Situation Awareness and Workload in a Dynamic Control Task." *Ergonomics* 42 (3): 462–492.
- [72] Engelson, S. P., and D. V. McDermott. 1992. *Passive Robot Map Building with Exploration Scripts*, YALEU/DCS/TR-898, Department of Computer Science, Yale University.
- [73] Ethics Wikipedia. <https://en.wikipedia.org/wiki/Ethics>.
- [74] Everett, H. R. 1995. *Sensors for Mobile Robots: Theory and Application*. Wellesley, MA: A. K. Peters, Ltd.
- [75] Farinelli A., Iocchi L., and D. Nardi. 2004. "Multirobot Systems: A Classification Focused on Coordination." *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 34 (5): 2015–2028.
- [76] Firby, R. J., R. E. Kahn, P. N. Prokopwicz, and M. J. Swain. 1996. "An Architecture for Vision and Action," Pp. 72–79 in *Proceedings of 1995 International Joint Conference on Artificial Intelligence*. San Mateo, CA: Morgan Kaufmann Pub.
- [77] Firby, R. J., P. N. Prokopwicz, M. J. Swain, R. E. Kahn, and D. Franklin. Spring 1996. "Programming CHIP for the IJCAI-95 Robot Competition." *AI Magazine* 17 (1) 71–81.
- [78] Gage, A., and Robin R. Murphy. 2004. "Sensor Scheduling in Mobile Robots Using Incomplete Information Via Min-Conflict with Happiness." *IEEE Transactions on Systems, Man, and Cybernetics, Part B* 34 (1): 454–467.
- [79] Gardner, H. 1987. *The Mind's New Science: A History of the Cognitive Revolution*. New York, NY: Basic Books, Inc.
- [80] Gat, Erann. 1998. "Three-layer Architectures." Pp. 195–210 in *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems* edited by. D. Kortenkamp, R. Bonasso, and R. Murphy. Cambridge, MA: MIT Press.
- [81] Ghezzi, Carlos, Mehdi Jazayeri, and Dino Mandrioli. 2002. *Fundamentals of Software Engineering*, 2nd ed. Chnadler, AZ: Pearson.
- [82] Gibson, J. J. 1979. *The Ecological Approach to Visual Perception*. London: Psychology Press.
- [83] Gifford, K. K., and G. W. Morgenthaler. October 1995. Optimal Path Coverage Strategies for Planetary Rover Vehicles.

- [84] Haber, R. N., and L. Haber. 1990. "Why Mobile Robots Need a Spatial Memory." *SPIE Sensor Fusion III:3-D Perception and Recognition* 1383: 411–424.
- [85] Haigh, Karen Zita, and Manuela M Veloso. 1999. Learning Situation-dependent Costs: Improving Planning from Probabilistic Robot Execution. *Robotics and Autonomous Systems* 29: 145–174.
- [86] Hall, Edward T. 1966. *The Hidden Dimension*. Garden City, NY: Doubleday & Co.
- [87] Hart, Sandra G. 1989. "NASA-task Load Index (NASA-TLX); 20 Years Later." In *Proceedings of the 33rd Annual Meeting of Human Factors and Ergonomics Society* 50: 904–908.
- [88] Henderson, T., and E. Shilcrat. 1984. "Logical Sensor Systems." *Journal of Robotics Systems* 1 (2): 169–193.
- [89] Henkel, Z., C. Y. Kim, R. R. Murphy, & B. Shrewsbury, Oct. 21–26, 2013. "RESPOND-R Test Instrument: A Summer Institute 2013 Case Study," presented at the *IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR 2013)*, Sweden, pp. 1–6.
- [90] Hirose, S. April 1996. "Untitled talk at the Planetary Rover Technology and Systems Workshop." *IEEE International Conference on Robotics and Automation*.
- [91] Honderd, G., W. Jongkind, C. Klomp, J. Dessim, and R. Paliwoda. 1991. "Strategy and Control of an Autonomous Cow-milking Robot System." *Robotics and Autonomous Systems* 7 (2–3): 165–179.
- [92] Horn, Berthold K. P. 1986. *Robot Vision*. Cambridge, MA: MIT Press.
- [93] Howard, Ayanna, and Homayoun Seraji. 2001. "Vision-based Terrain Characterization and Traversability Assessment." *Journal of Robotic Systems* 18 (10): 577–587.
- [94] Huang, C-M., and B. Mutlu. 2012. "Robot Behavior Toolkit: Generating Effective Social Behaviors for Robots." In *Proceedings of the 7th ACM/IEEE Conference on Human-Robot Interaction (HRI 2012)*.
- [95] Huang, Hui-Min, Kerry Pavek, James Albus, and Elena Messina. March 2005. "Autonomy Levels for Unmanned Systems (ALFUS) Framework: An Update." Pp. 439–448 in *Proceedings of the 2005 SPIE Defense and Security Symposium*.
- [96] Hyams J., M. Powell, and R. R. Murphy. 2000. Position "Estimation and Cooperative Navigation of Micro-Rovers using Color Segmentation." *Autonomous Robots, special issue* 9 (1): 7–16.
- [97] Jet Propulsion Laboratory. 2016. Mars Exploration Rovers <http://mars.nasa.gov/mer/overview/>.
- [98] Johnson, Constance M., Douglas Johnston, P. Kenyon Crowley, Helen Culbertson Helga E. Rippen, David J. Damico, and Catherine Plaisant. 2011. System Usability Scale AHRQ. https://healthit.ahrq.gov/sites/default/files/docs/citation/EHR_Usability_Toolkit_Background_Report.pdf.
- [99] Kaber, David B., and Mica R. Endsley. 1997. "Out-of-the-loop Performance Problems and the Use of Intermediate Levels of Automation for Improved Control System Functioning and Safety." *Process Safety Progress* 16 (3): 126–131.
- [100] Kajita, S., and B. Espiau. 2008. "Legged Robots." Chap. 16. in *Springer Handbook of Robotics*, edited by Bruno Siciliano and Oussama Khatib. Berlin-Heidelberg: Springer.
- [101] Kessel, C. J., and C. D. Wickens. 1982. "The Transfer of Failure-detection Skills between Monitoring and Controlling Dynamic Systems." *Human Factors* 24 (1): 49–60.
- [102] Keymeulen, Didier, Marc Durantez, Kenji Konaka, Yasuo Kuniyoshi, and Tetsuya Higuchi. 1997. "An Evolutionary Robot Navigation System Using a Gate-level Evolvable Hardware." Pp. 173–188 in *Learning Robots. European Workshop on Learning Robots 1997. Lecture Notes in Computer Science*, vol. 1514. edited by A. Birk and J. Demiris. Berlin-Heidelberg: Springer.
- [103] Khatib, Oussama. 1986. "Real-time Obstacle Avoidance for Manipulators and Mobile Robots." 1986. *The International Journal of Robotics Research* 5 (1): 90–98. <http://ijr.sagepub.com/content/5/1/90.abstract>.
- [104] Klein, Gary. 1999. *Sources of Power: How People Make Decisions*. Cambridge, MA: MIT Press.
- [105] Klein, Gary, David D Woods, Jeffrey M Bradshaw, Robert R Hoffman, and Paul J Feltovich. 2004. "Ten Challenges for Making Automation a 'Team Player' in Joint Human-agent Activity." *IEEE Intelligent Systems* 19 (6): 91–95.

- [106] Konolige, K., and K. Myers. 1998. *The Saphira Architecture for Autonomous Mobile Robots*, edited by R. Murphy, D. Kortenkamp, and R. Bonasson. Cambridge, MA: MIT Press.
- [107] Kortenkamp, D., and T. Weymouth. 1994. "Topological mapping for mobile robots using a combination of sonar and vision sensing." Pp. 974–984 in *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*.
- [108] Kortenkamp, David, R Peter Bonasso, and Robin Murphy, editors. 1998. *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems*. Cambridge, MA: MIT Press.
- [109] Kuipers, B., and Y-T. Byun. 1991. "A Robot Exploration and Mapping Strategy Based on a Semantic Hierarchy of Spatial Representations." *Robotics and Autonomous Systems* 8: 47–63.
- [110] Kurzweil, Ray. 2001. The Age of Intelligent Machines |A Coffeehouse Conversation on the Turing Test. <http://www.kurzweilai.net/the-Age-of-Intelligent-Machines-a-Coffeehouse-Conversation-on-the-Turi-Test>.
- [111] Lagnemma, K. D., and S. Dubowsky. 2002 "Terrain Estimation for High-Speed Rough-Terrain Autonomous Vehicle Navigation." Pp. 256–266 in *Proceedings of the International Society for Optics and Photonics* (SPIE vol. 4715).
- [112] Larson, Amy C., Guleser K. Demir, and Richard M. Voyles. 2005. "Terrain Classification Using Weakly-structured Vehicle/terrain Interaction." *Autonomous Robots* 19 (1): 41–52.
- [113] Latombe, J. C. 1990. *Robot Motion Planning*. Boston, MA: Kluwer Academic Publishers.
- [114] Levis, Alexander H., and Lee W. Wagenhals. 2000. C41SR Architectures: Developing a Process for C41SR Architecture Design. *Systems Engineering* 3: 225–247.
- [115] Levitt, T. S., and D. T. Lawton. 1990. "Qualitative Navigation for Mobile Robots." *Artificial Intelligence* 44 (3): 305–360.
- [116] Lewis, H. R., and C. H. Papadimitriou. 1981. *Elements of the Theory of Computation*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, Inc.
- [117] Lim, W., and J. Eilbert. 1991. "Plan-Behavior Interaction in Autonomous Navigation." Pp. 464–475 in *Proceedings of the International Society for Optics and Photonics* (SPIE vol. 1388).
- [118] Lorenz, Bernd, Francesco Di Nocera, Stefan Röttger, and Raja Parasuraman. 2001. "The Effects of Level of Automation on the Out-Of-The-Loop Unfamiliarity in a Complex Dynamic Fault-Management Task During Simulated Spaceflight Operations Aerospace Systems." In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* 45 (2) 44–48.
- [119] Maes, P., and R. A. Brooks. 1990. "Learning to Coordinate Behaviors." Proceedings of the Eighth National Conference on Artificial intelligence 2: 796–802.
- [120] Marr, D. 1982. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. San Francisco, CA: W.H. Freeman and Co.
- [121] Massie, Thomas, and J. K. Salisbury. "The Phantom Haptic Interface: A Device for Probing Virtual Objects." In *Proceedings of the ASME Winter Annual Meeting, Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, vol. 1, 295–301.
- [122] Matarić, Maja J., and François Michaud. 2008. "Behavior-Based Systems." Chap. 38 in *Springer Handbook of Robotics*. 1st ed. Edited by Bruno Siciliano and Oussama Khatib. Berlin-Heidelberg: Springer Verlag.
- [123] Matarić, M. "Minimizing Complexity in Controlling a Mobile Robot Population." 1992. Pp. 830–835 in *Proceedings of the IEEE International Conference on Robotics and Automation*.
- [124] Matarić, M. 1992a. "Behavior-Based Control: Main Properties and Implications." *Proceedings of Workshop on Intelligent Control Systems, International Conference on Robotics and Automation*, Nice, France, May.
- [125] McCarley, J. S., and C. D. Wickens. 2005. "Human Factors Implications of UAVs in the National Airspace." Report, Institute of Aviation, University of Illinois at Urbana-Champaign.
- [126] McKee, Gerard. 2008. "What Is Networked Robotics?" In *Informatics in Control. Automation and Robotics*, 15: 35–45.
- [127] Merriam-Webster's Collegiate Dictionary. 1998. 10th ed. Springfield, MA: Merriam-Webster.

- [128] Meyer, Jean-Arcady, and Agnès Guillot. 2008. 60. “Biologically-inspired Robots.” Chap. 60. Pp. 1395–1422 in *Handbook of Robotics*, edited by Bruno Siciliano and Oussama Khatib. Berlin-Heidelberg: Springer Verlag.
- [129] Meystel, A. 1990. “Knowledge Based Nested Hierarchical Control.” Pp. 63–152 in *Advances in Automation and Robotics*, vol. 2, edited by G. Saridis. Greenwich, CT: JAI Press.
- [130] Michaud, and M. Nicolescu. 2016. 13. Behavior-Based Systems. Pp. 307–328 in *Springer Handbook of Robotics* 2nd ed.
- [131] Mikulic, Dinko. 2012. *Design of demining machines*. London: Springer-Verlag.
- [132] Mitchell, Tom M. 1997. Machine learning. Boston, MA: WCB McGraw-Hill.
- [133] Moor, James H. 2006. “The nature, importance, and difficulty of machine ethics.” *IEEE Intelligent Systems* 21 (4): 18–21.
- [134] Moravec, Hans. 1988. *Mind Children: The Future of Robot and Human Intelligence*. Cambridge, MA: Harvard University Press.
- [135] Moravec, Hans. 2000. *Robot: Mere machine to transcendent mind*. New York, NY: Oxford Press.
- [136] Mori, M. 1970. “The Uncanny Valley.” Translated by Karl F. MacDorman and Takashi Minato. *IEEE Robotics & Automation Magazine* 19 (2): 98–100.
- [137] Murphy R. R., J. Kravitz, S. Stover, and R. Shoureshi. 2009. “Mobile Robots in Mine Rescue and Recovery.” *IEEE Robotics and Automation Magazine* 16 (2): 91–103.
- [138] Murphy, R. R., and A. Mali. 1997. “Lessons Learned in Integrating Sensing into Autonomous Mobile Robot Architectures.” *Journal of Experimental and Theoretical Artificial Intelligence* 9 (2–3): 191–209.
- [139] Murphy, R. R., and S. Stover. 2008. “Rescue Robots for Mudslides: A Descriptive Study of the 2005 La Conchita Mudslide Response: Field Reports. *Journal of Field Robotics, Special Issue on Search and Rescue Robots* 25 (1–2): 3–16.
- [140] Murphy, R. R., C. Lisetti, R. Tardif, L. Irish, and A. Gage. 2002. Emotion-based control of cooperating heterogeneous mobile robots. *IEEE Transactions on Robotics and Automation, special issue on Multi-Robot Systems* 18 (5): 744–757.
- [141] Murphy, Robin R. 2014. *Disaster Robotics*. Cambridge, MA: MIT Press.
- [142] Murphy, Robin R. 2015. Meta-analysis of Autonomy at the DARPA Robotics Challenge Trials. *Journal of Field Robotics, special issue on DARPA Robotics Challenge* 32 (2): 189–191.
- [143] Murphy, Robin R., and Debra Schreckenghost. March 2013. “Survey of Metrics for Human-robot Interaction.” Pp. 197–198. in *Proceedings of the 8th ACM/IEEE International Conference on Human-robot Interaction (HRI)*.
- [144] Murphy, Robin R., and James Shields. 2012. The Role of Autonomy in DODd Systems, Report, Department of Defense, Defense Science Board Task Force Report.
- [145] Murphy, Robin R., and David D. Woods. 2009. Beyond Asimov: The Three Laws of Responsible Robotics. *Intelligent Systems* 24 (4): 14–20.
- [146] Murphy, Robin R., Ken Hughes, Alisa Marzilli, and Eva Noll. 1997. “Integrating Explicit Path Planning with Reactive Control for Mobile Robots Using Trulla.” *Robotics and Autonomous Systems* 27 (4): 225–245.
- [147] Murphy, Robin R., Satoshi Tadokoro, and Alexander Kleiner. 2016. “Disaster Robotics,” Chap. 60. Pp. 1577–1604 in *Springer Handbook of Robotics*, 2nd ed., edited by Bruno Siciliano and Oussama Khatib. Berlin-Heidelberg: Springer.
- [148] Murphy, R. R. “An Artificial Intelligence Approach to the 1994 AUVs Unmanned Ground Vehicle Competition.” Oct. 1995. Pp. 1723–1728 in *1995 IEEE International Conference on Systems, Man, and Cybernetics*, Vancouver, B.C.
- [149] Murphy, R. R. 1996a. “Biological and Cognitive Foundations of Intelligent Sensor Fusion.” *IEEE Transactions on Systems, Man, and Cybernetics* 26 (1): 42–51.
- [150] Murphy, R. R., November 1996. “Use of Scripts for Coordinating Perception and Action.” In *Intelligent Robots*

and Systems 96, *IROS-96, Proceedings of the 1996 IEEE/RSJ, International Conference* 1: 156–161.

- [151] Murphy, R. R. 2002. “Marsupial Robots.” In *Robot Teams: From Diversity to Polymorphism*, edited by T. Balch and L. E. Parker. Wellesley, MA: A. K. Peters.
- [152] Murphy, R. R. 2004. “National Science Foundation Summer Field Institute for Rescue Robots for Research and Response (R4).” *AI Magazine* 25 (2): 133–136.
- [153] Murphy, R. R., and J. L. Burke. 2008. From remote tool to shared roles. *IEEE Robotics and Automation Magazine, special issue on New Vistas and Challenges for Teleoperation* 15 (4): 39–49.
- [154] Murphy, R. R., and J. L. Burke. 2010. “The Safe Human-robot Ratio.” Chap. 3 in *Human-Robot Interactions in Future Military Operations*, edited by Michael Barnes and Florian Jenssch. Boca Raton, FL: CRC Press.
- [155] Mutlu, Bilge, and Jodi Forlizzi. 2008. “Robots in Organizations: The Role of Workflow, Social, and Environmental Factors in Human-robot Interaction.” Pp. 287–294 in *Proceedings of the 3rd ACM/IEEE International Conference on Human Robot Interaction*. New York, NY: ACM.
- [156] Mutlu, Bilge, Fumitaka Yamaoka, Takayuki Kanda, Hiroshi Ishiguro, and Norihiro Hagita. March 2009. “Nonverbal Leakage in Robots: Communication of Intentions through Seemingly Unintentional Behavior.” Pp. 69–76. In *Proceedings of the 4th ACM/IEEE International Conference on Human Robot Interaction*.
- [157] Nagatani, Keiji, Seiga Kiribayashi, Yoshito Okada, Kazuki Otake, Kazuya Yoshida, Satoshi Tadokoro, Takeshi Nishimura, Tomoaki Yoshida, Eiji Koyanagi, Mineo Fukushima, and Shinji Kawatsuma. 2013. “Emergency Response to the Nuclear Accident at the Fukushima Daiichi Nuclear Power Plants Using Mobile Rescue Robots.” *Journal of Field Robotics* 30 (1): 44–63.
- [158] National Research Council. 2003. *Technology Development for Army Unmanned Ground Vehicles*. Washington, DC: The National Academies Press.
- [159] Neisser, U. August, 1989. “Direct Perception and Recognition as Distinct Perceptual Systems.” Address presented to the Cognitive Science Society.
- [160] Neisser, Ulric. 1976. *Cognition and Reality - Principles and Implications of Cognitive Psychology*. New York, NY: W. H. Freeman and Company.
- [161] Nelson, R. C. 1991. “Visual Homing Using an Associative Memory.” *Biological Cybernetics* 65 (4): 281–291.
- [162] Neumann, Peter G. 2016. Automated Car Woes—Whoa There! *ACM Ubiquity* (July).
- [163] Nolfi, S., and D. Floreano. 2000. *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. Cambridge, MA: MIT Press.
- [164] Oliveira E., K. Fischer, and O. Stephankova. 1999. “Multi-Agent Systems: Which Research for Which Applications.” *Robotics and Autonomous Systems* 27: 91–106.
- [165] Parker, L. E., D. Rus, and G. S. Sukhatme. 2016. “Multiple Mobile Robot Systems.” Chap. 53 in Springer Handbook of Robotics, 2nd ed., edited by Bruno Siciliano and Oussama Khatib. Berlin-Heidelberg: Springer Verlag.
- [166] Parker, Lynne. 2008. “Multiple Mobile Robot Systems.” Chap 40 in *Springer Handbook of Robotics*, 1st ed., edited by Bruno Siciliano and Oussama Khatib. Berlin-Heidelberg: Springer Verlag.
- [167] Penrose, Roger. 1989. *The Emperor’s New Mind: Concerning Computers, Minds, and the Laws of Physics*. New York, NY: Oxford University Press, Inc.
- [168] Peschel, J. M., B. A. Duncan, and R. R. Murphy. 2012. “Exploratory Results for a Mission Specialist Interface in Micro Unmanned Aerial Systems.” *The 2012 International Conference on Collaboration Technologies and Systems (CTS)*, 131–140.
- [169] Picard, R. W. 1997. *Affective Computing*. Cambridge, MA: MIT Press.
- [170] Pratt, Kevin, and R. R. Murphy. 2012. “Protection from Human Error: Guarded Motion Methodologies for Mobile Robots.” *IEEE Robotics and Automation Magazine* 19 (4): 36–47.
- [171] Pratt, Kevin, Robin Murphy, Sam Stover, and Chandler Griffin. 2009. “CONOPS and Autonomy Recommendations for VTOL Small Unmanned Aerial System Based on Hurricane Katrina Operations.” *Journal of Field Robotics* 26 (8): 636–650.

- [172] Prodhan, Georgina. 2016 “Europe’s robots to become ‘electronic persons’ under draft plan.” *Science News*. June 21, 2016 |1:07pm EDT.
- [173] Raibert, Marc H. 1986. *Legged Robots that Balance*. Cambridge, MA: MIT Press.
- [174] Rasmussen, Jens. 1986. *Information Processing and Human-machine Interaction: An Approach to Cognitive Engineering*. New York, NY: Elsevier Science Inc.
- [175] Reeves, B., and C. Nass. 1996. *The Media Equation: How People Treat Computers, Television, and New Media like Real People and Places*. New York, NY: Cambridge University Press.
- [176] Rich, Elaine, and Kevin Knight. 1991. *Artificial Intelligence*. New York, NY: McGraw Hill Companies.
- [177] RoboLaw: Regulating Emerging Robotic Technologies in Europe: Robotics facing Law and Ethics. September 2014. FP7-SCIENCE-IN-SOCIETY-2011-1. Project No.: 289092. SSSA. <http://www.robolaw.eu>.
- [178] Rogers, Erika. 2004. “Human-Robot Interaction.” Pp. 328–332 in *Berkshire Encyclopedia of Human-Computer Interaction*.
- [179] Rogers, Everett M. 2003. *Diffusion of Innovations*, 5th ed. New York, NY: Simon and Schuster.
- [180] Rumelhart, David E, James L McClelland, and The PDP Research Group. 1986. Vol. 1, *Parallel Distributed Processing*. Cambridge, MA: MIT Press.
- [181] Russell, Stuart, and Peter Norvig. 2009. *Artificial Intelligence: A Modern Approach*. Cambridge, MA: MIT Press.
- [182] Ryan Firebee Wikipedia. https://en.wikipedia.org/wiki/Ryan_Firebee.
- [183] Saranli, Uluç, Martin Buehler, and Daniel E. Koditschek. 2001. “RHex: A Simple and Highly Mobile Hexapod Robot.” *The International Journal of Robotics Research* 20 (7): 616–631.
- [184] Sarmiento, T. A., B. A. Duncan, and R. R. Murphy. 2015. “Preliminary Analysis of Reconstructions from Aerial Images of Disaster Props.” Pp. 1–2 in *SSRR 2015 2015 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*.
- [185] Schaal, Stefan, Jan Peters, Jun Nakanishi, and Auke Ijspeert. “Learning Movement Primitives.” Pp. 561–572 in *Robotics Research. The Eleventh International Symposium*. Berlin-Heidelberg: Springer Velag.
- [186] Schach, Stephen R. 1996. *Classical and Object-oriented Software Engineering*, 3rd ed. Burr Ridge, IL: Irwin Professional Publishers.
- [187] Schank, Roger C., and Robert P. Abelson. 2013. *Scripts, Plans, Goals, and Understanding: An Inquiry into Human Knowledge Structures*. London: Psychology Press.
- [188] Scholtz, J. 2003. “Theory and Evaluation of Human Robot Interactions.” Pp. 125–134 in *Proceedings of the 36th Annual Hawaii International Conference on System Sciences*.
- [189] Schultz, A. Adams, W., and B. Yamauchi. May 1999. “Integrating Exploration, Localization, Navigation and Planning with a Common Representation.” *Autonomous Robots* 6 (3): 293–308.
- [190] Shamah, Benjamin. “Experimental Comparison of Skid Steering vs. Explicit Steering for a Wheeled Mobile Robot.” (masters thesis, Carnegie Mellon University, 1999).
- [191] Sheridan, T. 1992. *Telerobotics, Automation, and Human Supervisory Control*. Cambridge, MA: MIT Press.
- [192] Sheridan, T. 1993. Space Teleoperation Through Time Delay: Review and Rrognosis. *IEEE Transactions on Robotics and Automation* 9 (5): 592–605.
- [193] Shneiderman, Ben, Catherine Plaisant, Maxine Cohen, and Steven Jacobs. 2009. *Designing the User Interface: Strategies for Effective Human-computerInteraction*. 5th ed. Upper Saddle River, NJ: Prentice-Hall.
- [194] Siegwart, Roland, Illah Reza Nourbakhsh, and Davide Scaramuzza. 2011. *Introduction to Autonomous Mobile Robots*, 2nd ed. *Intelligent Robotics and Autonomous Agents series*. Cambridge MA: MIT Press.
- [195] Simon, Herbert A. 1996. *The Sciences of the Artificial*, 3rd ed. Cambridge, MA: MIT Press.
- [196] Slack, M. 1993. Navigation Templates: Mediating Qualitative Guidance and Quantitative Control in Mobile Robots. *IEEE Transactions on Systems, Man, and Cybernetics* 23 (2): 452–466.
- [197] Smith, R., and P. Cheeseman. 1986. On the Representation of and Estimation of Spatial Uncertainty.

- International Journal of Robotics Research* 5: 56–68.
- [198] “Sojourners ‘Smarts’ Reflect Latest in Automation.” Jet Propulsion Laboratory Press Release, Aug. 8, 1997.
- [199] Stark, L., and K. Bowyer. 1996. *Generic Object Recognition Using Form and Function*. Vol. 10 of *Series in Machine Perception and Artificial Intelligence*. Singapore: World Scientific.
- [200] Stone, Peter, and Manuela Veloso. 2002. A survey of multiagent and multirobot systems. *Robot Teams: From Diversity to Polymorphism*.
- [201] Suarez, Jesus, and Robin R. Murphy. 2012. Using the Kinect for Search and Rescue Robotics. Pp. 1–2 in *2012 IEEE International Symposium on Safety, Security, and Rescue Robotics* (SSRR).
- [202] Swain, M. J., and D. H. Ballard. 1991. Color Indexing. *International Journal of Computer Vision* 7 (1): 11–32.
- [203] Tadokoro, S., Robin Murphy, S. Stover, W. Brack, M. Konyo, T. Nishimura, and O. Tanimoto. 2009. “Application of Active Scope Camera to Forensic Investigation of Construction Accident.” Pp. 47–50 in *IEEE International Workshop on Advanced Robotics and Its Social Impacts* (ARSO2009).
- [204] Teichman, Alex, and Sebastian Thrun. 2012. “Tracking-based Semi-supervised Learning.” *The International Journal of Robotics Research* 31 (7): 804–818.
- [205] Thorpe, C. E. 1984. “Path Relaxation: Path Planning for a Mobile Robot.” Technical Report CMU-RI-TR-84-5. Pittsburgh, PA: Carnegie Mellon University.
- [206] Thrun, S., M. Beetz, M. Bennewitz, W. Burgard, A. B. Cremers, F. Dellaert, D. Fox, D. Hähnel, C. Rosenberg, N. Roy, J. Schulte, and D. Schulz. 2000. Probabilistic Algorithms and the Interactive Museum Tour-guide Robot Minerva. *International Journal of Robotics Research* 19 (11): 972–999.
- [207] Thrun, S., W. Burgard, and D. Fox. 2005. *Probabilistic Robotics*. Cambridge, MA: MIT Press.
- [208] Tucker, Balch. 2002. “Taxonomies of Multi-Robot Task and Reward.” *Robot Teams: From Diversity to Polymorphism*. Edited by T. Balch and L. Parker. Natick, MA: A. K. Peters, Ltd.
- [209] Umbaugh, Scott E. 1998. *Computer Vision and Image Processing: A Practical Approach Using CVIPtools*. Upper Saddle River, NJ: Prentice-Hall.
- [210] US Department of Health and Human Services. September 21, 2016. System Usability Scale <https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html>.
- [211] Uttal, W. R. 1989. “Teleoperators.” *Scientific American* 261 (6): 124–129.
- [212] Vicente, Kim J. 1999. *Cognitive Work Analysis: Toward Safe, Productive, and Healthy Computer-based Work*. Boca Raton, FL: CRC Press.
- [213] Voshell, Martin Gregory. 2009. Planning Support for Running Large Scale Exercises as Learning Laboratories (masters thesis, The Ohio State University, 2009).
- [214] Voshell, Martin Gregory, David D. Woods, and Flip. Phillips. Overcoming the Keyhole in Human-Robot Coordination: Simulation and Evaluation. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, Vol. 49, 442–446.
- [215] Wallach, Wendell, and Colin Allen. 2009. *Moral Machines: Teaching Robots Right from Wrong*. Oxford, England: Oxford University Press.
- [216] Wang, Youqing, Furong Gao, and Francis J Doyle,III. 2009. Survey on Iterative Learning Control, Repetitive Control, and Run-to-Run Control. *Journal of Process Control* 19 (10): 1589–1600.
- [217] Wieber, P-B., R. Tedrake, and S. Kuindersma. 2016. “Modeling and Control of Legged Robots,” Pp. 1203–1234 in *Springer Handbook of Robotics*, 2nd ed., edited by Bruno Siciliano and Oussama Khatib. Berlin-Heidelberg: Springer Verlag.
- [218] Williams, Brian C, Michel D. Ingham, Seung Chung, Paul Elliott, Michael Hofbaur, and Gregory T. Sullivan. 2004. “Model-based Programming of Fault-Aware Systems.” *AI Magazine* 24 (4): 61–75.
- [219] Winston, Patrick Henry. 1992. *Artificial Intelligence*. 3rd ed. Reading, MA: Addison-Wesley Publishing Company.
- [220] Woods, D., and E. Hollnagel. 2006. *Joint Cognitive Systems: Patterns in Cognitive Systems Engineering*. Boca

Raton, FL: CRC Press.

- [221] Woods, David D. 1984. "Visual Momentum: A Concept to Improve the Cognitive Coupling of Person and Computer." *International Journal of Man-Machine Studies* 21 (3): 229–244.
- [222] Yim, Mark, Wei-Min Shen, Behnam Salemi, Daniela Rus, Mark Moll, Hod Lipson, Eric Klavins, and Gregory S Chirikjian. 2007. "Modular Self-Reconfigurable Robot Systems [Grand Challenges of Robotics]." *IEEE Robotics and Automation Magazine* 14 (1): 43–52.

Index

3 Ds, 11
3T, 95
4-connected neighbors, 395
8-connected neighbors, 395

A* search algorithm, 396, 398, 399, 401, 405, 406, 462
abstract behavior, 188, 203, 205
actigrams, 175
action selection, 245
action-oriented perception, 171
action-perception cycle, 155, 156, 162
activation function, 457
active perception, 156
add-list, 325
admissibility condition, 398, 471
affective computing, 177
affordance, 154, 156, 158, 162, 180, 185, 203, 208, 212, 215, 218, 310
agent, 20
AI areas, 13–15
allothetic, 142
anchoring, 334
artificial intelligence, definition, 3
artificial landmarks, 361
artificial neural network, feedforward, 458
artificial neural network, recurrent, 458
artificial neural networks (ANN), 455
artificial neural networks, backpropagation, 458
artificial neural networks, deeplearning, 458
artificial neural networks, hiddenlayers, 457
assistive agent, 515
attention, 358
automatic guided vehicles, 27
autonomous underwater vehicle, 10
autonomy, constraint, 81
autonomy, intentional, 81
autonomy, none, 80
autonomy, process, 81
autonomy, systems-state, 81

axioms, 326

ballistic control, 26
BDI, 334
behavior, 141, 144
behavior, cancellation, 170
behavior, conscious, 141
behavior, dominance, 170
behavior, emergent, 140
behavior, equilibrium, 170
behavior, fixed-action patterns, 142
behavior, implicit chaining, 167
behavior, innate, 162
behavior, innate with memory, 163
behavior, learned, 164
behavior, mathematical definition, 146
behavior, reactive, 141
behavior, reflexes, 142
behavior, reflexive, 141, 142
behavior, sequence of innate, 163
behavior, taxes, 142
behavior table, 565
behavioral coordination, 186
behaviors, 70
belief-desires-intentions (BDI), 534
binary image, 264
biomimetic, 5
black factory, 28
bounded rationality, 49, 81
box canyon, 200

Capek, K., 4, 5
Cartographer, 343
causal chain, 220
central pattern generator, 244, 245, 247
closed-loop control, 26
closed-world assumption, 47, 134, 322, 332
cloud robotics, 474, 489
coexistence, ignorant, 501
coexistence, informed, 502
coexistence, intelligent, 502
cognitive fatigue, 117
cognitive task analysis (CTA), 528
cognitive work analysis (CWA), 529
cognizant failure, 346
common ground, 534
communication, direct, 495
communication, indirect, 495
compound releasers, 166
computational theory, 134, 135, 137, 141 264, 275, 279
conceptual dependency theory, 532

configuration space (Cspace), 389–393, 396, 402
consequentialism, 590
contract net protocols, 503
cooperative mobility, 500
coordination, competing methods, 187
coordination, cooperating methods, 187, 188
coordination, sequencing methods, 187
coordination function, 186
correspondence, 288
crawling, 235
credit assignment problem, 460
cross-talk, 298
cybernetics, 23

D* search algorithm, 405–407, 411
Darkspot, 519
DARPA, 31
Dartmouth Conference, 31
data, experimental, 576
data, six categories, 577
data coupling, 201
Deep Space 1, 56
degrees of freedom, 389
delete-list, 325
deliberative function, generating, 73
deliberative function, implementing, 73
deliberative function, monitoring, 73
deliberative function, selecting, 73
deliberative functions, 73
deontology, 590
depth map, 292
design, ecological process, 562
design, five questions, 560
difference, 324
difference evaluator, 324
difference table, 324
digital elevation model (DEM), 429, 430
digital surface maps (DSM), 430
digital terrain elevation data (DTED), 427
digitization bias, 395
Dijkstra single source shortest path algorithm, 364, 372, 379, 383, 384, 415
direct perception, 154, 158, 162, 179, 180
disparity, 290
distinctive place, 365
distributed AI, 488
distributed problem solving, 488
domain, formative, 529
domain, normative, 528
drone, 35
dynamic balance, 241

ecological approach, 156
emergent social behavior, 486
epipolar lines, 291
ergonomics, 517, 521
ethics, ethical impact agents, 588
ethics, explicit ethical agents, 589
ethics, full ethical agents, 589
ethics, implicit ethical agents, 588
ethnogram, 176
events, 343
evolutionary robotics, 468
evolutionary robotics, epigenetics, 470
execution approval, 95, 96, 108
experimentation, concept experimentation, 574
experimentation, controlled, 573
experimentation, exercise, 573
experimentation, participant-observer, 575
exploration, 435
exploration, frontier-based, 436
exploration, generalized Voronoigraph (GVG), 437, 438
exproprioception, 112, 275
exteroception, 112, 275

failed precondition, 330
failure, external, 572
failure, human error, 572
failure, non-terminal, 573
failure, physical, 572
failure, terminal, 573
fault detection identificationand recovery (FDIR), 346
fault detection identificationand recovery (FDIR),fail upwards, 346
fault detection identificationand recovery (FDIR),model-based reasoning, 347
feedback, 26
finite state automata, 213
focus-of-attention, 221
foreshortening, 297, 298
four grammars, 532
frame, 134
frame problem, 47, 156, 322,332
full moral agency, 590
functional cohesion, 200
fuzzy logic, 187

gait, 240, 245
gateway, 361, 388, 437
General Problem Solver, 324
generalized Voronoi graph (GVG),390, 393
genetic algorithm, 470
genetic algorithm, crossover function, 471
genetic algorithm, genome, 470
gestures, 537

gestures, deictic, 537
gestures, signaling, 537
global world model, 162
goal node, 396
goal state, 324
GRUFF, 161
guarded motion, 108, 114

heterogeneous MRS, 496
heuristic function, 398
hill-climbing algorithm, 366,382, 383
holonomic, 231
homeostasis, 164, 177, 224
homogeneous MRS, 496
human out-of-the-loop (OOTL) control problem, 53, 57, 120, 539, 548
human supervisory control, 108
human-robot interaction, biometrics,544
human-robot interaction, interviews,surveys, journaling, 543
human-robot interaction, metrics, 546
human-robot interaction, modes ofcooperation, 514
human-robot interaction, observation, 543
human-robot interaction, specializedtests, 545
human-robot interaction metrics,coactivity, 546
human-robot interaction metrics,efficiency, 546
human-robot interaction metrics,productivity, 546
human-robot interaction metrics,reliability, 546
human-robot interaction metrics,safety, 546

idiothetic, 142
indexing, 221
industrial manipulator, 24
information collection, 359
initial node, 396
initial state, 324
innate releasing mechanism, 162, 165, 171, 172, 178
intelligent agent, 7
intelligent robot, 7
interest operator, 288
internal state, 163, 164, 167, 171, 173, 175–178, 205

joint cognitive system, 21, 108, 514, 515

keyhole effect, 117
Kinect, 15
Kiva Systems, 29
Kosslyn, S., 15

landmark, 361, 383, 384
language generation, 531
language understanding, 531
latency, 117

learning, 447
learning, critic, 474
learning, decision trees, 452
learning, induction, 450
learning, reinforcement, 450, 460
learning, support vectormachines, 452
learning, utility functions, 461
learning by example, clustering, 449
learning by example, data mining, 449, 454
learning by example, semi-supervised, 450
learning by example, supervised, 449
learning by example, unsupervised, 449
learning element, 474
leg event, 239
levels of autonomy, 76
lidar, 302, 305
local control strategy, 366
local minima problem, 199
local perceptual spaces, 335, 336
local world model, 156, 162
localization, 419, 420
localization, Extended Kalman Filter, 423
localization, grid-based, 423
localization, Markov, 421, 423
localization, Monte Carlo, 424, 425
logistic function, 457
Luddites, 28

magnitude profile, 191, 193, 194
magnitude profile, exponential, 193
magnitude profile, linear, 193
magnitude profiles, 193
manual control, 109
mapping, 424
Markov decision process, 460
marsupial robot, 497
meadow map, 391
means-ends analysis, 324
Media Equation, 518, 552
metric navigation, 360
minimal spanning tree algorithm, 384
Mission Planner, 341–343, 346
mixed team, 21
morality, functional, 590
morality, operational, 589
motes, 8
motivation, 164, 166, 167
motor schema, 144
multiagent systems, 488
multirobot system, coordination, 493
multirobot system, systems, 494

multirobot system, task, 490
multirobot systems, 484
multirobot systems, aware, notcoordinated, 499
multirobot systems, collective swarms, 499
multirobot systems, intentionally coordination, 499
multirobot systems, strongly coordinated but distributed, 500
multirobot systems, stronglycoordinated but weaklycentralized, 500
multirobot systems, unaware, 499
multirobot systems, weaklycoordinated, 500

NASA, 33
natural landmarks, 361
natural terrain, 426
navigation, associative methods, 369
navigation, relational methods, 364
Navigator, 341, 342, 346

negative obstacle, 300
Nested Hierarchical Controller, 339, 340, 342, 346
networked robots, 489
new term problem, 455
niche targetability, 67
nonholonomic, 231

object recognition, 72
octree, 395
OODA, 78
open loop control, 26
open-world assumption, 47, 134, 322, 333
operational architecture, 66
operational architecture, Autonomous Control Levels(ACL), 78, 81
operational architecture, canonical, 68
operational architecture, Levels of Automation (LOA), 76, 80, 81
operational architecture, Levels of Initiative, 80
operator, 324
operator control unit, 519
optic flow, 158, 159
orthomosaic, 429, 430
overfitting, 451

path planning, 359
path relaxation, 393, 411
percept, 138, 146, 148
perception, guide, 171
perception, releaser, 171
perceptron, 457
perceptual distinguishability, 369
perceptual schema, 144
perceptual stability, 369
performance element, 474
persona, 74
physically situated, 6

piano mover's problem, 408, 412
Pilot, 341, 346
point cloud, 304–307
polar plot, 206, 294
pose, 419, 421–425, 429, 439
potential fields, 188
potential fields, attractive, 191
potential fields, harmonic functions, 202
potential fields, navigation templates, 202
potential fields, perpendicular, 190
potential fields, random, 202
potential fields, repulsive, 191
potential fields, tangential, 191
potential fields, uniform, 190
preconditions, 322, 325
predicates, 326
primitives, 68
primitives, perceptual ability, 69
primitives, planning horizon, 69
primitives, time-scale, 70
primitives, world model, 70
problem generation, 474
procedural cohesion, 201
professional ethics, 587
proprioception, 112, 275
proxemics, 518, 535, 537

Q-learning, 461
Q-learning, online learning, 466
Q-learning, rewards matrix, 464
quadtrees, 395
QualNav, 369

radial depth map, 293
rana computatrix, 137, 140, 141, 143, 146, 148
range image, 292
range segmentation, 302
rapidly exploring random trees (RRT), 409
reactive planning, 343
Real-Time Control System (RCS), 70, 90
reasoning, 358
recognition, 161, 162, 179
reconfigurable robots, 489
rectified images, 290
reference trajectory, 238, 240
reflex, 193
regular grid, 394–396, 401, 408
reinforcement learning, policy, 460
releaser, 165, 167–169, 171, 172, 177, 178
remote control, 109
remote presence, 12, 105, 515

remotely operated vehicle, 10, 12
replanning, continuous, 406
replanning, event-driven, 406
RGB-D, 15
RGB-D camera, 304
RHex, 5
robustness, 67
running, 237

safe human-robot ratio, 119, 538
Saphira, 336
scale-invariant feature transform(SIFT), 286, 429
schema, 143
schema class, 143
schema instantiation (SI), 143
scripts, 220
selective attention, 156
semi-autonomy, 109
sensemaking, 523
sensor, 108, 253
sensor networks, 489
Sequencer, 343
SFX, 369, 370, 372
Shakey, 31, 32
shared control, 76
simulator sickness, 117
simultaneous localization andmapping (SLAM), 425
simultaneous localization andmapping, loop closure, 425
simultaneous localization and mapping, Rao-Blackwellized Filtering, 425
situational awareness, 519, 522, 523
Skill Manager, 344
skills, 220, 343
Sliding, 235
social intelligence, 74
social interactions, 110
societal rules, 501
spatial memory, 358, 360, 361, 379
specular reflection, 296–298, 303, 304
speech recognition, 531
speech synthesis, 531
SRI, 31
state augmentation, 424
static stability, 240
statically balanced., 240
steering, 231
steering, Ackerman, 233
steering, omnidirectional wheels, 233
steering, polymorphic, 234
steering, skid, 233
steering, synchro-drive, 232
stereo pair, 288

stereopsis, 288
stigmergy, 495
stimulus-response, 141
Strips, 324
structural models, 159
sub-script, 220
subgoal obsession, 402, 404, 406, 411
substitution myth, 53, 57
subsumption, 204, 343
subsumption, inhibition, 209
subsumption, layers of competence, 205
subsumption, suppression, 209
support polygon, 240, 241
surface reconstruction, 307
swarm behaviors, 74
swarm robots, 485, 499
symbol grounding, 72, 123, 333, 334, 534
symbol grounding, physical, 333
systems architecture, 66
systems architecture, HybridDeliberative/Reactive, 93

task allocation, 486
task interference, 487
task rehearsal, 95, 96, 108
taskable agent, 105, 515
teach pendant, 27
team, 513, 517
team, ad hoc, 513
technical architecture, 66
telefactor, 22
telekinesis problem, 117
telemanipulator, 22
teleoperation, 104
teleoperation, communication link, 108
teleoperation, display, 105, 108
teleoperation, local, 105
teleoperation, remote, 107
teleoperation task characteristics, 122
teleoperator, 22
telepresence, 123
termination condition, 404
terrain identification, 428
Three Laws of Responsible Robotics, 592
Three Laws of Robotics, 5, 591
three mobility regimes, 433
time heuristic, 118
time to contact, 158
tool, 20
topological navigation, 359, 360
trade spaces, 55
trade spaces, fitness, 55

trade spaces, impact, 56
trade spaces, perspectives, 56
trade spaces, plans, 55
trade spaces, responsibility, 56
traded control, 76
transition table, 372
traversability, 434
traversability, accessibility elements, 435
traversability, severity of obstacles, 434
traversability, surface properties, 435
traversability, tortuosity, 434
traversability, verticality, 434
trust, 518, 540, 541

UAV, fixed-wing, 9
UAV, micro, 9
UAV, rotor-craft, 9
UGV size, 8
unattended ground sensors, 8
Uncanny Valley, 513, 534, 552
universal usability, 519
unmanned aerial system, 8
unmanned aerial vehicles (UAV), 8
unmanned ground vehicles (UGV), 8
unmanned marine vehicles, 10
unmanned surface vehicle, 10
unmanned underwater vehicle, 10
unmanned vehicles, 8
update rate, 344
user interface, diagnostic, 518
user interface, explicative, 519
user interface, operational, 519
user interface, Ten Minute Rule, 517

Van der Pol equation, 244
vector, 188
vector summation, 187, 188, 196, 199, 223
virtual gait, 243
virtual reality, 123
virtual sensor, 338, 339
virtue ethics, 590
visual homing, 369
Voronoi edge, 393

waldo, 23
waypoints, 388
Wizard of Oz (WOZ), 543
world model, 46, 326

zero moment point, 242

Intelligent Robotics and Autonomous Agents

Edited by Ronald C. Arkin

Dorigo, Marco, and Marco Colombetti, *Robot Shaping: An Experiment in Behavior Engineering*

Arkin, Ronald C., *Behavior-Based Robotics*

Stone, Peter, *Layered Learning in Multiagent Systems: A Winning Approach to Robotic Soccer*

Wooldridge, Michael, *Reasoning About Rational Agents*

Murphy, Robin R., *Introduction to AI Robotics*

Mason, Matthew T., *Mechanics of Robotic Manipulation*

Kraus, Sarit, *Strategic Negotiation in Multiagent Environments*

Nolfi, Stefano, and Dario Floreano, *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*

Siegwart, Roland, and Illah R. Nourbakhsh, *Introduction to Autonomous Mobile Robots*

Breazeal, Cynthia L., *Designing Sociable Robots*

Bekey, George A., *Autonomous Robots: From Biological Inspiration to Implementation and Control*

Choset, Howie, Kevin M. Lynch, Seth Hutchinson, George Kantor, Wolfram Burgard, Lydia E. Kavraki, and Sebastian Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*

Thrun, Sebastian, Wolfram Burgard, and Dieter Fox, *Probabilistic Robotics*

Mataric, Maja J., *The Robotics Primer*

Wellman, Michael P., Amy Greenwald, and Peter Stone, *Autonomous Bidding Agents: Strategies and Lessons from the Trading Agent Competition*

Floreano, Dario and Claudio Mattiussi, *Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies*

Sterling, Leon S. and Kuldar Taveter, *The Art of Agent-Oriented Modeling*

- Stoy, Kasper, David Brandt, and David J. Christensen, *An Introduction to Self-Reconfigurable Robots*
- Lin, Patrick, Keith Abney, and George A. Bekey, editors, *Robot Ethics: The Ethical and Social Implications of Robotics*
- Weiss, Gerhard, editor, *Multiagent Systems*, second edition
- Vargas, Patricia A., Ezequiel A. Di Paolo, Inman Harvey, and Phil Husbands, editors, *The Horizons of Evolutionary Robotics*
- Murphy, Robin R., *Disaster Robotics*
- Cangelosi, Angelo and Matthew Schlesinger, *Developmental Robotics: From Babies to Robots*
- Everett, H. R., *Unmanned Systems of World Wars I and II*
- Sitti, Metin, *Mobile Microrobotics*
- Murphy, Robin R., *Introduction to AI Robotics*, second edition