Probabilistic ROBOTICS

SEBASTIAN THRUN
WOLFRAM BURGARD
DIETER FOX

TESLA EXHIBIT 1011, Page cover

Probabilistic
Robotics

Probabilistic Robotics

Sebastian Thrun
Wolfram Burgard
Dieter Fox

The MIT Press
Cambridge, Massachusetts
London, England

© 2006 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.

MIT Press books may be purchased at special quantity discounts for business or sales promotional use. For information, please email special_sales@mitpress.mit.edu or write to Special Sales Department, The MIT Press, 55 Hayward Street, Cambridge, MA 02142.

Typeset in 10/13 Lucida Bright by the authors using LaTeX 2_ε.
Printed and bound in the United States of America.

Library of Congress Cataloging-in-Publication Data

Thrun, Sebastian, 1967–
Probabilistic robotics / Sebastian Thrun, Wolfram Burgard, Dieter Fox.
p. cm. – (Intelligent robotics and autonomous agents series)
Includes bibliographical references and index.
ISBN-13: 978-0-262-20162-9 (alk. paper)
1. Robotics. 2. Probabilities. I. Burgard, Wolfram. II. Fox, Dieter. III. Title. IV. Intelligent robotics and autonomous agents.
TJ211.T575 2005
629.8′92–dc22

2005043346

10 9 8 7 6 5 6 3

Brief Contents

I Basics 1

1 Introduction 3

2 Recursive State Estimation 13

3 Gaussian Filters 39

4 Nonparametric Filters 85

5 Robot Motion 117

6 Robot Perception 149

II Localization 189

7 Mobile Robot Localization: Markov and Gaussian 191

8 Mobile Robot Localization: Grid And Monte Carlo 237

III Mapping 279

9 Occupancy Grid Mapping 281

10 Simultaneous Localization and Mapping 309

11 The GraphSLAM Algorithm 337

12 The Sparse Extended Information Filter 385

13 The FastSLAM Algorithm 437

IV Planning and Control 485

14 Markov Decision Processes 487

15 Partially Observable Markov Decision Processes 513

16 *Approximate POMDP Techniques* 547
17 *Exploration* 569

Contents

Preface **xvii**

Acknowledgments **xix**

I Basics 1

1 *Introduction* 3

1.1 Uncertainty in Robotics 3
1.2 Probabilistic Robotics 4
1.3 Implications 9
1.4 Road Map 10
1.5 Teaching Probabilistic Robotics 11
1.6 Bibliographical Remarks 11

2 *Recursive State Estimation* 13

2.1 Introduction 13
2.2 Basic Concepts in Probability 14
2.3 Robot Environment Interaction 19
2.3.1 State 20
2.3.2 Environment Interaction 22
2.3.3 Probabilistic Generative Laws 24
2.3.4 Belief Distributions 25
2.4 Bayes Filters 26
2.4.1 The Bayes Filter Algorithm 26
2.4.2 Example 28
2.4.3 Mathematical Derivation of the Bayes Filter 31
2.4.4 The Markov Assumption 33

2.5 Representation and Computation 34
2.6 Summary 35
2.7 Bibliographical Remarks 36
2.8 Exercises 36

3 *Gaussian Filters* 39

3.1 Introduction 39
3.2 The Kalman Filter 40
3.2.1 Linear Gaussian Systems 40
3.2.2 The Kalman Filter Algorithm 43
3.2.3 Illustration 44
3.2.4 Mathematical Derivation of the KF 45
3.3 The Extended Kalman Filter 54
3.3.1 Why Linearize? 54
3.3.2 Linearization Via Taylor Expansion 56
3.3.3 The EKF Algorithm 59
3.3.4 Mathematical Derivation of the EKF 59
3.3.5 Practical Considerations 61
3.4 The Unscented Kalman Filter 65
3.4.1 Linearization Via the Unscented Transform 65
3.4.2 The UKF Algorithm 67
3.5 The Information Filter 71
3.5.1 Canonical Parameterization 71
3.5.2 The Information Filter Algorithm 73
3.5.3 Mathematical Derivation of the Information Filter 74
3.5.4 The Extended Information Filter Algorithm 75
3.5.5 Mathematical Derivation of the Extended Information Filter 76
3.5.6 Practical Considerations 77
3.6 Summary 79
3.7 Bibliographical Remarks 81
3.8 Exercises 81

4 *Nonparametric Filters* 85

4.1 The Histogram Filter 86
4.1.1 The Discrete Bayes Filter Algorithm 86
4.1.2 Continuous State 87
4.1.3 Mathematical Derivation of the Histogram Approximation 89

4.1.4 Decomposition Techniques 92
4.2 Binary Bayes Filters with Static State 94
4.3 The Particle Filter 96
4.3.1 Basic Algorithm 96
4.3.2 Importance Sampling 100
4.3.3 Mathematical Derivation of the PF 103
4.3.4 Practical Considerations and Properties of Particle Filters 104
4.4 Summary 113
4.5 Bibliographical Remarks 114
4.6 Exercises 115

5 *Robot Motion* 117

5.1 Introduction 117
5.2 Preliminaries 118
5.2.1 Kinematic Configuration 118
5.2.2 Probabilistic Kinematics 119
5.3 Velocity Motion Model 121
5.3.1 Closed Form Calculation 121
5.3.2 Sampling Algorithm 122
5.3.3 Mathematical Derivation of the Velocity Motion Model 125
5.4 Odometry Motion Model 132
5.4.1 Closed Form Calculation 133
5.4.2 Sampling Algorithm 137
5.4.3 Mathematical Derivation of the Odometry Motion Model 137
5.5 Motion and Maps 140
5.6 Summary 143
5.7 Bibliographical Remarks 145
5.8 Exercises 145

6 *Robot Perception* 149

6.1 Introduction 149
6.2 Maps 152
6.3 Beam Models of Range Finders 153
6.3.1 The Basic Measurement Algorithm 153
6.3.2 Adjusting the Intrinsic Model Parameters 158
6.3.3 Mathematical Derivation of the Beam Model 162

6.3.4 Practical Considerations 167
6.3.5 Limitations of the Beam Model 168
6.4 Likelihood Fields for Range Finders 169
6.4.1 Basic Algorithm 169
6.4.2 Extensions 172
6.5 Correlation-Based Measurement Models 174
6.6 Feature-Based Measurement Models 176
6.6.1 Feature Extraction 176
6.6.2 Landmark Measurements 177
6.6.3 Sensor Model with Known Correspondence 178
6.6.4 Sampling Poses 179
6.6.5 Further Considerations 180
6.7 Practical Considerations 182
6.8 Summary 183
6.9 Bibliographical Remarks 184
6.10 Exercises 185

II Localization 189

7 *Mobile Robot Localization: Markov and Gaussian* 191
7.1 A Taxonomy of Localization Problems 193
7.2 Markov Localization 197
7.3 Illustration of Markov Localization 200
7.4 EKF Localization 201
7.4.1 Illustration 201
7.4.2 The EKF Localization Algorithm 203
7.4.3 Mathematical Derivation of EKF Localization 205
7.4.4 Physical Implementation 210
7.5 Estimating Correspondences 215
7.5.1 EKF Localization with Unknown Correspondences 215
7.5.2 Mathematical Derivation of the ML Data Association 216
7.6 Multi-Hypothesis Tracking 218
7.7 UKF Localization 220
7.7.1 Mathematical Derivation of UKF Localization 220
7.7.2 Illustration 223
7.8 Practical Considerations 229

7.9 Summary 232
7.10 Bibliographical Remarks 233
7.11 Exercises 234

8 *Mobile Robot Localization: Grid And Monte Carlo* 237

8.1 Introduction 237
8.2 Grid Localization 238
8.2.1 Basic Algorithm 238
8.2.2 Grid Resolutions 239
8.2.3 Computational Considerations 243
8.2.4 Illustration 245
8.3 Monte Carlo Localization 250
8.3.1 Illustration 250
8.3.2 The MCL Algorithm 252
8.3.3 Physical Implementations 253
8.3.4 Properties of MCL 253
8.3.5 Random Particle MCL: Recovery from Failures 256
8.3.6 Modifying the Proposal Distribution 261
8.3.7 KLD-Sampling: Adapting the Size of Sample Sets 263
8.4 Localization in Dynamic Environments 267
8.5 Practical Considerations 273
8.6 Summary 274
8.7 Bibliographical Remarks 275
8.8 Exercises 276

III Mapping 279

9 *Occupancy Grid Mapping* 281

9.1 Introduction 281
9.2 The Occupancy Grid Mapping Algorithm 284
9.2.1 Multi-Sensor Fusion 293
9.3 Learning Inverse Measurement Models 294
9.3.1 Inverting the Measurement Model 294
9.3.2 Sampling from the Forward Model 295
9.3.3 The Error Function 296
9.3.4 Examples and Further Considerations 298
9.4 Maximum A Posteriori Occupancy Mapping 299
9.4.1 The Case for Maintaining Dependencies 299

9.4.2 Occupancy Grid Mapping with Forward Models 301
9.5 Summary 304
9.6 Bibliographical Remarks 305
9.7 Exercises 307

10 *Simultaneous Localization and Mapping* 309

10.1 Introduction 309
10.2 SLAM with Extended Kalman Filters 312
10.2.1 Setup and Assumptions 312
10.2.2 SLAM with Known Correspondence 313
10.2.3 Mathematical Derivation of EKF SLAM 317
10.3 EKF SLAM with Unknown Correspondences 323
10.3.1 The General EKF SLAM Algorithm 323
10.3.2 Examples 324
10.3.3 Feature Selection and Map Management 328
10.4 Summary 330
10.5 Bibliographical Remarks 332
10.6 Exercises 334

11 *The GraphSLAM Algorithm* 337

11.1 Introduction 337
11.2 Intuitive Description 340
11.2.1 Building Up the Graph 340
11.2.2 Inference 343
11.3 The GraphSLAM Algorithm 346
11.4 Mathematical Derivation of GraphSLAM 353
11.4.1 The Full SLAM Posterior 353
11.4.2 The Negative Log Posterior 354
11.4.3 Taylor Expansion 355
11.4.4 Constructing the Information Form 357
11.4.5 Reducing the Information Form 360
11.4.6 Recovering the Path and the Map 361
11.5 Data Association in GraphSLAM 362
11.5.1 The GraphSLAM Algorithm with Unknown Correspondence 363
11.5.2 Mathematical Derivation of the Correspondence Test 366
11.6 Efficiency Consideration 368
11.7 Empirical Implementation 370

11.8 Alternative Optimization Techniques 376
11.9 Summary 379
11.10 Bibliographical Remarks 381
11.11 Exercises 382

12 *The Sparse Extended Information Filter* 385

12.1 Introduction 385
12.2 Intuitive Description 388
12.3 The SEIF SLAM Algorithm 391
12.4 Mathematical Derivation of the SEIF 395
12.4.1 Motion Update 395
12.4.2 Measurement Updates 398
12.5 Sparsification 398
12.5.1 General Idea 398
12.5.2 Sparsification in SEIFs 400
12.5.3 Mathematical Derivation of the Sparsification 401
12.6 Amortized Approximate Map Recovery 402
12.7 How Sparse Should SEIFs Be? 405
12.8 Incremental Data Association 409
12.8.1 Computing Incremental Data Association Probabilities 409
12.8.2 Practical Considerations 411
12.9 Branch-and-Bound Data Association 415
12.9.1 Recursive Search 416
12.9.2 Computing Arbitrary Data Association Probabilities 416
12.9.3 Equivalence Constraints 419
12.10 Practical Considerations 420
12.11 Multi-Robot SLAM 424
12.11.1 Integrating Maps 424
12.11.2 Mathematical Derivation of Map Integration 427
12.11.3 Establishing Correspondence 429
12.11.4 Example 429
12.12 Summary 432
12.13 Bibliographical Remarks 434
12.14 Exercises 435

13 *The FastSLAM Algorithm* 437

13.1 The Basic Algorithm 439

13.2 Factoring the SLAM Posterior 439
13.2.1 Mathematical Derivation of the Factored SLAM Posterior 442
13.3 FastSLAM with Known Data Association 444
13.4 Improving the Proposal Distribution 451
13.4.1 Extending the Path Posterior by Sampling a New Pose 451
13.4.2 Updating the Observed Feature Estimate 454
13.4.3 Calculating Importance Factors 455
13.5 Unknown Data Association 457
13.6 Map Management 459
13.7 The FastSLAM Algorithms 460
13.8 Efficient Implementation 460
13.9 FastSLAM for Feature-Based Maps 468
13.9.1 Empirical Insights 468
13.9.2 Loop Closure 471
13.10 Grid-based FastSLAM 474
13.10.1 The Algorithm 474
13.10.2 Empirical Insights 475
13.11 Summary 479
13.12 Bibliographical Remarks 481
13.13 Exercises 482

IV Planning and Control 485

14 *Markov Decision Processes* 487
14.1 Motivation 487
14.2 Uncertainty in Action Selection 490
14.3 Value Iteration 495
14.3.1 Goals and Payoff 495
14.3.2 Finding Optimal Control Policies for the Fully Observable Case 499
14.3.3 Computing the Value Function 501
14.4 Application to Robot Control 503
14.5 Summary 507
14.6 Bibliographical Remarks 509
14.7 Exercises 510

15 *Partially Observable Markov Decision Processes* 513
15.1 Motivation 513
15.2 An Illustrative Example 515
15.2.1 Setup 515
15.2.2 Control Choice 516
15.2.3 Sensing 519
15.2.4 Prediction 523
15.2.5 Deep Horizons and Pruning 526
15.3 The Finite World POMDP Algorithm 527
15.4 Mathematical Derivation of POMDPs 531
15.4.1 Value Iteration in Belief Space 531
15.4.2 Value Function Representation 532
15.4.3 Calculating the Value Function 533
15.5 Practical Considerations 536
15.6 Summary 541
15.7 Bibliographical Remarks 542
15.8 Exercises 544

16 *Approximate POMDP Techniques* 547
16.1 Motivation 547
16.2 QMDPs 549
16.3 Augmented Markov Decision Processes 550
16.3.1 The Augmented State Space 550
16.3.2 The AMDP Algorithm 551
16.3.3 Mathematical Derivation of AMDPs 553
16.3.4 Application to Mobile Robot Navigation 556
16.4 Monte Carlo POMDPs 559
16.4.1 Using Particle Sets 559
16.4.2 The MC-POMDP Algorithm 559
16.4.3 Mathematical Derivation of MC-POMDPs 562
16.4.4 Practical Considerations 563
16.5 Summary 565
16.6 Bibliographical Remarks 566
16.7 Exercises 566

17 *Exploration* 569
17.1 Introduction 569
17.2 Basic Exploration Algorithms 571
17.2.1 Information Gain 571

17.2.2 Greedy Techniques 572
17.2.3 Monte Carlo Exploration 573
17.2.4 Multi-Step Techniques 575
17.3 Active Localization 575
17.4 Exploration for Learning Occupancy Grid Maps 580
17.4.1 Computing Information Gain 580
17.4.2 Propagating Gain 585
17.4.3 Extension to Multi-Robot Systems 587
17.5 Exploration for SLAM 593
17.5.1 Entropy Decomposition in SLAM 593
17.5.2 Exploration in FastSLAM 594
17.5.3 Empirical Characterization 598
17.6 Summary 600
17.7 Bibliographical Remarks 602
17.8 Exercises 604

Bibliography **607**

Index **639**

Preface

This book provides a comprehensive introduction into the emerging field of probabilistic robotics. Probabilistic robotics is a subfield of robotics concerned with perception and control. It relies on statistical techniques for representing information and making decisions. By doing so, it accommodates the uncertainty that arises in most contemporary robotics applications. In recent years, probabilistic techniques have become one of the dominant paradigms for algorithm design in robotics. This monograph provides a first comprehensive introduction into some of the major techniques in this field.

This book has a strong focus on algorithms. All algorithms in this book are based on a single overarching mathematical foundation: Bayes rule, and its temporal extension known as Bayes filters. This unifying mathematical framework is the core commonality of probabilistic algorithms.

In writing this book, we have tried to be as complete as possible with regards to technical detail. Each chapter describes one or more major algorithms. For each algorithm, we provide the following four things: (1) an example implementation in pseudo code; (2) a complete mathematical derivation from first principles that makes the various assumptions behind each algorithm explicit; (3) empirical results insofar as they further the understanding of the algorithms presented in the book; and (4) a detailed discussion of the strengths and weaknesses of each algorithm—from a practitioner's perspective. Developing all this for many different algorithms proved to be a laborious task. The result might at times be a bit difficult to digest for the casual reader—although skipping the mathematical derivation sections is always an option! We hope that a careful reader emerges with a much deeper level of understanding than any superficial, non-mathematical exposition of this topic would have been able to convey.

This book is the result of more than a decade of research by us, the authors, our students, and many of our colleagues in the field. We began writing it in 1999, hoping that it would take not much more than a few months to complete this book. Five years have passed, and almost nothing from the original draft has survived. Through working on this book, we have learned much more about information and decision theory than we thought we ever would. We are happy to report that much of what we learned has made it into this book.

This monograph is written for students, researchers, and practitioners in robotics. We believe everybody building robots has to develop software. Hence the material in this book should be relevant to every roboticist. It should also be of interest to applied statisticians, and people concerned with real-world sensor data outside the realm of robotics. To serve a wide range of readers with varying technical backgrounds, we have attempted to make this book as self-contained as possible. Some prior knowledge of linear algebra and basic probability and statistics will be helpful, but we have included a primer for the basic laws of probability, and avoided the use of advanced mathematical techniques throughout this text.

This book is also written for classroom use. Each chapter offers a number of exercises and suggests hands-on projects. When used in the classroom, each chapter should be covered in one or two lectures. Chapters should be skipped or reordered quite arbitrarily; in fact, in our own teaching we usually start right in the middle of the book, with Chapter 7. We recommend that the study of the book be accompanied by practical, hands-on experimentation as directed by the exercises at the end of each chapter. Nothing more important in robotics than doing it yourself!

Despite our very best efforts, we believe there will still be technical errors left in this book. Many of these errors have been corrected in this third printing of the book. We continue to post corrections on the book's Web site, along with other materials relevant to this book:

`www.probabilistic-robotics.org`

We hope you enjoy this book!

Sebastian Thrun
Wolfram Burgard
Dieter Fox

TESLA EXHIBIT 1011, Page xviii

Acknowledgments

This book would not have been possible without the help and support from so many friends, family members, students, and colleagues in the field. We cannot possibly list all of them here.

Much of the material in this book is the result of collaborations with our current and past students and post-docs. We specifically would like to acknowledge Rahul Biswas, Matthew Deans, Frank Dellaert, James Diebel, Brian Gerkey, Dirk Hähnel, Johnathan Ko, Cody Kwok, John Langford, Lin Liao, David Lieb, Benson Limketkai, Michael Littman, Yufeng Liu, Andrew Lookingbill, Dimitris Margaritis, Michael Montemerlo, Mark Moors, Mark Paskin, Joelle Pineau, Charles Rosenberg, Nicholas Roy, Aaron Shon, Jamie Schulte, Dirk Schulz, David Stavens, Cyrill Stachniss, and Chieh-Chih Wang, along with all other past and present members of our labs. Greg Armstrong, Grinnell More, Tyson Sawyer, and Walter Steiner were instrumental in keeping our robots running over the years.

Much of our research was conducted while we were with Carnegie Mellon University in Pittsburgh, PA, and we thank our former colleagues and friends at CMU for many inspirational discussions. We also would like to express our gratitude to Armin Cremers from the University of Bonn who brought the three of us together in his research group and who set the seed for our wonderful collaboration.

We are indebted to numerous colleagues whose comments and insights were instrumental during the course of our research. We would specifically like to thank Gary Bradski, Howie Choset, Henrik Christensen, Hugh Durrant-Whyte, Nando de Freitas, Zoubin Gharamani, Geoffrey Gordon, Steffen Gutmann, Andrew Howards, Leslie Kaelbling, Daphne Koller, Kurt Konolige Ben Kuipers, John Leonard, Tom Mitchell, Kevin Murphy, Eduardo Nebot, Paul Newman, Andrew Y. Ng, Reid Simmons, Satinder Singh, Gau-

rav Sukhatme, Juan Tardós, Ben Wegbreit, and Alex Zelinsky for their feedback over the years.

Anita Araneda, Gal Elidan, Udo Frese, Gabe Hoffmann, John Leonard, Benson Limketkai, Rudolph van der Merwe, Anna Petrovskaya, Bob Wang, and Stefan Williams gave us extensive comments on earlier drafts of this book, which we gratefully acknowledge. Chris Manning kindly provided the Latex macros for this book, and Bob Prior was instrumental in getting the book published.

A number of agencies and corporations have made this work possible through generous financial contributions and technical advice. We are particularly grateful for DARPA's support under a number of programs (TMR, MARS, LAGR, SDR, MICA, and CoABS). We also acknowledge funding from the National Science Foundation, via its CAREER, ITR, and various CISE grant programs; from the German Research Foundation; and from the European Commission. Generous financial support was also provided by a number of corporate sponsors and individual donors: Android, Bosch, DaimlerChrysler, Intel, Google, Microsoft, Mohr Davidow Ventures, Samsung, and Volkswagen of America. We specifically thank John Blitch, Doug Gage, Sharon Heise, James Hendler, Larry Jackel, Alex Krott, Wendell Sykes, and Ed van Reuth for providing us with ample challenges and guidance throughout the years. Naturally, the views and conclusions contained in this document are those of the authors, and should not be interpreted as necessarily representing policies or endorsements of any of our sponsors.

We owe our most important acknowledgments to our families, whose love and dedication carried us through this immense project. We specifically thank Petra Dierkes-Thrun, Anja Gross-Burgard and Carsten Burgard, and Luz, Sofia and Karla Fox for their love and support.

PART I

Basics

1 *Introduction*

1.1 Uncertainty in Robotics

Robotics is the science of perceiving and manipulating the physical world through computer-controlled devices. Examples of successful robotic systems include mobile platforms for planetary exploration, industrial robotics arms in assembly lines, cars that travel by themselves, and manipulators that assist surgeons. Robotics systems are situated in the physical world, perceive information on their environments through sensors, and manipulate through physical forces.

While much of robotics is still in its infancy, the idea of "intelligent" manipulating devices has an enormous potential to change society. Wouldn't it be great if all our cars were able to safely steer themselves, making car accidents a notion of the past? Wouldn't it be great if robots, and not people, would clean up nuclear disaster sites like Chernobyl? Wouldn't it be great if our homes were populated by intelligent assistants that take care of all domestic repair and maintenance tasks?

To do these tasks, robots have to be able to accommodate the enormous uncertainty that exists in the physical world. There is a number of factors that contribute to a robot's uncertainty.

First and foremost, *robot environments* are inherently unpredictable. While the degree of uncertainty in well-structured environments such as assembly lines is small, environments such as highways and private homes are highly dynamic and in many ways highly unpredictable. The uncertainty is particularly high for robots operating in the proximity of people.

Sensors are limited in what they can perceive. Limitations arise from several factors. The range and resolution of a sensor is subject to physical limitations. For example, cameras cannot see through walls, and the spatial res-

olution of a camera image is limited. Sensors are also subject to noise, which perturbs sensor measurements in unpredictable ways and hence limits the information that can be extracted. And finally, sensors can break. Detecting a faulty sensor can be extremely difficult.

Robot actuation involves motors that are, at least to some extent, unpredictable. Uncertainty arises from effects like control noise, wear-and-tear, and mechanical failure. Some actuators, such as heavy-duty industrial robot arms, are quite accurate and reliable. Others, like low-cost mobile robots, can be extremely flaky.

Some uncertainty is caused by the robot's software. All *internal models* of the world are approximate. Models are abstractions of the real world. As such, they only partially model the underlying physical processes of the robot and its environment. Model errors are a source of uncertainty that has often been ignored in robotics, despite the fact that most robotic models used in state-of-the-art robotics systems are rather crude.

Uncertainty is further created through *algorithmic approximations*. Robots are real-time systems. This limits the amount of computation that can be carried out. Many popular algorithms are approximate, achieving timely response through sacrificing accuracy.

The level of uncertainty depends on the application domain. In some robotic applications, such as assembly lines, humans can cleverly engineer the system so that uncertainty is only a marginal factor. In contrast, robots operating in residential homes or on other planets will have to cope with substantial uncertainty. Such robots are forced to act even though neither their sensors, nor their internal models, will provide it with sufficient information to make the right decisions with absolute certainty. As robotics is now moving into the open world, the issue of uncertainty has become a major stumbling block for the design of capable robot systems. Managing uncertainty is possibly the most important step towards robust real-world robot systems.

Hence this book.

1.2 Probabilistic Robotics

This book provides a comprehensive overview of *probabilistic robotics*. Probabilistic robotics is a relatively new approach to robotics that pays tribute to the uncertainty in robot perception and action. The key idea in probabilistic robotics is to represent uncertainty explicitly using the calculus of probability

theory. Put differently, instead of relying on a single "best guess" as to what might be the case, probabilistic algorithms represent information by probability distributions over a whole space of guesses. By doing so, they can represent ambiguity and degree of belief in a mathematically sound way. Control choices can be made robust relative the uncertainty that remains, and probabilistic robotics can even actively chose to reduce their uncertainty when this appears to be the superior choice. Thus, probabilistic algorithms degrade gracefully in the face of uncertainty. As a result, they outperform alternative techniques in many real-world applications.

We shall illustrate probabilistic robotics with two motivating examples: one pertaining to robot perception, and another to planning and control.

MOBILE ROBOT LOCALIZATION

Our first example is *mobile robot localization*. Robot localization is the problem of estimating a robot's coordinates relative to an external reference frame. The robot is given a map of its environment, but to localize itself relative to this map it needs to consult its sensor data. Figure 1.1 illustrates such a situation. The environment is known to possess three indistinguishable doors. The task of the robot is to find out where it is, through sensing and motion.

This specific localization problem is known as *global localization*. In global localization, a robot is placed somewhere in a known environment and has to localize itself from scratch. The probabilistic paradigm represents the robot's momentary *belief* by a probability density function over the space of all locations. This is illustrated in diagram (a) in Figure 1.1. This diagram shows a uniform distribution over all locations. Now suppose the robot takes a first sensor measurement and observes that it is next to a door. Probabilistic techniques exploit this information to update the belief. The 'posterior' belief is shown in diagram (b) in Figure 1.1. It places an increased probability at places near doors, and lower probability near walls. Notice that this distribution possesses three peaks, each corresponding to one of the indistinguishable doors in the environment. Thus, by no means does the robot *know* where it is. Instead, it now has three, distinct hypotheses which are each equally plausible given the sensor data. We also note that the robot assigns positive probability to places *not* next to a door. This is the natural result of the inherent uncertainty in sensing: With a small, non-zero probability, the robot might have erred in its assessment of seeing a door. The ability to maintain low-probability hypotheses is essential for attaining robustness.

Now suppose the robot moves. Diagram (c) in Figure 1.1 shows the effect on a robot's belief. The belief has been shifted in the direction of motion. It also possesses a larger spread, which reflects the uncertainty that is intro-

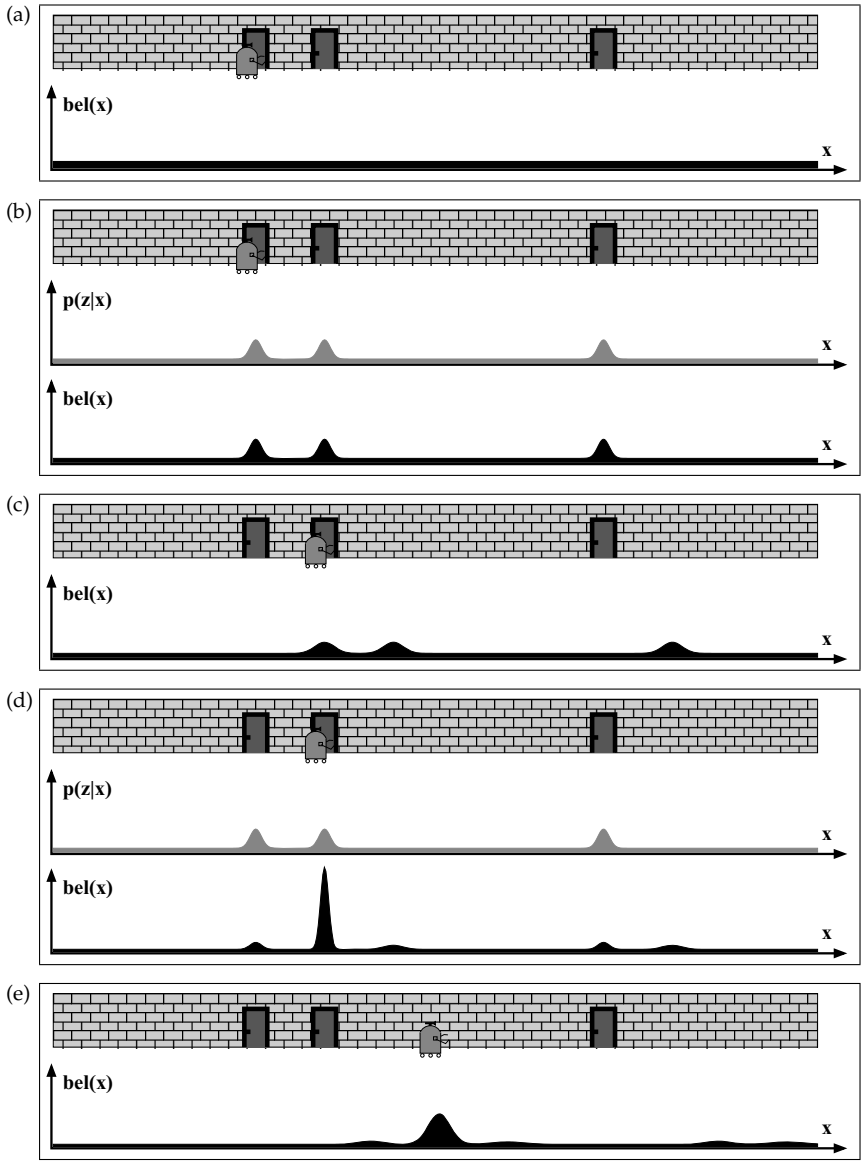
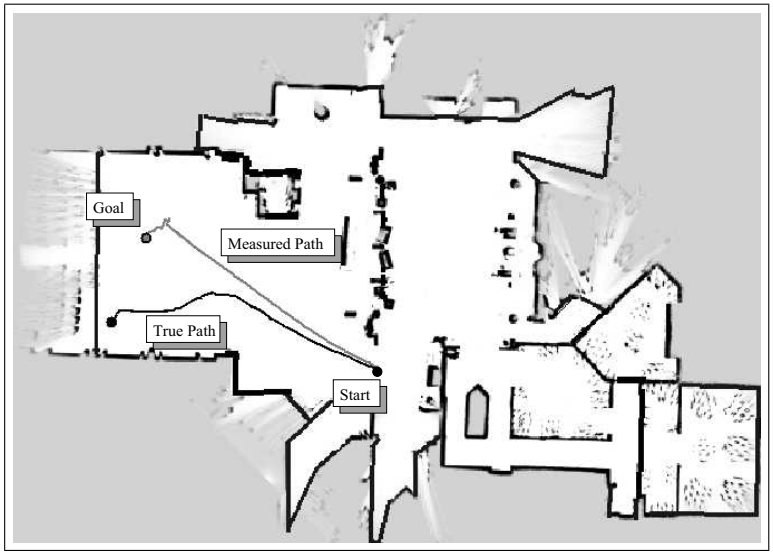


Figure 1.1 The basic idea of *Markov localization*: A mobile robot during global localization. Markov localization techniques will be investigated in Chapters 7 and 8.

(a)



(b)

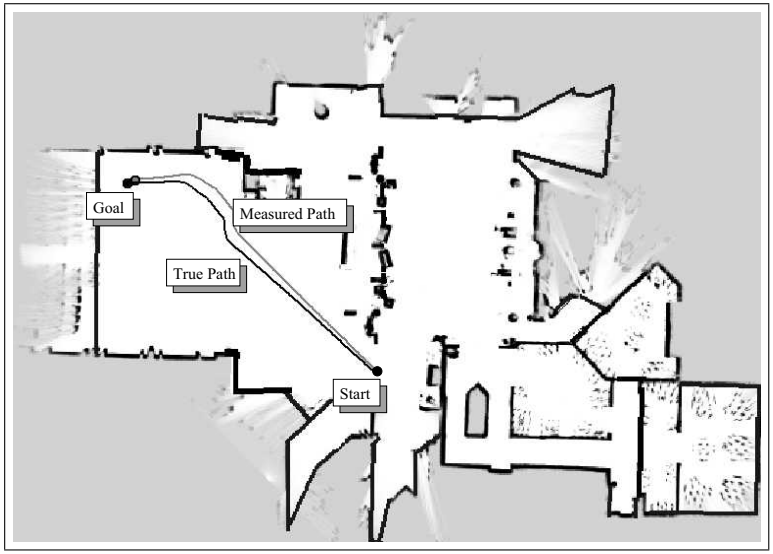


Figure 1.2 Top image: a robot navigating through open, featureless space may lose track of where it is. Bottom: This can be avoided by staying near known obstacles. These figures are results of an algorithm called *coastal navigation*, which will be discussed in Chapter 16. Images courtesy of Nicholas Roy, MIT.

duced by robot motion. Diagram (d) in Figure 1.1 depicts the belief after observing another door. This observation leads our algorithm to place most of the probability mass on a location near one of the doors, and the robot is now quite confident as to where it is. Finally, Diagram (e) shows a belief as the robot travels further down the corridor.

This example illustrates many aspects of the probabilistic paradigm. Stated probabilistically, the robot perception problem is a state estimation problem, and our localization example uses an algorithm known as *Bayes filter* for posterior estimation over the space of robot locations. The representation of information is a probability density function. The update of this function represents the information gained through sensor measurements, or the information lost through processes in the world that increase a robot's uncertainty.

BAYES FILTER

Our second example brings us into the realm of robotic planning and control. As just argued, probabilistic algorithms can compute a robot's momentary uncertainty. But they can also anticipate future uncertainty, and take such uncertainty into consideration when determining the right choice of control. One such algorithm is called *coastal navigation*. An example of coastal navigation is shown in Figure 1.2. This figure shows a 2-D map of an actual building. The top diagram compares an estimated path with an actual path: The divergence is the result of the uncertainty in robot motion that we just discussed. The interesting insight is: not all trajectories induce the same level of uncertainty. The path in Figure 1.2a leads through relatively open space, deprived of features that could help the robot to remain localized. Figure 1.2b shows an alternative path. This trajectory seeks a distinct corner, and then "hugs" the wall so as to stay localized. Not surprisingly, the uncertainty will be reduced for the latter path, hence chances of arriving at the goal location are actually higher.

COASTAL NAVIGATION

This example illustrates one of the many ways proper consideration of uncertainty affects the robot's controls. In our example, the anticipation of possible uncertainty along one trajectory makes the robot prefer a second, longer path, just so as to reduce the uncertainty. The new path is better, in the sense that the robot has a much higher chance of actually being at the goal when believing that it is. In fact, the second path is an example of active information gathering. The robot has, through its probabilistic consideration, determined that the best choice of action is to seek information along its path, in its pursuit to reach a target location. Probabilistic planning techniques anticipate uncertainty and can plan for information gathering, and probabilistic control techniques realize the results of such plans.

1.3 Implications

Probabilistic robotics seamlessly integrates models with sensor data, overcoming the limitations of both at the same time. These ideas are not just a matter of low-level control. They cut across all levels of robotic software, from the lowest to the highest.

In contrast with traditional programming techniques in robotics—such as model-based motion planning techniques or reactive behavior-based approaches—probabilistic approaches tend to be more robust in the face of sensor limitations and model limitations. This enables them to scale much better to complex real-world environments than previous paradigms, where uncertainty is of even greater importance. In fact, certain probabilistic algorithms are currently the only known working solutions to hard robotic estimation problems, such as the localization problem discussed a few pages ago, or the problem of building accurate maps of very large environments.

In comparison to traditional model-based robotic techniques, probabilistic algorithms have weaker requirements on the accuracy of the robot's models, thereby relieving the programmer from the insurmountable burden to come up with accurate models. Probabilistic algorithms have weaker requirements on the accuracy of robotic sensors than those made by many reactive techniques, whose sole control input is the momentary sensor input. Viewed probabilistically, the *robot learning problem* is a long-term estimation problem. Thus, probabilistic algorithms provide a sound methodology for many flavors of robot learning.

However, these advantages come at a price. The two most frequently cited limitations of probabilistic algorithms are *computational complexity*, and a *need to approximate*. Probabilistic algorithms are inherently less efficient than their non-probabilistic counterparts. This is due to the fact that they consider entire probability densities instead of a single guess. The need to approximate arises from the fact that most robot worlds are continuous. Computing exact posterior distributions tends to be computationally intractable. Sometimes, one is fortunate in that the uncertainty can be approximated tightly with a compact parametric model (e.g., Gaussians). In other cases, such approximations are too crude to be of use, and more complicated representations must be employed.

Recent developments in computer hardware has made an unprecedented number of FLOPS available at bargain prices. This development has certainly aided the field of probabilistic robotics. Further, recent research has successfully increased the computational efficiency of probabilistic algo-

rithms, for a range of hard robotics problems—many of which are described in depth in this book. Nevertheless, computational challenges remain. We shall revisit this discussion at numerous places, where we investigate the strengths and weaknesses of specific probabilistic solutions.

1.4 Road Map

This book is organized in four major parts.

- Chapters 2 through 4 introduce the basic mathematical framework that underlies all of the algorithms described in this book, along with key algorithms. These chapters are the mathematical foundation of this book.

- Chapters 5 and 6 present probabilistic models of mobile robots. In many ways, these chapters are the probabilistic generalization of classical robotics models. They form the robotic foundation for the material that follows.

- The mobile robot localization problem is discussed in Chapters 7 and 8. These chapters combine the basic estimation algorithms with the probabilistic models discussed in the previous two chapters.

- Chapters 9 through 13 discuss the much richer problem of robotic mapping. As before, they are all based on the algorithms discussed in the foundational chapters, but many of them utilize tricks to accommodate the enormous complexity of this problem.

- Problems of probabilistic planning and control are discussed in Chapters 14 through 17. Here we begin by introducing a number of fundamental techniques, and then branch into practical algorithms for controlling a robot probabilistically. The final chapter, Chapter 17, discusses the problem of robot exploration from a probabilistic perspective.

The book is best read in order, from the beginning to the end. However, we have attempted to make each individual chapter self-explanatory. Frequent sections called *"Mathematical Derivation of . . . "* can safely be skipped on first reading without compromising the coherence of the overall material in this book.

TESLA EXHIBIT 1011, Page 10

1.5 Teaching Probabilistic Robotics

When used in the classroom, we do *not* recommend to teach the chapters in order—unless the students have an unusually strong appreciation of abstract mathematical concepts. Particle filters are easier to teach than Gaussian filters, and students tend to get more excited by mobile robot localization problems than abstract filter algorithms. In our own teachings, we usually begin with Chapter 2, and move directly to Chapters 7 and 8. While teaching localization, we go back to the material in Chapters 3 through 6 as needed. We also teach Chapter 14 early, to expose students to the problems related to planning and control early on in a course.

As a teacher, feel free to use slides and animations from the book's Web site

`www.probabilistic-robotics.org`

to illustrate the various algorithms in this book. And feel free to send us, the authors, pointers to your class Web sites and any material that could help others in teaching Probabilistic Robotics.

The material in this book is best taught with hands-on implementation assignments. There is nothing more educational in robotics than programming an actual robot. And nobody can explain the pitfalls and challenges in robotics better than Nature!

1.6 Bibliographical Remarks

MODEL-BASED PARADIGM

The field of robotics has gone through a series of paradigms for software design. The first major paradigm emerged in the mid-1970s, and is known as the *model-based paradigm*. The model-based paradigm began with a number of studies showing the hardness of controlling a high-DOF robotic manipulator in continuous spaces (Reif 1979). It culminated in text like Schwartz et al.'s (1987) analysis of the complexity of robot motion, a first singly exponential general motion planning algorithm by Canny (1987), and Latombe's (1991) seminal introductory text into the field of model-based motion planning (additional milestone contributions will be discussed in Chapter 14). This early work largely ignored the problem of uncertainty—even though it extensively began using randomization as a technique for solving hard motion planning problems (Kavraki et al. 1996). Instead, the assumption was that a full and accurate model of the robot and the environment be given, and the robot be deterministic. The model had to be sufficiently accurate that the residual uncertainty was managed by a low-level motion controller. Most motion planning techniques simply produced a single reference trajectory for the control of a manipulator, although ideas such as *potential fields* (Khatib 1986) and *navigation functions* (Koditschek 1987) provided mechanisms for reacting to the unforeseen—as long as it could be sensed. Applications of these early techniques, if any, were confined to environments where every little bit of uncertainty could be engineered away, or sensed with sufficient accuracy.

BEHAVIOR-BASED ROBOTICS

The field took a radical shift in the mid-1980s, when the lack of sensory feedback became the focus of an entire community of researchers within robotics. With strong convictions, the field of *behavior-based robotics* rejected the idea of any internal model. Instead, it was the interaction

with a physical environment of a *situated agent* (Kaelbling and Rosenschein 1991) that created the complexity in robot motion (a phenomena often called *emergent behavior* (Steels 1991)). Consequently, sensing played a paramount role, and internal models were rejected (Brooks 1990).

The enthusiasm in this field was fueled by some early successes that were far beyond the reach of traditional model-based motion planning algorithms. One of them was "Genghis," a hexapod robot developed by Brooks (1986). A relatively simple finite state automaton was able to control the gait of this robot even in rugged terrain. The key to success of such techniques lay in sensing: the control was entirely driven by environment interaction, as perceived through the robot's sensors. Some of the early work impressed by creating a seemingly complex robot through clever exploitation of environment feedback (Connell 1990). More recently, the paradigm enjoyed commercial success through a robotic vacuum cleaning robot (IRobots Inc. 2004), whose software follows the behavior-based paradigm.

Due to the lack of internal models and a focus on simple control mechanism, most robot systems were confined to relatively simple tasks, where the momentary sensor information was sufficient to determine the right choice of control. Recognizing this limitation, more recent work in this field embraced *hybrid control* architectures (Arkin 1998), in which behavior-based technique provided low-level control, whereas a model-based planner coordinated the robot's actions at a high, abstract level. Such hybrid architectures are commonplace in robotics today. They are not dissimilar to the seminal work on three-layered architectures by Gat (1998), which took its origins in "Shakey the Robot" (Nilsson 1984).

Modern probabilistic robotics has emerged since the mid-1990s, although its roots can be traced back to the invention of the Kalman filter (Kalman 1960). In many ways, probabilistic robotics falls in between model-based and behavior-based techniques. In probabilistic robotics, there are models, but they are assumed to be incomplete and insufficient for control. There are also sensor measurements, but they too are assumed to be incomplete and insufficient for control. Through the integration of both, models and sensor measurements, a control action can be devised. Statistics provides the mathematical glue to integrate models and sensor measurements.

Many of the key advances in the field of probabilistic robotics will be discussed in future chapters. Some of the cornerstones in this field include the advent of Kalman filter techniques for high-dimensional perception problems by Smith and Cheeseman (1986), the invention of occupancy grid maps by (Elfes 1987; Moravec 1988), and the re-introduction of partially observable planning techniques due to Kaelbling et al. (1998). The past decade has seen an explosion of techniques: Particle filters have become vastly popular (Dellaert et al. 1999), and researchers have developed new programming methodologies focused on Bayesian information processing (Thrun 2000b; Lebeltel et al. 2004; Park et al. 2005). This development went hand in hand with the deployment of physical robot systems driven by probabilistic algorithms, such as industrial machines for cargo handling by Durrant-Whyte (1996), entertainment robots in museums (Burgard et al. 1999a; Thrun et al. 2000a; Siegwart et al. 2003), and robots in nursing and health care (Pineau et al. 2003d). An open-source software package for mobile robot control that heavily utilizes probabilistic techniques is described in Montemerlo et al. (2003a).

The field of commercial robotics is also at a turning point. In its annual World Robotics Survey, the *United Nations and the International Federation of Robotics* 2004 finds a 19% annual increase in the size of the robotic market worldwide. Even more spectacular is the change of market segments, which indicates a solid transition from industrial applications to service robotics and consumer products.

6 *Robot Perception*

6.1 Introduction

Environment measurement models comprise the second domain-specific model in probabilistic robotics, next to motion models. Measurement models describe the formation process by which sensor measurements are generated in the physical world. Today's robots use a variety of different sensor modalities, such as tactile sensors, range sensors, or cameras. The specifics of the model depends on the sensor: Imaging sensors are best modeled by projective geometry, whereas sonar sensors are best modeled by describing the sound wave and its reflection on surfaces in the environment.

Probabilistic robotics explicitly models the noise in sensor measurements. Such models account for the inherent uncertainty in the robot's sensors. Formally, the measurement model is defined as a conditional probability distribution $p(z_t \mid x_t, m)$, where x_t is the robot pose, z_t is the measurement at time t, and m is the map of the environment. Although we mainly address range-sensors throughout this chapter, the underlying principles and equations are not limited to this type of sensors. Instead the basic principle can be applied to any kind of sensor, such as a camera or a bar-code operated landmark detector.

SONAR RANGE SCAN

To illustrate the basic problem of mobile robots that use their sensors to perceive their environment, Figure 6.1a shows a typical *sonar range scan* obtained in a corridor with a mobile robot equipped with a cyclic array of 24 ultrasound sensors. The distances measured by the individual sensors are depicted in light gray and the map of the environment is shown in black. Most of these measurements correspond to the distance of the nearest object in the measurement cone; some measurements, however, have failed to detect any object.

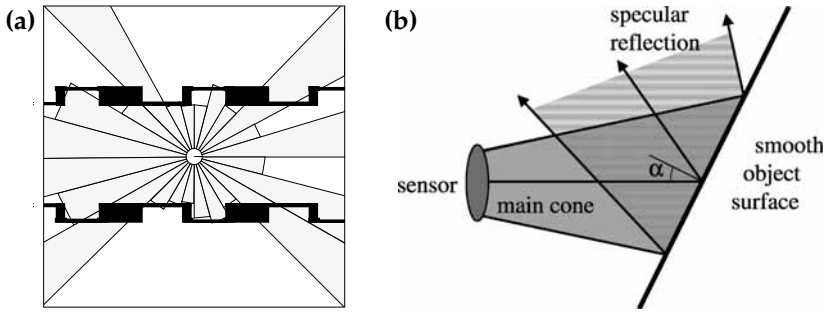


Figure 6.1 (a) Typical ultrasound scan of a robot in its environment. (b) A misreading in ultrasonic sensing. This effect occurs when firing a sonar signal towards a reflective surface at an angle α that exceeds half the opening angle of the sensor.

The inability for sonar to reliably measure range to nearby objects is often paraphrased as sensor noise. Technically, this noise is quite predictable: When measuring smooth surfaces (such as walls), the reflection is usually *specular*, and the wall effectively becomes a mirror for the sound wave. This can be problematic when hitting a smooth surface at an angle. Here the echo may travel into a direction other than the sonar sensor, as illustrated in Figure 6.1b. This effect often leads to overly large range measurements when compared to the true distance to the nearest object in the main cone. The likelihood of this to happen depends on a number of properties, such as the surface material, the angle between the surface normal and the direction of the sensor cone, the range of the surface, the width of the main sensor cone, and the sensitivity of the sonar sensor. Other errors, such as short readings, may be caused by cross-talk between different sensors (sound is slow!) or by unmodeled objects in the proximity of the robot, such as people.

SPECULAR REFLECTION

LASER RANGE SCAN

Figure 6.2 shows a typical *laser range scan*, acquired with a 2-D laser range finder. Laser is similar to sonar in that it also actively emits a signal and records its echo, but in the case of laser the signal is a light beam. A key difference to sonars is that lasers provide much more focused beams. The specific laser in Figure 6.2 is based on a time-of-flight measurement, and measurements are spaced in one degree increments.

As a rule of thumb, the more accurate a sensor model, the better the results—though there are some important caveats that were already discussed in Chapter 2.4.4. In practice, however, it is often impossible to model

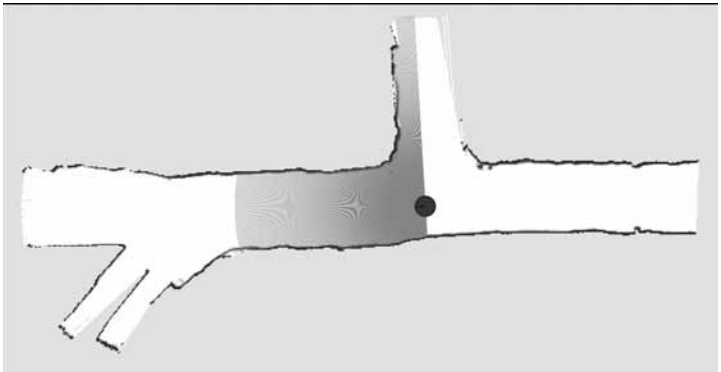


Figure 6.2 A typical laser range scan, acquired with a SICK LMS laser. The environment shown here is a coal mine. Image courtesy of Dirk Hähnel, University of Freiburg.

a sensor accurately, primarily due to the complexity of physical phenomena.

Often, the response characteristics of a sensor depends on variables we prefer not to make explicit in a probabilistic robotics algorithm (such as the surface material of walls, which for no particular reason is commonly not considered in robotic mapping). Probabilistic robotics accommodates inaccuracies of sensor models in the stochastic aspects: By modeling the measurement process as a conditional probability density, $p(z_t \mid x_t)$, instead of a deterministic function $z_t = f(x_t)$, the uncertainty in the sensor model can be accommodated in the non-deterministic aspects of the model. Herein lies a key advantage of probabilistic techniques over classical robotics: in practice, we can get away with extremely crude models. However, when devising a probabilistic model, care has to be taken to capture the different types of uncertainties that may affect a sensor measurement.

Many sensors generate more than one numerical measurement value when queried. For example, cameras generate entire arrays of values (brightness, saturation, color); similarly, range finders usually generate entire scans of ranges. We will denote the number of such measurement values within a measurement z_t by K, hence we can write:

$$z_t = \{z_t^1, \ldots, z_t^K\} \tag{6.1}$$

We will use z_t^k to refer to an individual measurement (e.g., one range value).

The probability $p(z_t \mid x_t, m)$ is obtained as the product of the individual measurement likelihoods

$$p(z_t \mid x_t, m) = \prod_{k=1}^{K} p(z_t^k \mid x_t, m) \tag{6.2}$$

Technically, this amounts to an *independence assumption* between the noise in each individual measurement beam—just as our Markov assumption assumes independent noise over time (c.f., Chapter 2.4.4). This assumption is only true in the ideal case. We already discussed possible causes of dependent noise in Chapter 2.4.4. To recapitulate, dependencies typically exist due to a range of factors: people, who often corrupt measurements of several adjacent sensors; errors in the model m; approximations in the posterior; and so on. For now, however, we will simply not worry about violations of the independence assumption, as we will return to this issue in later chapters.

6.2 Maps

To express the process of generating measurements, we need to specify the environment in which a measurement is generated. A *map* of the environment is a list of objects in the environment and their locations. We have already informally discussed maps in the previous chapter, where we developed robot motion models that took into consideration the occupancy of different locations in the world. Formally, a map m is a list of objects in the environment along with their properties:

$$m = \{m_1, m_2, \ldots, m_N\} \tag{6.3}$$

Here N is the total number of objects in the environment, and each m_n with $1 \leq n \leq N$ specifies a property. Maps are usually indexed in one of two ways, known as *feature-based* and *location-based*. In feature-based maps, n is a feature index. The value of m_n contains, next to the properties of a feature, the Cartesian location of the feature. In location-based maps, the index n corresponds to a specific location. In planar maps, it is common to denote a map element by $m_{x,y}$ instead of m_n, to make explicit that $m_{x,y}$ is the property of a specific world coordinate, $(x \;\; y)$.

VOLUMETRIC MAPS

Both types of maps have advantages and disadvantages. Location-based maps are *volumetric*, in that they offer a label for any location in the world. Volumetric maps contain information not only about objects in the environment, but also about the absence of objects (e.g., free-space). This is quite

different in feature-based maps. *Feature-based maps* only specify the shape of the environment at the specific locations, namely the locations of the objects contained in the map. Feature representation makes it easier to adjust the position of an object; e.g., as a result of additional sensing. For this reason, feature-based maps are popular in the robotic mapping field, where maps are constructed from sensor data. In this book, we will encounter both types of maps—in fact, we will occasionally move from one representation to the other.

A classical map representation is known as *occupancy grid map*, which will be discussed in detail in Chapter 9. Occupancy maps are location-based: They assign to each x-y coordinate a binary occupancy value that specifies whether or not a location is occupied with an object. Occupancy grid maps are great for mobile robot navigation: They make it easy to find paths through the unoccupied space.

Throughout this book, we will drop the distinction between the physical world and the map. Technically, sensor measurements are caused by physical objects, not the map of those objects. However, it is tradition to condition sensor models on the map m; hence we will adopt a notation that suggests measurements depend on the map.

6.3 Beam Models of Range Finders

Range finders are among the most popular sensors in robotics. Our first *measurement model* in this chapter is therefore an approximative physical model of range finders. Range finders measure the range to nearby objects. Range may be measured along a beam—which is a good model of the workings of laser range finders—or within a cone—which is the preferable model of ultrasonic sensors.

6.3.1 The Basic Measurement Algorithm

Our model incorporates four types of measurement errors, all of which are essential to making this model work: small measurement noise, errors due to unexpected objects, errors due to failures to detect objects, and random unexplained noise. The desired model $p(z_t \mid x_t, m)$ is therefore a mixture of four densities, each of which corresponds to a particular type of error:

1. **Correct range with local measurement noise.** In an ideal world, a range finder would always measure the correct range to the nearest object in its

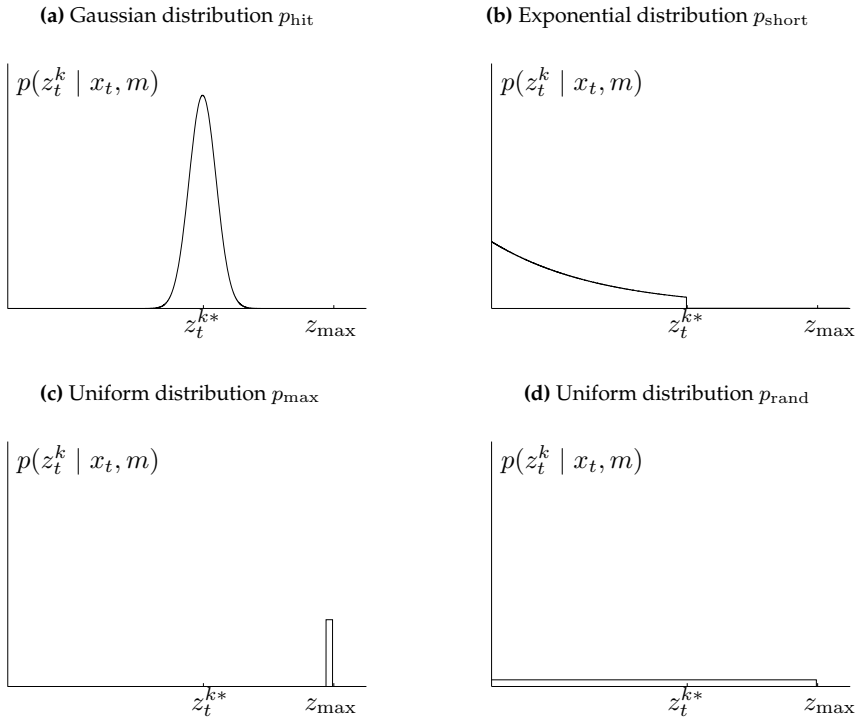


Figure 6.3 Components of the range finder sensor model. In each diagram the horizontal axis corresponds to the measurement z_t^k, the vertical to the likelihood.

measurement field. Let us use z_t^{k*} to denote the "true" range of the object measured by z_t^k. In location-based maps, the range z_t^{k*} can be determined using *ray casting*; in feature-based maps, it is usually obtained by searching for the closest feature within a measurement cone. However, even if the sensor correctly measures the range to the nearest object, the value it returns is subject to error. This error arises from the limited resolution of range sensors, atmospheric effect on the measurement signal, and so on. This *measurement noise* is usually modeled by a narrow Gaussian with mean z_t^{k*} and standard deviation σ_{hit}. We will denote the Gaussian by p_{hit}. Figure 6.3a illustrates this density p_{hit}, for a specific value of z_t^{k*}.

MEASUREMENT NOISE

In practice, the values measured by the range sensor are limited to the interval $[0; z_{\max}]$, where $z_{\max}$ denotes the maximum sensor range. Thus,

the measurement probability is given by

$$p_{\text{hit}}(z_t^k \mid x_t, m) = \begin{cases} \eta\, \mathcal{N}(z_t^k; z_t^{k*}, \sigma_{\text{hit}}^2) & \text{if } 0 \leq z_t^k \leq z_{\max} \\ 0 & \text{otherwise} \end{cases} \tag{6.4}$$

where z_t^{k*} is calculated from x_t and m via ray casting, and $\mathcal{N}(z_t^k; z_t^{k*}, \sigma_{\text{hit}}^2)$ denotes the univariate normal distribution with mean z_t^{k*} and standard deviation σ_{hit}:

$$\mathcal{N}(z_t^k; z_t^{k*}, \sigma_{\text{hit}}^2) = \frac{1}{\sqrt{2\pi\sigma_{\text{hit}}^2}}\, e^{-\frac{1}{2}\frac{(z_t^k - z_t^{k*})^2}{\sigma_{\text{hit}}^2}} \tag{6.5}$$

The normalizer η evaluates to

$$\eta = \left(\int_0^{z_{\max}} \mathcal{N}(z_t^k; z_t^{k*}, \sigma_{\text{hit}}^2)\, dz_t^k \right)^{-1} \tag{6.6}$$

The standard deviation σ_{hit} is an intrinsic noise parameter of the measurement model. Below we will discuss strategies for setting this parameter.

2. **Unexpected objects.** Environments of mobile robots are dynamic, whereas maps m are static. As a result, objects not contained in the map can cause range finders to produce surprisingly short ranges—at least when compared to the map. A typical example of moving objects are people that share the operational space of the robot. One way to deal with such objects is to treat them as part of the state vector and estimate their location; another, much simpler approach, is to treat them as sensor noise. Treated as sensor noise, unmodeled objects have the property that they cause ranges to be shorter than z_t^{k*}, not longer.

   The likelihood of sensing unexpected objects decreases with range. To see, imagine there are two people that independently and with the same fixed likelihood show up in the perceptual field of a proximity sensor. One person's range is r_1, and the second person's range is r_2. Let us further assume that $r_1 < r_2$, without loss of generality. Then we are more likely to measure r_1 than r_2. Whenever the first person is present, our sensor measures r_1. However, for it to measure r_2, the second person must be present *and* the first must be absent.

   Mathematically, the probability of range measurements in such situations is described by an *exponential distribution*. The parameter of this distribution, λ_{short}, is an intrinsic parameter of the measurement model. According to the definition of an exponential distribution we obtain the following

equation for $p_{\text{short}}(z_t^k \mid x_t, m)$:

$$p_{\text{short}}(z_t^k \mid x_t, m) = \begin{cases} \eta\, \lambda_{\text{short}}\, e^{-\lambda_{\text{short}} z_t^k} & \text{if } 0 \leq z_t^k \leq z_t^{k*} \\ 0 & \text{otherwise} \end{cases} \tag{6.7}$$

As in the previous case, we need a normalizer η since our exponential is limited to the interval $\left[0; z_t^{k*}\right]$. Because the cumulative probability in this interval is given as

$$\begin{aligned} \int_0^{z_t^{k*}} \lambda_{\text{short}}\, e^{-\lambda_{\text{short}} z_t^k}\, dz_t^k &= -e^{-\lambda_{\text{short}} z_t^{k*}} + e^{-\lambda_{\text{short}} 0} \\ &= 1 - e^{-\lambda_{\text{short}} z_t^{k*}} \end{aligned} \tag{6.8}$$

the value of η can be derived as:

$$\eta = \frac{1}{1 - e^{-\lambda_{\text{short}} z_t^{k*}}} \tag{6.9}$$

Figure 6.3b depicts this density graphically. This density falls off exponentially with the range z_t^k.

3. **Failures.** Sometimes, obstacles are missed altogether. For example, this happens frequently for sonar sensors as a result of specular reflections. Failures also occur with laser range finders when sensing black, light-absorbing objects, or for some laser systems when measuring objects in bright sunlight. A typical result of a *sensor failure* is a *max-range measurement*: the sensor returns its maximum allowable value $z_{\max}$. Since such events are quite frequent, it is necessary to explicitly model max-range measurements in the measurement model.

SENSOR FAILURE

We will model this case with a point-mass distribution centered at $z_{\max}$:

$$p_{\max}(z_t^k \mid x_t, m) = I(z = z_{\max}) = \begin{cases} 1 & \text{if } z = z_{\max} \\ 0 & \text{otherwise} \end{cases} \tag{6.10}$$

Here I denotes the indicator function that takes on the value 1 if its argument is true, and is 0 otherwise. Technically, $p_{\max}$ does not possess a probability density function. This is because $p_{\max}$ is a discrete distribution. However, this shall not worry us here, as our mathematical model of evaluating the probability of a sensor measurement is not affected by the non-existence of a density function. (In our diagrams, we simply draw $p_{\max}$ as a very narrow uniform distribution centered at $z_{\max}$, so that we can pretend a density exists).

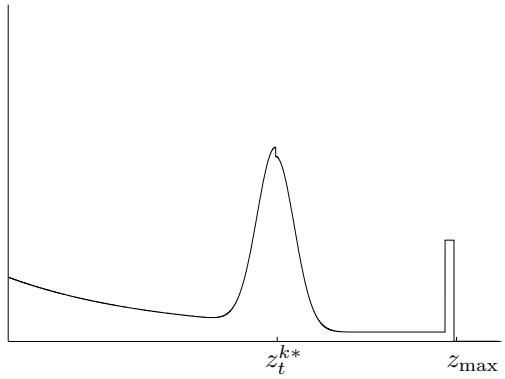


Figure 6.4 "Pseudo-density" of a typical mixture distribution $p(z_t^k \mid x_t, m)$.

UNEXPLAINABLE MEASUREMENTS

4. **Random measurements.** Finally, range finders occasionally produce entirely *unexplainable measurements*. For example, sonars often generate phantom readings when they bounce off walls, or when they are subject to cross-talk between different sensors. To keep things simple, such measurements will be modeled using a uniform distribution spread over the entire sensor measurement range $[0; z_{\max}]$:

$$p_{\text{rand}}(z_t^k \mid x_t, m) = \begin{cases} \frac{1}{z_{\max}} & \text{if } 0 \leq z_t^k < z_{\max} \\ 0 & \text{otherwise} \end{cases} \tag{6.11}$$

Figure 6.3d shows the density of the distribution p_{rand}.

These four different distributions are now mixed by a weighted average, defined by the parameters z_{hit}, z_{short}, $z_{\max}$, and z_{rand} with $z_{\text{hit}} + z_{\text{short}} + z_{\max} + z_{\text{rand}} = 1$.

$$p(z_t^k \mid x_t, m) = \begin{pmatrix} z_{\text{hit}} \\ z_{\text{short}} \\ z_{\max} \\ z_{\text{rand}} \end{pmatrix}^T \cdot \begin{pmatrix} p_{\text{hit}}(z_t^k \mid x_t, m) \\ p_{\text{short}}(z_t^k \mid x_t, m) \\ p_{\max}(z_t^k \mid x_t, m) \\ p_{\text{rand}}(z_t^k \mid x_t, m) \end{pmatrix} \tag{6.12}$$

A typical density resulting from this linear combination of the individual densities is shown in Figure 6.4 (with our visualization of the point-mass distribution $p_{\max}$ as a small uniform density). As the reader may notice, the basic characteristics of all four basic models are still present in this combined density.

```
1:  Algorithm beam_range_finder_model(z_t, x_t, m):

2:      q = 1
3:      for k = 1 to K do
4:          compute z_t^{k*} for the measurement z_t^k using ray casting
5:          p = z_hit · p_hit(z_t^k | x_t, m) + z_short · p_short(z_t^k | x_t, m)
6:              + z_max · p_max(z_t^k | x_t, m) + z_rand · p_rand(z_t^k | x_t, m)
7:          q = q · p
8:      return q
```

Table 6.1 Algorithm for computing the likelihood of a range scan z_t, assuming conditional independence between the individual range measurements in the scan.

The range finder model is implemented by the algorithm **beam_range_finder_model** in Table 6.1. The input of this algorithm is a complete range scan z_t, a robot pose x_t, and a map m. Its outer loop (lines 2 and 7) multiplies the likelihood of individual sensor beams z_t^k, following Equation (6.2). Line 4 applies ray casting to compute the noise-free range for a particular sensor measurement. The likelihood of each individual range measurement z_t^k is computed in line 5, which implements the mixing rule for densities stated in (6.12). After iterating through all sensor measurements z_t^k in z_t, the algorithm returns the desired probability $p(z_t \mid x_t, m)$.

6.3.2 Adjusting the Intrinsic Model Parameters

In our discussion so far we have not addressed the question of how to choose the various parameters of the sensor model. These parameters include the mixing parameters z_{hit}, z_{short}, z_{max}, and z_{rand}. They also include the parameters σ_{hit} and λ_{short}. We will refer to the set of all intrinsic parameters as Θ. Clearly, the likelihood of any sensor measurement is a function of Θ. Thus, we will now discuss an algorithm for *adjusting model parameters*.

One way to determine the intrinsic parameters is to rely on data. Figure 6.5 depicts two series of 10,000 measurements obtained with a mobile robot traveling through a typical office environment. Both plots show only range measurements for which the expected range was approximately 3 me-

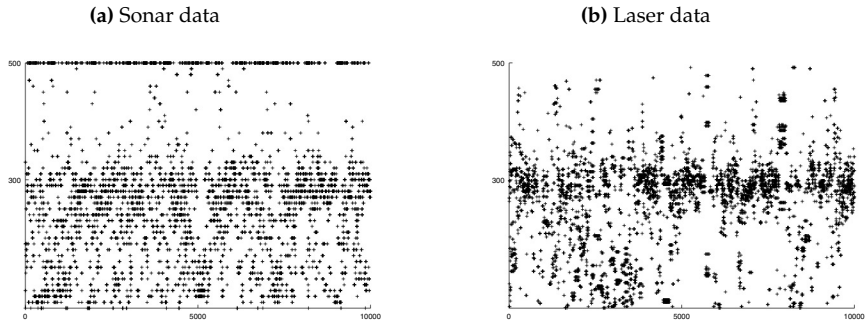


Figure 6.5 Typical data obtained with (a) a sonar sensor and (b) a laser-range sensor in an office environment for a "true" range of 300 cm and a maximum range of 500 cm.

ters (between 2.9m and 3.1m). The left plot depicts the data for sonar sensors, and the right plot the corresponding data for laser sensors. In both plots, the x-axis shows the number of the reading (from 1 to 10,000), and the y-axis is the range measured by the sensor.

Whereas most of the measurements are close to the correct range for both sensors, the behaviors of the sensors differ substantially. The ultrasound sensor appears to suffer from many more measurement noise and detection errors. Quite frequently it fails to detect an obstacle, and instead reports maximum range. In contrast, the laser range finder is more accurate. However, it also occasionally reports false ranges.

A perfectly acceptable way to set the intrinsic parameters Θ is by hand: simply eyeball the resulting density until it agrees with your experience. Another, more principled way is to learn these parameters from actual data. This is achieved by maximizing the likelihood of a reference data set $Z = \{z_i\}$ with associated positions $X = \{x_i\}$ and map m, where each z_i is an actual measurement, x_i is the pose at which the measurement was taken, and m is the map. The likelihood of the data Z is given by

$$p(Z \mid X, m, \Theta) \tag{6.13}$$

Our goal is to identify intrinsic parameters Θ that maximize this likelihood. Any estimator, or algorithm, that maximizes the likelihood of data is known as a *maximum likelihood estimator*, or *ML estimator*.

MAXIMUM LIKELIHOOD ESTIMATOR

Table 6.2 depicts the algorithm **learn_intrinsic_parameters**, which is an algorithm for calculating the maximum likelihood estimate for the intrinsic

1: **Algorithm learn_intrinsic_parameters(**Z, X, m**):**

2: *repeat until convergence criterion satisfied*

3: *for all* z_i *in* Z *do*

4: $\eta = [\, p_{\text{hit}}(z_i \mid x_i, m) + p_{\text{short}}(z_i \mid x_i, m) + p_{\text{max}}(z_i \mid x_i, m) + p_{\text{rand}}(z_i \mid x_i, m)\,]^{-1}$

5: *calculate* z_i^*

6: $e_{i,\text{hit}} = \eta\, p_{\text{hit}}(z_i \mid x_i, m)$

7: $e_{i,\text{short}} = \eta\, p_{\text{short}}(z_i \mid x_i, m)$

8: $e_{i,\text{max}} = \eta\, p_{\text{max}}(z_i \mid x_i, m)$

9: $e_{i,\text{rand}} = \eta\, p_{\text{rand}}(z_i \mid x_i, m)$

10: $z_{\text{hit}} = |Z|^{-1} \sum_i e_{i,\text{hit}}$

11: $z_{\text{short}} = |Z|^{-1} \sum_i e_{i,\text{short}}$

12: $z_{\text{max}} = |Z|^{-1} \sum_i e_{i,\text{max}}$

13: $z_{\text{rand}} = |Z|^{-1} \sum_i e_{i,\text{rand}}$

14: $\sigma_{\text{hit}} = \sqrt{\frac{1}{\sum_i e_{i,\text{hit}}} \sum_i e_{i,\text{hit}} (z_i - z_i^*)^2}$

15: $\lambda_{\text{short}} = \frac{\sum_i e_{i,\text{short}}}{\sum_i e_{i,\text{short}}\, z_i}$

16: *return* $\Theta = \{z_{\text{hit}}, z_{\text{short}}, z_{\text{max}}, z_{\text{rand}}, \sigma_{\text{hit}}, \lambda_{\text{short}}\}$

Table 6.2 Algorithm for learning the intrinsic parameters of the beam-based sensor model from data.

parameters. As we shall see below, the algorithm is an instance of the *expectation maximization* (EM) algorithm, an iterative procedure for estimating ML parameters.

Initially, the algorithm **learn_intrinsic_parameters** in Table 6.2 requires a good initialization of the intrinsic parameters σ_{hit} and λ_{short}. In lines 3 through 9, it estimates auxiliary variables: Each $e_{i,\text{xxx}}$ is the probability that the measurement z_i is caused by "xxx," where "xxx" is chosen from the four aspects of the sensor model, hit, short, max, and random. Subsequently, it es-

timates the intrinsic parameters in lines 10 through 15. The intrinsic parameters, however, are a function of the expectations calculated before. Adjusting the intrinsic parameters causes the expectations to change, for which reason the algorithm has to be iterated. However, in practice the iteration converges quickly, and a dozen iterations are usually sufficient to give good results.

Figure 6.6 graphically depicts four examples of data and the ML measurement model calculated by **learn_intrinsic_parameters**. The first row shows approximations to data recorded with the ultrasound sensor. The second row contains plots of two functions generated for laser range data. The columns correspond to different "true" ranges. The data is organized in histograms. One can clearly see the differences between the different graphs. The smaller the range z_t^{k*} the more accurate the measurement. For both sensors the Gaussians are narrower for the shorter range than they are for the longer measurement. Furthermore, the laser range finder is more accurate than the ultrasound sensor, as indicated by the narrower Gaussians and the smaller number of maximum range measurements. The other important thing to notice is the relatively high likelihood of short and random measurements. This large error likelihood has a disadvantage and an advantage: On the negative side, it reduces the information in each sensor reading, since the difference in likelihood between a hit and a random measurement is small. On the positive side this model is less susceptible to unmodeled *systematic* perturbations, such as people who block the robot's path for long periods of time.

Figure 6.7 illustrates the learned sensor model in action. Shown in Figure 6.7a is a 180 degree range scan. The robot is placed in a previously acquired occupancy grid map at its true pose. Figure 6.7b plots a map of the environment along with the likelihood $p(z_t \mid x_t, m)$ of this range scan projected into x-y-space (by maximizing over the orientation θ). The darker a location, the more likely it is. As is easily seen, all regions with high likelihood are located in the corridor. This comes at little surprise, as the specific scan is geometrically more consistent with corridor locations than with locations inside any of the rooms. The fact that the probability mass is spread out throughout the corridor suggests that a single sensor scan is insufficient to determine the robot's exact pose. This is largely due to the symmetry of the corridor. The fact that the posterior is organized in two narrow horizontal bands is due to the fact that the orientation of the robot is unknown: each of these bands corresponds to one of the two surviving heading directions of the robot.

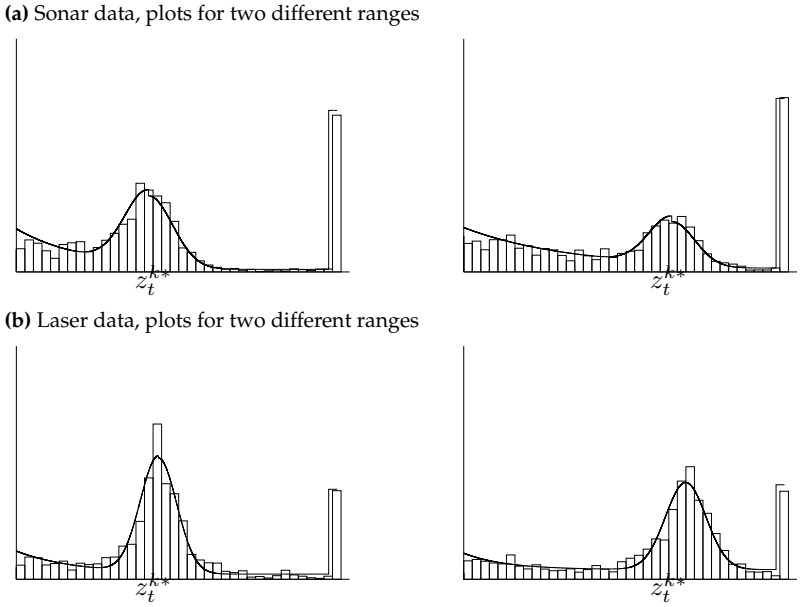


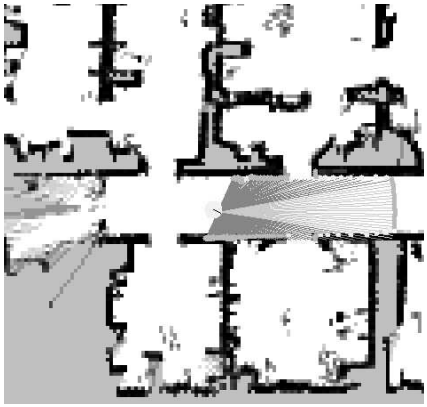
Figure 6.6 Approximation of the beam model based on (a) sonar data and (b) laser range data. The sensor models depicted on the left were obtained by a maximum likelihood approximation to the data sets depicted in Figure 6.5.

6.3.3 Mathematical Derivation of the Beam Model

To derive the ML estimator, it shall prove useful to introduce auxiliary variables c_i, the so-called correspondence variable. Each c_i can take on one of four values, hit, short, max, and random, corresponding to the four possible mechanisms that might have produced a measurement z_i.

Let us first consider the case in which the c_i's are known. We know which of the four mechanisms described above caused each measurement z_i. Based on the values of the c_i's, we can decompose Z into four disjoint sets, Z_{hit}, Z_{short}, Z_{max}, and Z_{rand}, which together comprise the set Z. The ML estimators for the intrinsic parameters z_{hit}, z_{short}, z_{max}, and z_{rand} are simply the

(a) Laser scan and part of the map



(b) Likelihood for different positions

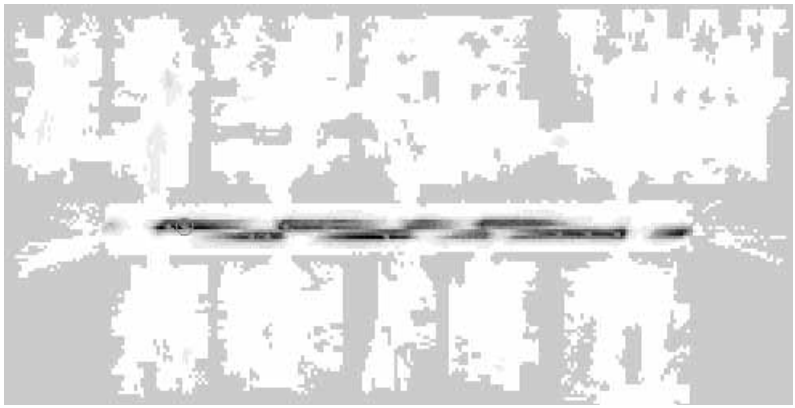


Figure 6.7 Probabilistic model of perception: (a) Laser range scan, projected into a previously acquired map m. (b) The likelihood $p(z_t \mid x_t, m)$, evaluated for all positions x_t and projected into the map (shown in gray). The darker a position, the larger $p(z_t \mid x_t, m)$.

normalized ratios:

$$\begin{pmatrix} z_{\text{hit}} \\ z_{\text{short}} \\ z_{\text{max}} \\ z_{\text{rand}} \end{pmatrix} = |Z|^{-1} \begin{pmatrix} |Z_{\text{hit}}| \\ |Z_{\text{short}}| \\ |Z_{\text{max}}| \\ |Z_{\text{rand}}| \end{pmatrix} \tag{6.14}$$

The remaining intrinsic parameters, σ_{hit} and λ_{short}, are obtained as follows.

For the data set Z_{hit}, we get from (6.5)

$$p(Z_{\text{hit}} \mid X, m, \Theta) = \prod_{z_i \in Z_{\text{hit}}} p_{\text{hit}}(z_i \mid x_i, m, \Theta) = \prod_{z_i \in Z_{\text{hit}}} \frac{1}{\sqrt{2\pi\sigma_{\text{hit}}^2}} e^{-\frac{1}{2}\frac{(z_i - z_i^*)^2}{\sigma_{\text{hit}}^2}} \tag{6.15}$$

Here z_i^* is the "true" range, computed from the pose x_i and the map m. A classic trick of ML estimation is to maximize the logarithm of the likelihood, instead of the likelihood directly. The logarithm is a strictly monotonic function, hence the maximum of the log-likelihood is also the maximum of the original likelihood. The log-likelihood is given by

$$\log p(Z_{\text{hit}} \mid X, m, \Theta) = \sum_{z_i \in Z_{\text{hit}}} \left[-\frac{1}{2} \log 2\pi\sigma_{\text{hit}}^2 - \frac{1}{2} \frac{(z_i - z_i^*)^2}{\sigma_{\text{hit}}^2} \right] \tag{6.16}$$

which is now easily transformed as follows

$$\begin{aligned} &\log p(Z_{\text{hit}} \mid X, m, \Theta) \\ &= -\frac{1}{2} \sum_{z_i \in Z_{\text{hit}}} \left[\log 2\pi\sigma_{\text{hit}}^2 + \frac{(z_i - z_i^*)^2}{\sigma_{\text{hit}}^2} \right] \\ &= -\frac{1}{2} \left[|Z_{\text{hit}}| \log 2\pi + 2|Z_{\text{hit}}| \log \sigma_{\text{hit}} + \sum_{z_i \in Z_{\text{hit}}} \frac{(z_i - z_i^*)^2}{\sigma_{\text{hit}}^2} \right] \\ &= \text{const.} - |Z_{\text{hit}}| \log \sigma_{\text{hit}} - \frac{1}{2\sigma_{\text{hit}}^2} \sum_{z_i \in Z_{\text{hit}}} (z_i - z_i^*)^2 \end{aligned} \tag{6.17}$$

The derivative of this expression in the intrinsic parameter σ_{hit} is as follows:

$$\frac{\partial \log p(Z_{\text{hit}} \mid X, m, \Theta)}{\partial \sigma_{\text{hit}}} = -\frac{|Z_{\text{hit}}|}{\sigma_{\text{hit}}} + \frac{1}{\sigma_{\text{hit}}^3} \sum_{z_i \in Z_{\text{hit}}} (z_i - z_i^*)^2 \tag{6.18}$$

The maximum of the log-likelihood is now obtained by setting this derivative to zero. From that we get the solution to our ML estimation problem.

$$\sigma_{\text{hit}} = \sqrt{\frac{1}{|Z_{\text{hit}}|} \sum_{z_i \in Z_{\text{hit}}} (z_i - z_i^*)^2} \tag{6.19}$$

The estimation of the remaining intrinsic parameter λ_{short} proceeds just about in the same way. The posterior over the data Z_{short} is given by

$$p(Z_{\text{short}} \mid X, m, \Theta) = \prod_{z_i \in Z_{\text{short}}} p_{\text{short}}(z_i \mid x_i, m) \tag{6.20}$$

$$= \prod_{z_i \in Z_{\text{short}}} \lambda_{\text{short}}\, e^{-\lambda_{\text{short}} z_i}$$

The logarithm is

$$\begin{aligned} \log p(Z_{\text{short}} \mid X, m, \Theta) &= \sum_{z_i \in Z_{\text{short}}} \log \lambda_{\text{short}} - \lambda_{\text{short}} z_i \\ &= |Z_{\text{short}}| \log \lambda_{\text{short}} - \lambda_{\text{short}} \sum_{z_i \in Z_{\text{short}}} z_i \end{aligned} \tag{6.21}$$

The first derivative of this expression with respect to the intrinsic parameter λ_{short} is as follows:

$$\frac{\partial \log p(Z_{\text{short}} \mid X, m, \Theta)}{\partial \lambda_{\text{short}}} = \frac{|Z_{\text{short}}|}{\lambda_{\text{short}}} - \sum_{z_i \in Z_{\text{short}}} z_i \tag{6.22}$$

Setting this to zero gives us the ML estimate for the intrinsic parameter λ_{short}

$$\lambda_{\text{short}} = \frac{|Z_{\text{short}}|}{\sum_{z_i \in Z_{\text{short}}} z_i} \tag{6.23}$$

This derivation assumed knowledge of the parameters c_i. We now extend it to the case where the c_i's are unknown. As we shall see, the resulting ML estimation problem lacks a closed-form solution. However, we can devise a technique that iterates two steps, one that calculates an expectation for the c_i's and one that computes the intrinsic model parameters under these expectations. As noted, the resulting algorithm is an instance of the *expectation maximization* algorithm, usually abbreviated as EM.

EM ALGORITHM

To derive EM, it will be beneficial to define the likelihood of the data Z first:

$$\begin{aligned} &\log p(Z \mid X, m, \Theta) \\ &= \sum_{z_i \in Z} \log p(z_i \mid x_i, m, \Theta) \\ &= \sum_{z_i \in Z_{\text{hit}}} \log p_{\text{hit}}(z_i \mid x_i, m) + \sum_{z_i \in Z_{\text{short}}} \log p_{\text{short}}(z_i \mid x_i, m) \\ &\quad + \sum_{z_i \in Z_{\text{max}}} \log p_{\text{max}}(z_i \mid x_i, m) + \sum_{z_i \in Z_{\text{rand}}} \log p_{\text{rand}}(z_i \mid x_i, m) \end{aligned} \tag{6.24}$$

This expression can be rewritten using the variables c_i:

$$\log p(Z \mid X, m, \Theta) = \sum_{z_i \in Z} I(c_i = \text{hit})\ \log p_{\text{hit}}(z_i \mid x_i, m) \tag{6.25}$$

$$
\begin{aligned}
&+I(c_i = \text{short})\ \log p_{\text{short}}(z_i \mid x_i, m) \\
&+I(c_i = \text{max})\ \log p_{\text{max}}(z_i \mid x_i, m) \\
&+I(c_i = \text{rand})\ \log p_{\text{rand}}(z_i \mid x_i, m)
\end{aligned}
$$

where I is the indicator function. Since the values for c_i are unknown, it is common to integrate them out. Put differently, EM maximizes the expectation $E[\log p(Z \mid X, m, \Theta)]$, where the expectation is taken over the unknown variables c_i:

$$
\begin{aligned}
(6.26)\quad & E[\log p(Z \mid X, m, \Theta)] \\
= \quad & \sum_i p(c_i = \text{hit})\ \log p_{\text{hit}}(z_i \mid x_i, m) + p(c_i = \text{short})\ \log p_{\text{short}}(z_i \mid x_i, m) \\
& +p(c_i = \text{max})\ \log p_{\text{max}}(z_i \mid x_i, m) + p(c_i = \text{rand})\ \log p_{\text{rand}}(z_i \mid x_i, m) \\
=: \quad & \sum_i e_{i,\text{hit}}\ \log p_{\text{hit}}(z_i \mid x_i, m) + e_{i,\text{short}}\ \log p_{\text{short}}(z_i \mid x_i, m) \\
& +e_{i,\text{max}}\ \log p_{\text{max}}(z_i \mid x_i, m) + e_{i,\text{rand}}\ \log p_{\text{rand}}(z_i \mid x_i, m)
\end{aligned}
$$

With the definition of the variable e as indicated. This expression is maximized in two steps. In a first step, we consider the intrinsic parameters σ_{hit} and λ_{short} given and calculate the expectation over the variables c_i.

$$
(6.27)\quad \begin{pmatrix} e_{i,\text{hit}} \\ e_{i,\text{short}} \\ e_{i,\text{max}} \\ e_{i,\text{rand}} \end{pmatrix} := \begin{pmatrix} p(c_i = \text{hit}) \\ p(c_i = \text{short}) \\ p(c_i = \text{max}) \\ p(c_i = \text{rand}) \end{pmatrix} = \eta \begin{pmatrix} p_{\text{hit}}(z_i \mid x_i, m) \\ p_{\text{short}}(z_i \mid x_i, m) \\ p_{\text{max}}(z_i \mid x_i, m) \\ p_{\text{rand}}(z_i \mid x_i, m) \end{pmatrix}
$$

The normalizer is given by

$$
\begin{aligned}
(6.28)\quad \eta \quad = \quad & [\ p_{\text{hit}}(z_i \mid x_i, m) + p_{\text{short}}(z_i \mid x_i, m) \\
& +p_{\text{max}}(z_i \mid x_i, m) + p_{\text{rand}}(z_i \mid x_i, m)\ ]^{-1}
\end{aligned}
$$

This step is called the "E-step," indicating that we calculate expectations over the latent variables c_i. The remaining step is now straightforward, since the expectations decouple the dependencies between the different components of the sensor model. First, we note that the ML mixture parameters are simply the normalized expectations

$$
(6.29)\quad \begin{pmatrix} z_{\text{hit}} \\ z_{\text{short}} \\ z_{\text{max}} \\ z_{\text{rand}} \end{pmatrix} = |Z|^{-1} \sum_i \begin{pmatrix} e_{i,\text{hit}} \\ e_{i,\text{short}} \\ e_{i,\text{max}} \\ e_{i,\text{rand}} \end{pmatrix}
$$

The ML parameters σ_{hit} and λ_{short} are then obtained analogously, by replacing the hard assignments in (6.19) and (6.23) by soft assignments weighted by the expectations.

$$\sigma_{\text{hit}} = \sqrt{\frac{1}{\sum_{z_i \in Z} e_{i,\text{hit}}} \sum_{z_i \in Z} e_{i,\text{hit}} (z_i - z_i^*)^2} \tag{6.30}$$

and

$$\lambda_{\text{short}} = \frac{\sum_{z_i \in Z} e_{i,\text{short}}}{\sum_{z_i \in Z} e_{i,\text{short}} \; z_i} \tag{6.31}$$

6.3.4 Practical Considerations

In practice, computing the densities of all sensor readings can be quite involved from a computational perspective. For example, laser range scanners often return hundreds of values per scan, at a rate of several scans per second. Since one has to perform a ray casting operation for each beam of the scan and every possible pose considered, the integration of the whole scan into the current belief cannot always be carried out in real-time. One typical approach to solve this problem is to incorporate only a small subset of all measurements (e.g., 8 equally spaced measurements per laser range scan instead of 360). This approach has an important additional benefit. Since adjacent beams of a range scan are often not independent, the state estimation process becomes less susceptible to correlated noise in adjacent measurements.

When dependencies between adjacent measurements are strong, the ML model may make the robot overconfident and yield suboptimal results. One simple remedy is to replace $p(z_t^k \mid x_t, m)$ by a "weaker" version $p(z_t^k \mid x_t, m)^\alpha$ for $\alpha < 1$. The intuition here is to reduce, by a factor of α, the information extracted from a sensor measurement (the log of this probability is given by $\alpha \log p(z_t^k \mid x_t, m)$). Another possibility—which we will only mention here—is to learn the intrinsic parameters in the context of the application: For example, in mobile localization it is possible to train the intrinsic parameters via gradient descent to yield good localization results over multiple time steps. Such a multi-time step methodology is significantly different from the single time step ML estimator described above. In practical implementations it can yield superior results; see Thrun (1998a).

The main drain of computing time for beam-based models is the ray casting operation. The runtime costs of computing $p(z_t \mid x_t, m)$ can be substantially reduced by *pre-cashing* the ray casting algorithm, and storing the

result in memory—so that the ray casting operation can be replaced by a (much faster) table lookup. An obvious implementation of this idea is to decompose the state space into a fine-grained three-dimensional grid, and to pre-compute the ranges z_t^{k*} for each grid cell. This idea was already investigated in Chapter 4.1. Depending on the resolution of the grid, the memory requirements can be significant. In mobile robot localization, we find that pre-computing the range with a grid resolution of 15 centimeters and 2 degrees works well for indoor localization problems. It fits well into the RAM for moderate-sized computers, yielding speed-ups by an order of magnitude over the plain implementation that casts rays online.

6.3.5 Limitations of the Beam Model

The beam-based sensor model, while closely linked to the geometry and physics of range finders, suffers two major drawbacks.

In particular, the beam-based model exhibits a *lack of smoothness.* In cluttered environments with many small obstacles, the distribution $p(z_t^k \mid x_t, m)$ can be very unsmooth in x_t. Consider, for example, an environment with many chairs and tables (like a typical conference room). A robot like the ones shown in Chapter 1 will sense the legs of those obstacles. Obviously, small changes of a robot's pose x_t can have a tremendous impact on the correct range of a sensor beam. As a result, the measurement model $p(z_t^k \mid x_t, m)$ is highly discontinuous in x_t. The heading direction θ_t is particularly affected, since small changes in heading can cause large displacements in x-y-space at a range.

Lack of smoothness has two problematic consequences. First, any approximate belief representation runs the danger of missing the correct state, as nearby states might have drastically different posterior likelihoods. This poses constraints on the accuracy of the approximation which, if not met, increase the resulting error in the posterior. Second, hill climbing methods for finding the most likely state are prone to local minima, due to the large number of local maxima in such unsmooth models.

The beam-based model is also computational involved. Evaluating $p(z_t^k \mid x_t, m)$ for each single sensor measurement z_t^k involves ray casting, which is computationally expensive. As noted above, the problem can be partially remedied by pre-computing the ranges over a discrete grid in pose space. Such an approach shifts the computation into an initial off-line phase, with the benefit that the algorithm is faster at run time. However, the resulting tables are very large, since they cover a large three-dimensional space. Thus,

pre-computing ranges is computationally expensive and requires substantial memory.

6.4 Likelihood Fields for Range Finders

6.4.1 Basic Algorithm

LIKELIHOOD FIELD

We will now describe an alternative model, called *likelihood field*, which overcomes these limitations. This model lacks a plausible physical explanation. In fact, it is an "ad hoc" algorithm that does not necessarily compute a conditional probability relative to any meaningful generative model of the physics of sensors. However, the approach works well in practice. The resulting posteriors are much smoother even in cluttered space, and the computation is more efficient.

The key idea is to first project the end points of a sensor scan z_t into the global coordinate space of the map. To do so, we need to know where relative to the global coordinate frame the robot's local coordinate system is located, where on the robot the sensor beam z_k originates, and where the sensor points. As usual let $x_t = (x \;\; y \;\; \theta)^T$ denote a robot pose at time t. Keeping with our two-dimensional view of the world, we denote the relative location of the sensor in the robot's fixed, local coordinate system by $(x_{k,\text{sens}} \;\; y_{k,\text{sens}})^T$, and the angular orientation of the sensor beam relative to the robot's heading direction by $\theta_{k,\text{sens}}$. These values are sensor-specific. The end point of the measurement z_t^k is now mapped into the global coordinate system via the obvious trigonometric transformation.

$$\begin{pmatrix} x_{z_t^k} \\ y_{z_t^k} \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x_{k,\text{sens}} \\ y_{k,\text{sens}} \end{pmatrix} + z_t^k \begin{pmatrix} \cos(\theta + \theta_{k,\text{sens}}) \\ \sin(\theta + \theta_{k,\text{sens}}) \end{pmatrix} \tag{6.32}$$

These coordinates are only meaningful when the sensor detects an obstacle. If the range sensor takes on its maximum value $z_t^k = z_{\max}$, these coordinates have no meaning in the physical world (even though the measurement does carry information). The likelihood field measurement model simply discards max-range readings.

Similar to the beam model discussed before, we assume three types of sources of noise and uncertainty:

1. **Measurement noise.** Noise arising from the measurement process is modeled using Gaussians. In x-y-space, this involves finding the nearest obstacle in the map. Let $dist$ denote the Euclidean distance between the

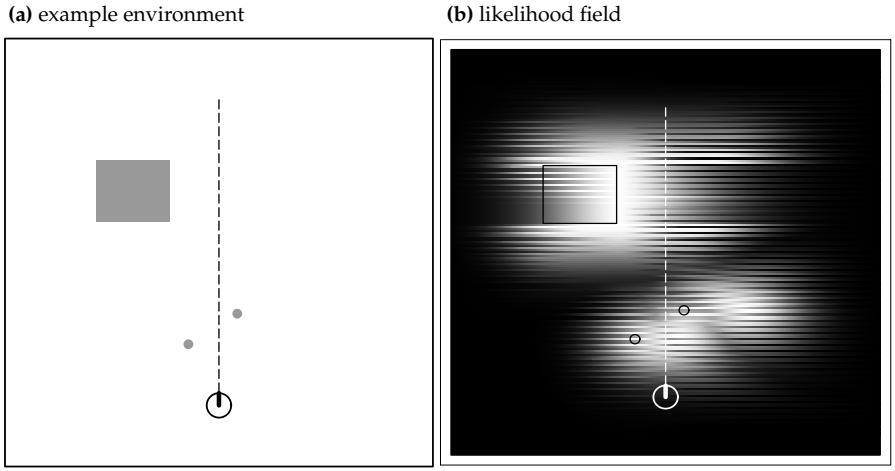


Figure 6.8 (a) Example environment with three obstacles (gray). The robot is located towards the bottom of the figure, and takes a measurement z_t^k as indicated by the dashed line. (b) Likelihood field for this obstacle configuration: the darker a location, the less likely it is to perceive an obstacle there. The probability $p(z_t^k \mid x_t, m)$ for the specific sensor beam is shown in Figure 6.9.

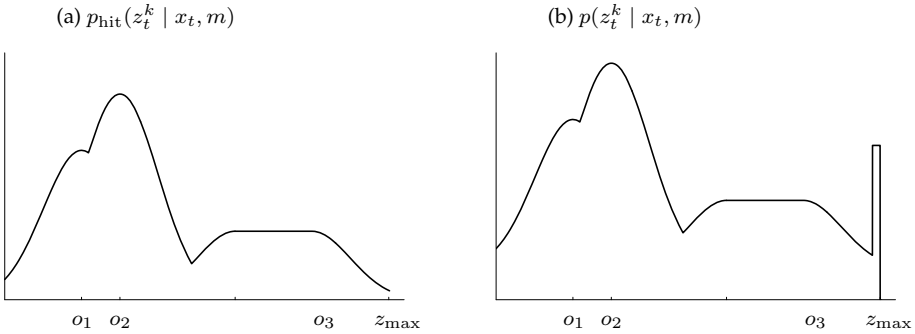


Figure 6.9 (a) Probability $p_{\text{hit}}(z_t^k)$ as a function of the measurement z_t^k, for the situation depicted in Figure 6.8. Here the sensor beam passes by three obstacles, with respective nearest points o_1, o_2, and o_3. (b) Sensor probability $p(z_t^k \mid x_t, m)$, obtained for the situation depicted in Figure 6.8, obtained by adding two uniform distributions.

measurement coordinates $(x_{z_t^k} \;\; y_{z_t^k})^T$ and the nearest object in the map m. Then the probability of a sensor measurement is given by a zero-centered Gaussian, which captures the sensor noise:

$$p_{\text{hit}}(z_t^k \mid x_t, m) \quad = \quad \varepsilon_{\sigma_{\text{hit}}}(dist) \tag{6.33}$$

Figure 6.8a depicts a map, and Figure 6.8b shows the corresponding Gaussian likelihood for measurement points $(x_{z_t^k} \;\; y_{z_t^k})^T$ in 2-D space. The brighter a location, the more likely it is to measure an object with a range finder. The density p_{hit} is now obtained by intersecting (and normalizing) the likelihood field by the sensor axis, indicated by the dashed line in Figure 6.8. The resulting function is the one shown in Figure 6.9a.

2. **Failures.** As before, we assume that max-range readings have a distinct large likelihood. As before, this is modeled by a point-mass distribution p_{max}.

3. **Unexplained random measurements.** Finally, a uniform distribution p_{rand} is used to model random noise in perception.

Just as for the beam-based sensor model, the desired probability $p(z_t^k \mid x_t, m)$ integrates all three distributions:

$$z_{\text{hit}} \cdot p_{\text{hit}} + z_{\text{rand}} \cdot p_{\text{rand}} + z_{\text{max}} \cdot p_{\text{max}} \tag{6.34}$$

using the familiar mixing weights z_{hit}, z_{rand}, and z_{max}. Figure 6.9b shows an example of the resulting distribution $p(z_t^k \mid x_t, m)$ along a measurement beam. It should be easy to see that this distribution combines p_{hit}, as shown in Figure 6.9a, and the distributions p_{max} and p_{rand}. Much of what we said about adjusting the mixing parameters transfers over to our new sensor model. They can be adjusted by hand or learned using the ML estimator. A representation like the one in Figure 6.8b, which depicts the likelihood of an obstacle detection as a function of global x-y-coordinates, is called the *likelihood field*.

Table 6.3 provides an algorithm for calculating the measurement probability using the likelihood field. The reader should already be familiar with the outer loop, which multiplies the individual values of $p(z_t^k \mid x_t, m)$, assuming independence between the noise in different sensor beams. Line 4 checks if the sensor reading is a max range reading, in which case it is simply ignored. Lines 5 to 8 handle the interesting case: Here the distance to the nearest obstacle in x-y-space is computed (line 7), and the resulting likelihood is obtained in line 8 by mixing a normal and a uniform distribution. As

1: **Algorithm likelihood_field_range_finder_model**(z_t, x_t, m)**:**

2: $q = 1$
3: *for all* k *do*
4: *if* $z_t^k \neq z_{\max}$
5: $x_{z_t^k} = x + x_{k,\text{sens}} \cos\theta - y_{k,\text{sens}} \sin\theta + z_t^k \cos(\theta + \theta_{k,\text{sens}})$
6: $y_{z_t^k} = y + y_{k,\text{sens}} \cos\theta + x_{k,\text{sens}} \sin\theta + z_t^k \sin(\theta + \theta_{k,\text{sens}})$
7: $dist = \min_{x',y'} \left\{ \sqrt{(x_{z_t^k} - x')^2 + (y_{z_t^k} - y')^2} \,\middle|\, \langle x', y' \rangle \text{ occupied in } m \right\}$
8: $q = q \cdot \left(z_{\text{hit}} \cdot \mathbf{prob}(dist, \sigma_{\text{hit}}) + \frac{z_{\text{random}}}{z_{\max}} \right)$
9: *return* q

Table 6.3 Algorithm for computing the likelihood of a range finder scan using Euclidean distance to the nearest neighbor. The function **prob**$(dist, \sigma_{\text{hit}})$ computes the probability of the distance under a zero-centered Gaussian distribution with standard deviation σ_{hit}.

before, the function **prob**$(dist, \sigma_{\text{hit}})$ computes the probability of $dist$ under a zero-centered Gaussian distribution with standard deviation σ_{hit}.

The search for the nearest neighbor in the map (line 7) is the most costly operation in algorithm **likelihood_field_range_finder_model**. To speed up this search, it is advantageous to pre-compute the likelihood field, so that calculating the probability of a measurement amounts to a coordinate transformation followed by a table lookup. Of course, if a discrete grid is used, the result of the lookup is only approximate, in that it might return the wrong obstacle coordinates. However, the effect on the probability $p(z_t^k \mid x_t, m)$ is typically small even for moderately course grids.

6.4.2 Extensions

A key advantage of the likelihood field model over the beam-based model discussed before is smoothness. Due to the smoothness of the Euclidean distance, small changes in the robot's pose x_t only have small effects on the resulting distribution $p(z_t^k \mid x_t, m)$. Another key advantage is that the pre-computation takes place in 2-D, instead of 3-D, increasing the compactness of the pre-computed information.

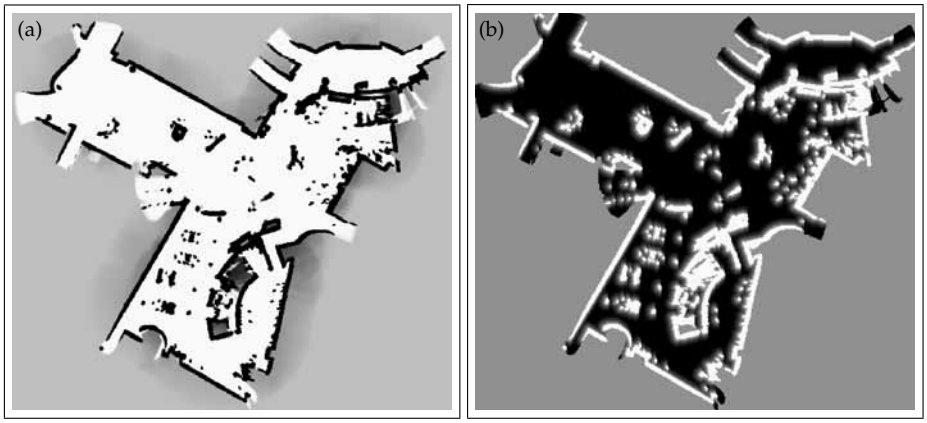


Figure 6.10 (a) Occupancy grid map of the San Jose Tech Museum, (b) pre-processed likelihood field.

However, the current model has three key disadvantages: First, it does not explicitly model people and other dynamics that might cause short readings. Second, it treats sensors as if they can "see through walls." This is because the ray casting operation was replaced by a nearest neighbor function, which is incapable of determining whether a path to a point is intercepted by an obstacle in the map. And third, our approach does not take map uncertainty into account. In particular, it cannot handle *unexplored* areas, for which the map is highly uncertain or unspecified.

The basic algorithm **likelihood_field_range_finder_model** can be extended to diminish the effect of these limitations. For example, one might sort map occupancy values into three categories: *occupied*, *free*, and *unknown*, instead of just the first two. When a sensor measurement z_t^k falls into the category *unknown*, its probability $p(z_t^k \mid x_t, m)$ is assumed to be the constant value $\frac{1}{z_{\max}}$. The resulting probabilistic model is crude. It assumes that in the unexplored space every sensor measurement is equally likely.

Figure 6.10 shows a map and the corresponding likelihood field. Here again the gray-level of an x-y-location indicates the likelihood of receiving a sensor reading there. The reader may notice that the distance to the nearest obstacle is only employed *inside* the map, which corresponds to the explored terrain. Outside, the likelihood $p(z_t^k \mid x_t, m)$ is a constant. For computational efficiency, it is worthwhile to pre-compute the nearest neighbor for a fine-grained 2-D grid.

Likelihood fields over the visible space can also be defined for the most

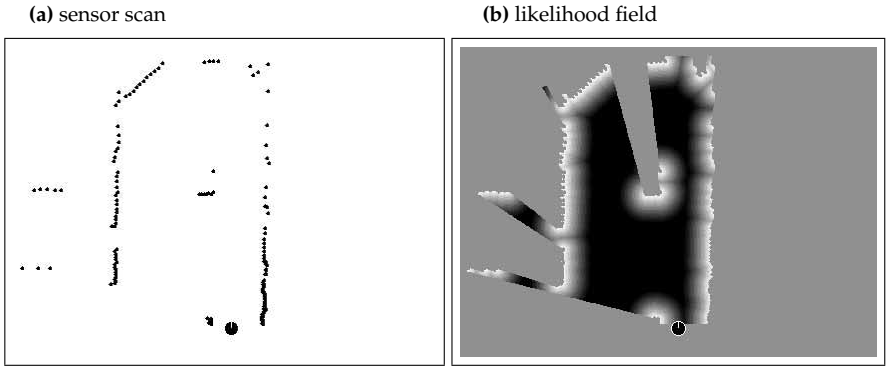


Figure 6.11 (a) Sensor scan, from a bird's eye perspective. The robot is placed at the bottom of this figure, generating a proximity scan that consists of the 180 dots in front of the robot. (b) Likelihood function generated from this sensor scan. The darker a region, the smaller the likelihood for sensing an object there. Notice that occluded regions are white, hence infer no penalty.

recent scan, which in fact defines a local map. Figure 6.11 shows such a likelihood field. It plays an important role in techniques that align individual scans.

6.5 Correlation-Based Measurement Models

There exist a number of range sensor models in the literature that measure correlations between a measurement and the map. A common technique is known as *map matching*. Map matching requires techniques discussed in later chapters of this book, namely the ability to transform scans into occupancy maps. Typically, map matching compiles small numbers of consecutive scans into *local maps*, denoted m_{local}. Figure 6.12 shows such a local map, here in the form of an occupancy grid map. The sensor measurement model compares the local map m_{local} to the global map m, such that the more similar m and m_{local}, the larger $p(m_{\text{local}} \mid x_t, m)$. Since the local map is represented relative to the robot location, this comparison requires that the cells of the local map are transformed into the coordinate framework of the global map. Such a transformation can be done similar to the coordinate transform (6.32) of sensor measurements used in the likelihood field model. If the robot is at location x_t, we denote by $m_{x,y,\text{local}}(x_t)$ the grid cell in the local map that

MAP MATCHING

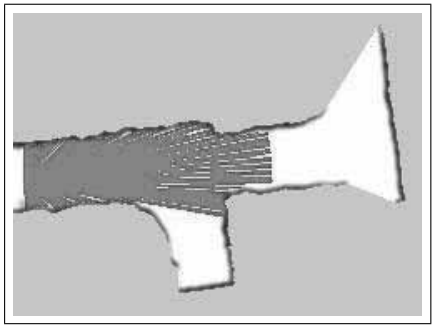


Figure 6.12 Example of a local map generated from 10 range scans, one of which is shown.

corresponds to $(x \;\; y)^T$ in global coordinates. Once both maps are in the same reference frame, they can be compared using the map correlation function, which is defined as follows:

$$\rho_{m,m_{\text{local},x_t}} = \frac{\sum_{x,y}(m_{x,y} - \bar{m}) \cdot (m_{x,y,\text{local}}(x_t) - \bar{m})}{\sqrt{\sum_{x,y}(m_{x,y} - \bar{m})^2 \sum_{x,y}(m_{x,y,\text{local}}(x_t) - \bar{m})^2}} \tag{6.35}$$

Here the sum is evaluated over cells defined in both maps, and $\bar{m}$ is the average map value:

$$\bar{m} = \frac{1}{2N} \sum_{x,y} (m_{x,y} + m_{x,y,\text{local}}) \tag{6.36}$$

where N denotes the number of elements in the overlap between the local and global map. The correlation $\rho_{m,m_{\text{local},x_t}}$ scales between ± 1. Map matching interprets the value

$$p(m_{\text{local}} \mid x_t, m) = \max\{\rho_{m,m_{\text{local},x_t}}, 0\} \tag{6.37}$$

as the probability of the local map conditioned on the global map m and the robot pose x_t. If the local map is generated from a single range scan z_t, this probability substitutes the measurement probability $p(z_t \mid x_t, m)$.

Map matching has a number of nice properties: just like the likelihood field model, it is easy to compute, though it does not yield smooth probabilities in the pose parameter x_t. One way to approximate the likelihood field (and to obtain smoothness) is to convolve the map m with a Gaussian smoothness kernel, and to run map matching on this smoothed map.

A key advantage of map matching over likelihood fields is that it explicitly considers the free-space in the scoring of two maps; the likelihood field technique only considers the end point of the scans, which by definition correspond to occupied space (or noise). On the other hand, many mapping techniques build local maps beyond the reach of the sensors. For example, many techniques build circular maps around the robot, setting to 0.5 areas beyond the range of actual sensor measurements. In such cases, there is a danger that the result of map matching incorporates areas beyond the actual measurement range, as if the sensor can "see through walls." Such side-effects are found in a number of implemented map matching techniques.

A further disadvantage is that map matching does not possess a plausible physical explanation. Correlations are the normalized quadratic distance between maps, which is *not* the noise characteristic of range sensors.

6.6 Feature-Based Measurement Models

6.6.1 Feature Extraction

FEATURES

The sensor models discussed thus far are all based on raw sensor measurements. An alternative approach is to extract *features* from the measurements. If we denote the feature extractor as a function f, the features extracted from a range measurement are given by $f(z_t)$. Most feature extractors extract a small number of features from high-dimensional sensor measurements. A key advantage of this approach is the enormous reduction of computational complexity: While inference in the high-dimensional measurement space can be costly, inference in the low-dimensional feature space can be orders of magnitude more efficient.

The discussion of specific algorithms for feature extraction is beyond the scope of this book. The literature offers a wide range of features for a number of different sensors. For range sensors, it is common to identify lines, corners, or local minima in range scans, which correspond to walls, corners, or objects such as tree trunks. When cameras are used for navigation, the processing of camera images falls into the realm of computer vision. Computer vision has devised a myriad of feature extraction techniques from camera images. Popular features include edges, corners, distinct patterns, and objects of distinct appearance. In robotics, it is also common to define places as features, such as hallways and intersections.

6.6.2 Landmark Measurements

In many robotics applications, features correspond to distinct objects in the physical world. For example, in indoor environments features may be door posts or windowsills; outdoors they may correspond to tree trunks or corners of buildings. In robotics, it is common to call those physical objects *landmarks*, to indicate that they are being used for robot navigation.

LANDMARKS

The most common model for processing landmarks assumes that the sensor can measure the range and the bearing of the landmark relative to the robot's local coordinate frame. Such sensors are called *range and bearing sensors*. The existence of a range-bearing sensor is not an implausible assumption: Any local feature extracted from range scans come with range and bearing information, as do visual features detected by stereo vision. In addition, the feature extractor may generate a *signature*. In this book, we assume a signature is a numerical value (e.g., an average color); it may equally be an integer that characterizes the type of the observed landmark, or a multidimensional vector characterizing a landmark (e.g., height and color).

RANGE AND BEARING SENSOR

SIGNATURE OF A LANDMARK

If we denote the range by r, the bearing by ϕ, and the signature by s, the feature vector is given by a collection of triplets

$$f(z_t) = \{f_t^1, f_t^2, \ldots\} = \{\begin{pmatrix} r_t^1 \\ \phi_t^1 \\ s_t^1 \end{pmatrix}, \begin{pmatrix} r_t^2 \\ \phi_t^1 \\ s_t^2 \end{pmatrix}, \ldots\} \tag{6.38}$$

The number of features identified at each time step is variable. However, many probabilistic robotic algorithms assume conditional independence between features

$$p(f(z_t) \mid x_t, m) = \prod_i p(r_t^i, \phi_t^i, s_t^i \mid x_t, m) \tag{6.39}$$

Conditional independence applies if the noise in each individual measurement $(r_t^i \quad \phi_t^i \quad s_t^i)^T$ is independent of the noise in other measurements $(r_t^j \quad \phi_t^j \quad s_t^j)^T$ (for $i \neq j$). Under the conditional independence assumption, we can process one feature at a time, just as we did in several of our range measurement models. This makes it much easier to develop algorithms that implement probabilistic measurement models.

Let us now devise a sensor model for features. In the beginning of this chapter, we distinguished between two types of maps: *feature-based* and *location-based*. Landmark measurement models are usually defined only for feature-based maps. The reader may recall that those maps consist of lists of features, $m = \{m_1, m_2, \ldots\}$. Each feature may possess a signature and a

location coordinate. The location of a feature, denoted $m_{i,x}$ and $m_{i,y}$, is simply its coordinate in the global coordinate frame of the map.

The measurement vector for a noise-free landmark sensor is easily specified by the standard geometric laws. We will model noise in landmark perception by independent Gaussian noise on the range, bearing, and the signature. The resulting measurement model is formulated for the case where the i-th feature at time t corresponds to the j-th landmark in the map. As usual, the robot pose is given by $x_t = (x \;\; y \;\; \theta)^T$.

$$\begin{pmatrix} r_t^i \\ \phi_t^i \\ s_t^i \end{pmatrix} = \begin{pmatrix} \sqrt{(m_{j,x} - x)^2 + (m_{j,y} - y)^2} \\ \operatorname{atan2}(m_{j,y} - y, m_{j,x} - x) - \theta \\ s_j \end{pmatrix} + \begin{pmatrix} \varepsilon_{\sigma_r^2} \\ \varepsilon_{\sigma_\phi^2} \\ \varepsilon_{\sigma_s^2} \end{pmatrix} \tag{6.40}$$

Here ε_{σ_r}, $\varepsilon_{\sigma_\phi}$, and ε_{σ_s} are zero-mean Gaussian error variables with standard deviations σ_r, σ_ϕ, and σ_s, respectively.

6.6.3 Sensor Model with Known Correspondence

DATA ASSOCIATION PROBLEM

A key problem for range/bearing sensors is known as the *data association problem*. This problem arises when landmarks cannot be uniquely identified, so that some residual uncertainty exists with regards to the identity of a landmark. For developing a range/bearing sensor model, it shall prove useful to introduce a *correspondence variable* between the feature f_t^i and the landmark m_j in the map. This variable will be denoted by c_t^i with $c_t^i \in \{1, \ldots, N+1\}$; N is the number of landmarks in the map m. If $c_t^i = j \leq N$, then the i-th feature observed at time t corresponds to the j-th landmark in the map. In other words, c_t^i is the true identity of an observed feature. The only exception occurs with $c_t^i = N + 1$: Here a feature observation does not correspond to any feature in the map m. This case is important for handling spurious landmarks; it is also of great relevance for the topic of robotic mapping, in which the robot may encounter previously unobserved landmarks.

CORRESPONDENCE VARIABLE

Table 6.4 depicts the algorithm for calculating the probability of a feature f_t^i with known correspondence $c_t^i \leq N$. Lines 3 and 4 calculate the true range and bearing to the landmark. The probability of the measured ranges and bearing is then calculated in line 5, assuming independence in the noise. As the reader easily verifies, this algorithm implements Equation (6.40).

1:	**Algorithm landmark_model_known_correspondence(**f_t^i, c_t^i, x_t, m**):**
2:	$j = c_t^i$
3:	$\hat{r} = \sqrt{(m_{j,x} - x)^2 + (m_{j,y} - y)^2}$
4:	$\hat{\phi} = \text{atan2}(m_{j,y} - y, m_{j,x} - x)$
5:	$q = \mathbf{prob}(r_t^i - \hat{r}, \sigma_r) \cdot \mathbf{prob}(\phi_t^i - \hat{\phi}, \sigma_\phi) \cdot \mathbf{prob}(s_t^i - s_j, \sigma_s)$
6:	*return* q

Table 6.4 Algorithm for computing the likelihood of a landmark measurement. The algorithm requires as input an observed feature $f_t^i = (r_t^i \;\; \phi_t^i \;\; s_t^i)^T$, and the true identity of the feature c_t^i, the robot pose $x_t = (x \;\; y \;\; \theta)^T$, and the map m. Its output is the numerical probability $p(f_t^i \mid c_t^i, m, x_t)$.

6.6.4 Sampling Poses

Sometimes it is desirable to sample robot poses x_t that correspond to a measurement f_t^i with feature identity c_t^i. We already encountered such sampling algorithms in the previous chapter, where we discussed robot motion models. Such sampling models are also desirable for sensor models. For example, when localizing a robot globally, it shall become useful to generate sample poses that incorporate a sensor measurement to generate initial guesses for the robot pose.

While in the general case, sampling poses x_t that correspond to a sensor measurement z_t is difficult, for our landmark model we can actually provide an efficient sampling algorithm. However, such sampling is only possible under further assumptions. In particular, we have to know the prior $p(x_t \mid c_t^i, m)$. For simplicity, let us assume this prior is uniform (it generally is not!). Bayes rule then suggests that

$$\begin{aligned} p(x_t \mid f_t^i, c_t^i, m) &= \eta \, p(f_t^i \mid c_t^i, x_t, m) \, p(x_t \mid c_t^i, m) \\ &= \eta \, p(f_t^i \mid c_t^i, x_t, m) \end{aligned} \tag{6.41}$$

Sampling from $p(x_t \mid f_t^i, c_t^i, m)$ can now be achieved from the "inverse" of the sensor model $p(f_t^i \mid c_t^i, x_t, m)$. Table 6.5 depicts an algorithm that samples poses x_t. The algorithm is tricky: Even in the noise-free case, a landmark observation does not uniquely determine the location of the robot. Instead, the robot may be on a circle around the landmark, whose diameter is the range to the landmark. The indeterminacy of the robot pose also follows

1: **Algorithm sample_landmark_model_known_correspondence(**f_t^i, c_t^i, m**):**

2: $j = c_t^i$
3: $\hat{\gamma} = \text{rand}(0, 2\pi)$
4: $\hat{r} = r_t^i + \textbf{sample}(\sigma_r)$
5: $\hat{\phi} = \phi_t^i + \textbf{sample}(\sigma_\phi)$
6: $x = m_{j,x} + \hat{r}\cos\hat{\gamma}$
7: $y = m_{j,y} + \hat{r}\sin\hat{\gamma}$
8: $\theta = \hat{\gamma} - \pi - \hat{\phi}$
9: *return* $(x \;\; y \;\; \theta)^T$

Table 6.5 Algorithm for sampling poses from a landmark measurement $f_t^i = (r_t^i \;\; \phi_t^i \;\; s_t^i)^T$ with known identity c_t^i.

from the fact that the range and bearing provide two constraints in a three-dimensional space of robot poses.

To implement a pose sampler, we have to sample the remaining free parameter, which determines where on the circle around the landmark the robot is located. This parameter is called $\hat{\gamma}$ in Table 6.5, and is chosen at random in line 3. Lines 4 and 5 perturb the measured range and bearing, exploiting the fact that the mean and the measurement are treated symmetrically in Gaussians. Finally, lines 6 through 8 recover the pose that corresponds to $\hat{\gamma}$, $\hat{r}$, and $\hat{\phi}$.

Figure 6.13 illustrates the pose distribution $p(x_t \mid f_t^i, c_t^i, m)$ (left diagram) and also shows a sample drawn with our algorithm **sample_landmark_model_known_correspondence** (right diagram). The posterior is projected into x-y-space, where it becomes a ring around the measured range r_t^i. In 3-D pose space, it is a spiral that unfolds the ring with the angle θ.

6.6.5 Further Considerations

Both algorithms for landmark-based measurements assume known correspondence. The case of unknown correspondence will be discussed in detail in later chapters, when we address algorithms for localization and mapping under unknown correspondence.

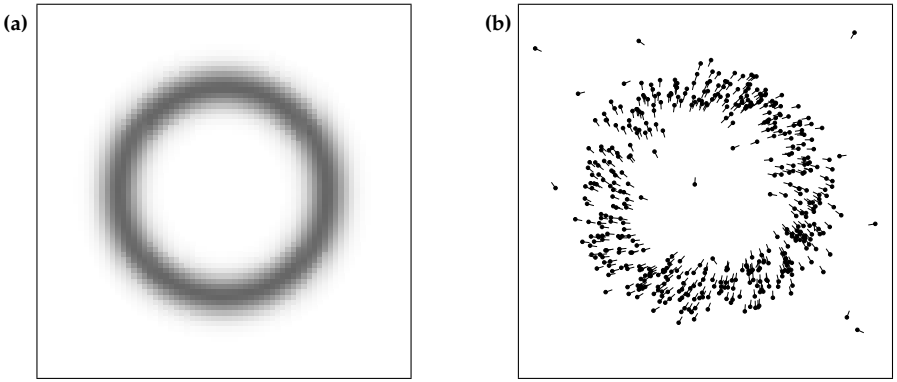


Figure 6.13 Landmark detection model: (a) Posterior distribution of the robot's pose given that it detected a landmark in 5m distance and 30deg relative bearing (projected onto 2-D). (b) Sample robot poses generated from such a detection. The lines indicate the orientation of the poses.

A comment is in order on the topic of landmark signature. Most published algorithms do not make the use of appearance features explicit. When the signature is not provided, all landmarks look equal, and the data association problem of estimating the correspondence variables is harder. We have included the signature in our model because it is a valuable source of information that can often be easily extracted from the sensor measurements.

As noted above, the main motivation for using features instead of the full measurement vector is computational in nature: It is much easier to manage a few hundred features than a few billion range measurements. Our model presented here is extremely crude, and it clearly does not capture the physical laws that underlie the sensor formation process. Nevertheless, the model tends to work well in a great number of applications.

It is important to notice that the reduction of measurements into features comes at a price. In the robotics literature, features are often (mis)taken for *sufficient statistics* of the measurement vector z_t. Let X be a variable of interest (e.g., a map, a pose), and Y some other information that we might bring to bear (e.g., past sensor measurements). Then f is a sufficient statistics of z_t if

SUFFICIENT STATISTICS

$$p(X \mid z_t, Y) \quad = \quad p(X \mid f(z_t), Y) \tag{6.42}$$

In practice, however, a lot of information is sacrificed by using features instead of the full measurement vector. This lost information makes certain

problems more difficult, such as the data association problem of determining whether or not the robot just revisited a previously explored location. It is easy to understand the effects of feature extraction by introspection: When you open your eyes, the visual image of your environment is probably sufficient to tell you unambiguously where you are—even if you were globally uncertain before. If, on the other hand, you only sense certain features, such as the relative location of door posts and windowsills, you would probably be much less certain as to where you are. Quite likely the information may be insufficient for global localization.

With the advent of fast computers, features have gradually lost importance in the field of robotics. Especially when using range sensors, most state-of-the-art algorithms rely on dense measurement vectors, and they use dense location-based maps to represent the environment. Nevertheless, features are still great for educational purposes. They enable us to introduce the basic concepts in probabilistic robotics, and with proper treatment of problems such as the correspondence problem they can be brought to bear even in cases where maps are composed of dense sets of scan points. For this reason, a number of algorithms in this book are first described for feature representations, and then extended into algorithms using raw sensor measurements.

6.7 Practical Considerations

This section surveyed a range of measurement models. We placed a strong emphasis on models for range finders, due to their great importance in robotics. However, the models discussed here are only representatives of a much broader class of probabilistic models. In choosing the right model, it is important to trade off physical realism with properties that might be desirable for an algorithm using these models. For example, we noted that a physically realistic model of range sensors may yield probabilities that are not smooth in the alleged robot pose—which in turn causes problems for algorithms such as particle filters. Physical realism is therefore not the only criterion in choosing the right sensor model; an equally important criterion is the goodness of a model for the algorithm that utilizes it.

As a general rule of thumb, the more accurate a model, the better. In particular, the more information we can extract from a sensor measurement, the better. Feature-based models extract relatively little information, by virtue of the fact that feature extractors project high-dimensional sensor measurements into lower dimensional space. As a result, feature-based methods tend

to produce inferior results. This disadvantage is offset by superior computational properties of feature-based representations.

When adjusting the intrinsic parameters of a measurement model, it is often useful to artificially inflate the uncertainty. This is because of a key limitation of the probabilistic approach: to make probabilistic techniques computationally tractable, we have to ignore dependencies that exist in the physical world, along with a myriad of latent variables that cause these dependencies. When such dependencies are not modeled, algorithms that integrate evidence from multiple measurements quickly become overconfident. Such *overconfidence* can ultimately lead to wrong conclusions, which negatively affects the results. In practice, it is therefore a good rule of thumb to reduce the information conveyed by a sensor. Doing so by projecting the measurement into a low-dimensional feature space is one way of achieving this. However, it suffers the limitations mentioned above. Uniformly decaying the information by exponentiating a measurement model with a parameter α, as discussed in Chapter 6.3.4, is a much better way, in that it does not introduce additional variance in the outcome of a probabilistic algorithm.

OVERCONFIDENCE

6.8 Summary

This section described probabilistic measurement models.

- Starting with models for range finders—and lasers in particular—we discussed measurement models $p(z_t^k \mid x_t, m)$. The first such model used ray casting to determine the shape of $p(z_t^k \mid x_t, m)$ for particular maps m and poses x_t. We devised a mixture model that addressed the various types of noise that can affect range measurements.

- We devised a maximum likelihood technique for identifying the intrinsic noise parameters of the measurement model. Since the measurement model is a mixture model, we provided an iterative procedure for maximum likelihood estimation. Our approach was an instance of the expectation maximization algorithm, which alternates a phase that calculates expectations over the type of error underlying a measurement, with a maximization phase that finds in closed form the best set of intrinsic parameters relative to these expectations.

- An alternative measurement model for range finders is based on likelihood fields. This technique used the nearest distance in 2-D coordinates to model the probability $p(z_t^k \mid x_t, m)$. We noted that this approach tends

to yield smoother distributions $p(z_t^k \mid x_t, m)$. This comes at the expense of undesired side effects: The likelihood field technique ignores information pertaining to free-space, and it fails to consider occlusions in the interpretation of range measurements.

- A third measurement model is based on map matching. Map matching maps sensor scans into local maps, and correlates those maps with global maps. This approach lacks a physical motivation, but can be implemented very efficiently.
- We discussed how pre-computation can reduce the computational burden at runtime. In the beam-based measurement model, the pre-computation takes place in 3-D; the likelihood field requires only a 2-D pre-computation.
- We presented a feature-based sensor model, in which the robot extracts the range, bearing, and signature of nearby landmarks. Feature-based techniques extract from the raw sensor measurements distinct features. In doing so, they reduce the dimensionality of the sensor measurements by several orders of magnitude.
- At the end of the chapter, a discussion on practical issues pointed out some of the pitfalls that may arise in concrete implementations.

6.9 Bibliographical Remarks

This chapter only skims the rich literature on physical modeling of sensors. More accurate models of sonar range sensor can be found in Blahut et al. (1991); Grunbaum et al. (1992) and in Etter (1996). Models of laser range finders are described by Rees (2001). An empirical discussion of proper noise models can be found in Sahin et al. (1998). Relative to these models, the models in this chapter are extremely crude.

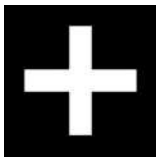
An early work on beam models for range sensors can be found in the seminal work by Moravec (1988). A similar model was later applied to mobile robot localization by Burgard et al. (1996). A beam-based model like the one described in this chapter together with the pre-caching of range measurements has first been described by Fox et al. (1999b). The likelihood fields have first been published by Thrun (2001), although they are closely related to the rich literature on scan matching techniques (Besl and McKay 1992). They in fact can be regarded as a soft variant of the correlation model described by Konolige and Chou (1999). Methods for computing the correlation between occupancy grid maps have also been quite popular. Thrun (1993) computes the sum of squared errors between the individual cells of two grid maps. Schiele and Crowley (1994) present a comparison of different models including correlation-based approaches. Yamauchi and Langley (1997) analyzed the robustness of map matching to dynamic environments. Duckett and Nehmzow (2001) transform local occupancy grids into histograms that can be matched more efficiently.

TESLA EXHIBIT 1011, Page 184

Range and bearings measurements for point landmarks are commonplace in the SLAM literature. Possibly the first mention is by Leonard and Durrant-Whyte (1991). In earlier work, Crowley (1989) devised measurement models for straight line objects.

6.10 Exercises

1. Many early robots navigating using features used artificial landmarks in the environment that were easy to recognize. A good place to mount such markers is a ceiling (why?). A classical example is a visual marker: Suppose we attach the following marker to the ceiling:



Let the world coordinates of the marker be x_m and y_m, and its orientation relative to the global coordinate system θ_m. We will denote the robot's pose by x_r, y_r, and θ_r.

Now assume that we are given a routine that can detect the marker in the image plane of a perspective camera. Let x_i and y_i denote the coordinates of the marker in the image plane, and θ_i its angular orientation. The camera has a focal length of f. From projective geometry, we know that each displacement d in x-y-space gets projected to a proportional displacement of $d \cdot \frac{f}{h}$ in the image plane. (You have to make some choices on your coordinate systems; make these choices explicit).

Your questions:

(a) Describe mathematically where to expect the marker (in global coordinates x_m, y_m, θ_m) when its image coordinates are x_i, y_i, θ_i, and the robot is at x_r, y_r, θ_r.

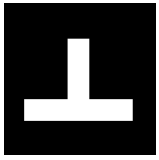
(b) Provide a mathematical equation for computing the image coordinates x_i, y_i, θ_i from the robot pose x_r, y_r, θ_r and the marker coordinates x_m, y_m, θ_m.

(c) Now give a mathematical equation for determining the robot coordinates x_r, y_r, θ_r assuming we know the true marker coordinates x_m, y_m, θ_m and the image coordinates x_i, y_i, θ_i.

(d) So far we assumed there is only a single marker. Now suppose there are multiple (indistinguishable) markers of the type shown above. How many such markers must a robot be able to see to uniquely identify its pose? Draw such a configuration, and argue why it is sufficient.

Hint: You don't need to consider the uncertainty in the measurement for answering this question. Also, note that the marker is symmetrical. This has an impact on the answer of these questions!

2. In this exercise, you will be asked to extend our calculation in the previous exercise to include error covariances. To simplify the calculation, we now assume a non-symmetric marker whose absolute orientation can be estimated:



Also for simplicity, we assume there shall be no noise in the orientation. However, the x-y estimates in the image plane will be noisy. Specifically, let the measurements be subject to zero-mean Gaussian noise with covariance

$$\Sigma = \begin{pmatrix} \sigma^2 & 0 & 0 \\ 0 & \sigma^2 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

for some positive value of σ^2.

Calculate for the three questions above the corresponding covariances. In particular,

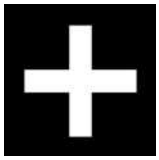
(a) Given the image coordinates x_i, y_i, θ_i and the robot coordinates x_r, y_r, θ_r, what is the error covariance for the values of x_m, y_m, θ_m.

(b) Given the robot coordinates x_r, y_r, θ_r and the marker covariance x_m, y_m, θ_m, what is your error covariance for the values of x_i, y_i, θ_i?

(c) Given the marker covariance x_m, y_m, θ_m and the image coordinates x_i, y_i, θ_i, what is your error covariance for the values of x_r, y_r, θ_r?

Notice that not all those distributions may be Gaussian. For this exercise, it is fine to apply a Taylor series expansion to attain a Gaussian posterior, but you have to explain how you did this.

3. Now you are being asked to implement a routine **sample_marker_model**, which accepts as an input the location of a marker x_m, y_m, θ_m and the location of the perceived marker in the image plane x_i, y_i, θ_i, and generates as an output samples of the robot pose x_r, y_r, θ_r. The marker is the same ambiguous marker as in Exercise 1:



Generate a plot of samples for the robot coordinates x_r and y_r, for the following parameters (you can ignore the orientation θ_r in your plot).

problem #	x_m	y_m	θ_m	x_i	y_i	θ_i	h/f	σ^2
#1	$0cm$	$0cm$	$0°$	$0cm$	$0cm$	$0°$	200	$0.1cm^2$
#2	$0cm$	$0cm$	$0°$	$1cm$	$0cm$	$0°$	200	$0.1cm^2$
#3	$0cm$	$0cm$	$0°$	$2cm$	$0cm$	$45°$	200	$0.1cm^2$
#4	$0cm$	$0cm$	$0°$	$2cm$	$0cm$	$45°$	200	$1.0cm^2$
#5	$50cm$	$150cm$	$10°$	$1cm$	$6cm$	$200°$	250	$0.5cm^2$

All your plots should show coordinate axes with units. Notice: If you cannot devise an exact sampler, provide an approximate one and explain your approximations.

4. For this exercise you need access to a robot with a sonar sensor, of the type often used in indoor robotics. Place the sensor in front of a flat wall, at a range d and an angle ϕ. Measure the frequency at which the sensor detects the wall. Plot this value for different values of d (in 0.5 meter increments) and different values of ϕ (in 5 degree increments). What do you find?

9 *Occupancy Grid Mapping*

9.1 Introduction

The previous two chapters discussed the application of probabilistic techniques to a low-dimensional perceptual problem, that of estimating a robot's pose. We assumed that the robot was given a map in advance. This assumption is legitimate in quite a few real-world applications, as maps are often available a priori or can be constructed by hand. Some application domains, however, do not provide the luxury of coming with an a priori map. Surprisingly enough, most buildings do not comply with the blueprints generated by their architects. And even if blueprints were accurate, they would not contain furniture and other items that, from a robot's perspective, determine the shape of the environment just as much as walls and doors. Being able to learn a map from scratch can greatly reduce the efforts involved in installing a mobile robot, and enable robots to adapt to changes without human supervision. In fact, mapping is one of the core competencies of truly autonomous robots.

Acquiring maps with mobile robots is a challenging problem for a number of reasons:

- The hypothesis space, which is the space of all possible maps, is huge. Since maps are defined over a continuous space, the space of all maps has infinitely many dimensions. Even under discrete approximations, such as the grid approximation that shall be used in this chapter, maps can easily be described 10^5 or more variables. The sheer size of this high-dimensional space makes it challenging to calculate full posteriors over maps; hence, the Bayes filtering approach that worked well for localization is inapplicable to the problem of learning maps, at least in its naive form discussed thus far.

- Learning maps is a "chicken-and-egg" problem, for which reason it is often referred to as the *simultaneous localization and mapping* (SLAM) or *concurrent mapping and localization problem*. First, there is a localization problem. When the robot moves through its environment, it accumulates errors in odometry, making it gradually less certain as to where it is. Methods exist for determining the robot's pose when a map is available, as we have seen in the previous chapter. Second, there is a mapping problem. Constructing a map when the robot's poses are known is also relatively easy—a claim that will be substantiated in this chapter and subsequent chapters. In the absence of both an initial map and exact pose information, however, the robot has to do both: estimating the map and localizing itself relative to this map.

Of course, not all mapping problems are equally hard. The hardness of the mapping problem is the result of a collection of factors, the most important of which are:

- **Size.** The larger the environment relative to the robot's perceptual range, the more difficult it is to acquire a map.

- **Noise in perception and actuation.** If robot sensors and actuators were noise-free, mapping would be a simple problem. The larger the noise, the more difficult the problem.

- **Perceptual ambiguity.** The more frequently different places look alike, the more difficult it is to establish correspondence between different locations traversed at different points in time.

- **Cycles.** Cycles in the environment are particularly difficult to map. If a robot just goes up and down a corridor, it can correct odometry errors incrementally when coming back. Cycles make robots return via different paths, and when closing a cycle the accumulated odometric error can be huge!

To fully appreciate the difficulty of the mapping problem, consider Figure 9.1. Shown there is a data set, collected in a large indoor environment. Figure 9.1a was generated using the robot's raw odometry information. Each black dot in this figure corresponds to an obstacle detected by the robot's laser range finder. Figure 9.1b shows the result of applying mapping algorithms to this data, one of which was the techniques described in this chapter. This example gives a good flavor of the problem at stake.

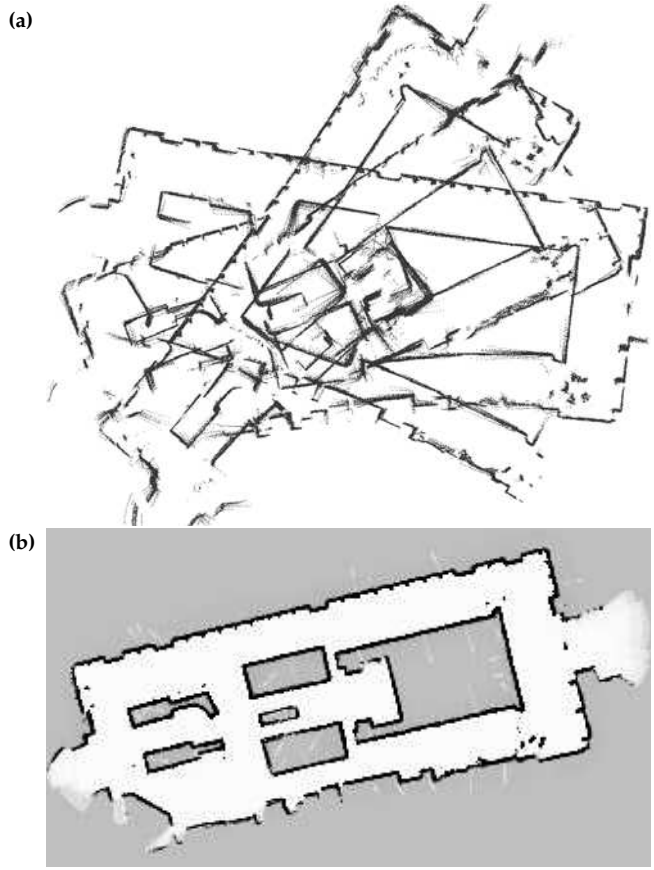


Figure 9.1 (a) Raw range data, position indexed by odometry. (b) Occupancy grid map.

In this chapter, we first study the mapping problem under the restrictive assumption that the robot poses are known. Put differently, we side-step the hardness of the SLAM problem by assuming some oracle informs us of the exact robot path during mapping. This problem, whose graphical model is depicted in Figure 9.2, is also known as *mapping with known poses*. We will discuss a popular family of algorithms, collectively called *occupancy grid mapping*. Occupancy grid mapping addresses the problem of generating consistent maps from noisy and uncertain measurement data, under the assumption that the robot pose is known. The basic idea of the occupancy

MAPPING WITH KNOWN POSES

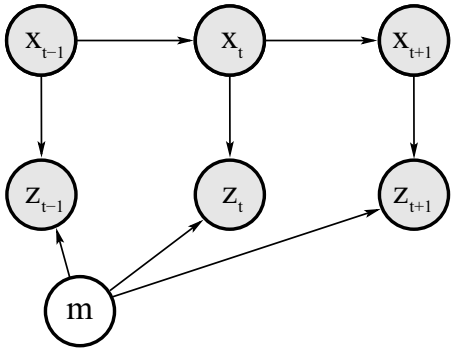


Figure 9.2 Graphical model of mapping with known poses. The shaded variables (poses x and measurements z) are known. The goal of mapping is to recover the map m.

grids is to represent the map as a field of random variables, arranged in an evenly spaced grid. Each random variable is binary and corresponds to the occupancy of the location it covers. Occupancy grid mapping algorithms implement approximate posterior estimation for those random variables.

The reader may wonder about the significance of a mapping technique that requires exact pose information. After all, no robot's odometry is perfect! The main utility of the occupancy grid technique is in post-processing: Many of the SLAM techniques discussed in subsequent chapters do not generate maps fit for path planning and navigation. Occupancy grid maps are often used after solving the SLAM problem by some other means, and taking the resulting path estimates for granted.

9.2 The Occupancy Grid Mapping Algorithm

The gold standard of any occupancy grid mapping algorithm is to calculate the posterior over maps given the data

$$p(m \mid z_{1:t}, x_{1:t}) \tag{9.1}$$

As usual, m is the map, $z_{1:t}$ the set of all measurements up to time t, and $x_{1:t}$ is the path of the robot defined through the sequence of all poses. The controls $u_{1:t}$ play no role in occupancy grid maps, since the path is already known. Hence, they will be omitted throughout this chapter.

The types of maps considered by occupancy grid maps are fine-grained

grids defined over the continuous space of locations. By far the most common domain of occupancy grid maps are 2-D floor plan maps, which describe a 2-D slice of the 3-D world. 2-D maps are usually the representation of choice when a robot navigates on a flat surface, and the sensors are mounted so that they capture only a slice of the world. Occupancy grid techniques generalize to 3-D representations but at significant computational expenses.

Let $\mathbf{m}_i$ denote the grid cell with index i. An occupancy grid map partitions the space into finitely many grid cells:

$$m = \{\mathbf{m}_i\} \tag{9.2}$$

Each $\mathbf{m}_i$ has attached to it a binary occupancy value, which specifies whether a cell is occupied or free. We will write "1" for occupied and "0" for free. The notation $p(\mathbf{m}_i = 1)$ or $p(\mathbf{m}_i)$ refers to the probability that a grid cell is occupied.

The problem with the posterior in Equation (9.1) is its dimensionality: the number of grid cells in maps like the one shown in Figure 9.1 are in the tens of thousands. For a map with 10,000 grid cells, the number of maps that can be represented by this map is $2^{10,000}$. Calculating a posterior probability for each single map is therefore intractable.

The standard occupancy grid approach breaks down the problem of estimating the map into a collection of separate problems, namely that of estimating

$$p(\mathbf{m}_i \mid z_{1:t}, x_{1:t}) \tag{9.3}$$

for all grid cell $\mathbf{m}_i$. Each of these estimation problems is now a binary problem with static state. This decomposition is convenient but not without problems. In particular, it does not enable us to represent dependencies among neighboring cells; instead, the posterior over maps is approximated as the product of its marginals:

$$p(m \mid z_{1:t}, x_{1:t}) = \prod_i p(\mathbf{m}_i \mid z_{1:t}, x_{1:t}) \tag{9.4}$$

We will return to this issue in Chapter 9.4 below, when we discuss more advanced mapping algorithms. For now, we will adopt this factorization for convenience.

Thanks to our factorization, the estimation of the occupancy probability for each grid cell is now a binary estimation problem with static state. A filter for this problem was already discussed in Chapter 4.2: the *binary Bayes filter*. The corresponding algorithm was depicted in Table 4.2 on page 94.

```
1:  Algorithm occupancy_grid_mapping({l_{t-1,i}}, x_t, z_t):
2:      for all cells m_i do
3:          if m_i in perceptual field of z_t then
4:              l_{t,i} = l_{t-1,i} + inverse_sensor_model(m_i, x_t, z_t) - l_0
5:          else
6:              l_{t,i} = l_{t-1,i}
7:          endif
8:      endfor
9:      return {l_{t,i}}
```

Table 9.1 The occupancy grid algorithm, a version of the binary Bayes filter in Table 4.2.

The algorithm in Table 9.1 applies this filter to the occupancy grid mapping problem. As in the original filter, our occupancy grid mapping algorithm uses the *log odds* representation of occupancy:

$$l_{t,i} = \log \frac{p(\mathbf{m}_i \mid z_{1:t}, x_{1:t})}{1 - p(\mathbf{m}_i \mid z_{1:t}, x_{1:t})} \tag{9.5}$$

This representation is already familiar from Chapter 4.2. The advantage of the log odds over the probability representation is that we can avoid numerical instabilities for probabilities near zero or one. The probabilities are easily recovered from the log odds ratio:

$$p(\mathbf{m}_i \mid z_{1:t}, x_{1:t}) = 1 - \frac{1}{1 + \exp\{l_{t,i}\}} \tag{9.6}$$

The algorithm **occupancy_grid_mapping** in Table 9.1 loops through all grid cells i, and updates those that fall into the sensor cone of the measurement z_t. For those where it does, it updates the occupancy value by virtue of the function **inverse_sensor_model** in line 4 of the algorithm. Otherwise, the occupancy value remains unchanged, as indicated in line 6. The constant l_0 is the prior of occupancy represented as a log odds ratio:

$$l_0 = \log \frac{p(\mathbf{m}_i = 1)}{p(\mathbf{m}_i = 0)} = \log \frac{p(\mathbf{m}_i)}{1 - p(\mathbf{m}_i)} \tag{9.7}$$

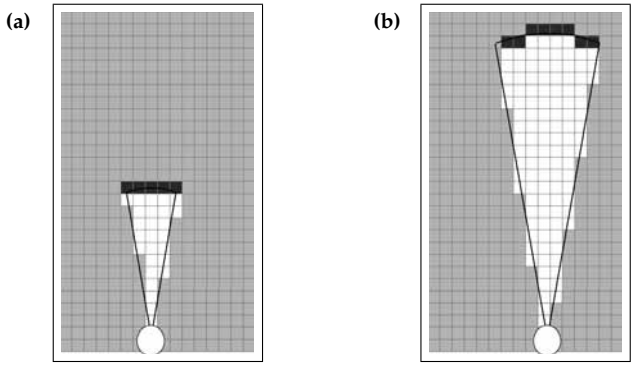


Figure 9.3 Two examples of an inverse measurement model **inverse_range_sensor_model** for two different measurement ranges. The darkness of each grid cell corresponds to the likelihood of occupancy. This model is somewhat simplistic; in contemporary implementations the occupancy probabilities are usually weaker at the border of the measurement cone.

The function **inverse_sensor_model** implements the inverse measurement model $p(\mathbf{m}_i \mid z_t, x_t)$ in its log odds form:

$$\textbf{inverse_sensor_model}(\mathbf{m}_i, x_t, z_t) \quad = \quad \log \frac{p(\mathbf{m}_i \mid z_t, x_t)}{1 - p(\mathbf{m}_i \mid z_t, x_t)} \tag{9.8}$$

A somewhat simplistic example of such a function for range finders is given in Table 9.2 and illustrated in Figure 9.3a&b. This model assigns to all cells within the sensor cone whose range is close to the measured range an occupancy value of l_{occ}. In Table 9.2, the width of this region is controlled by the parameter α, and the opening angle of the beam is given by β. We note that this model is somewhat simplistic; in contemporary implementations the occupancy probabilities are usually weaker at the border of the measurement cone.

The algorithm **inverse_sensor_model** calculates the inverse model by first determining the beam index k and the range r for the center-of-mass of the cell $\mathbf{m}_i$. This calculation is carried out in lines 2 through 5 in Table 9.2. As usual, we assume that the robot pose is given by $x_t = (x \;\; y \;\; \theta)^T$. In line 7, it returns the prior for occupancy in log odds form whenever the cell is outside the measurement range of this sensor beam, or if it lies more than $\alpha/2$ behind the detected range z_t^k. In line 9, it returns $l_{\text{occ}} > l_0$ if the range of the cell is within $\pm\alpha/2$ of the detected range z_t^k. It returns $l_{\text{free}} < l_0$ if the range to the

1: **Algorithm inverse_range_sensor_model($\mathbf{m}_i, x_t, z_t$):**
2: *Let x_i, y_i be the center-of-mass of $\mathbf{m}_i$*
3: $r = \sqrt{(x_i - x)^2 + (y_i - y)^2}$
4: $\phi = \operatorname{atan2}(y_i - y, x_i - x) - \theta$
5: $k = \operatorname{argmin}_j |\phi - \theta_{j,\text{sens}}|$
6: *if* $r > \min(z_{\max}, z_t^k + \alpha/2)$ *or* $|\phi - \theta_{k,\text{sens}}| > \beta/2$ *then*
7: *return* l_0
8: *if* $z_t^k < z_{\max}$ *and* $|r - z_t^k| < \alpha/2$
9: *return* l_{occ}
10: *if* $r \leq z_t^k$
11: *return* l_{free}
12: *endif*

Table 9.2 A simple inverse measurement model for robots equipped with range finders. Here α is the thickness of obstacles, and β the width of a sensor beam. The values l_{occ} and l_{free} in lines 9 and 11 denote the amount of evidence a reading carries for the two different cases.

cell is shorter than the measured range by more than $\alpha/2$. The left and center panel of Figure 9.3 illustrates this calculation for the main cone of a sonar beam.

A typical application of an inverse sensor model for ultrasound sensors is shown in Figure 9.4. Starting with an initial map the robot successively extends the map by incorporating local maps generated using the inverse model. A larger occupancy grid map obtained with this model for the same environment is depicted in Figure 9.5.

Figures 9.6 shows an example map next to a blueprint of a large open exhibit hall, and relates it to the occupancy map acquired by a robot. The map was generated using laser range data gathered in a few minutes. The gray-level in the occupancy map indicates the posterior of occupancy over an evenly spaced grid: The darker a grid cell, the more likely it is to be occupied. While occupancy maps are inherently probabilistic, they tend to quickly converge to estimates that are close to the two extreme posteriors, 1 and 0. In comparison between the learned map and the blueprint, the occupancy grid map shows all major structural elements, and obstacles as they were visi-

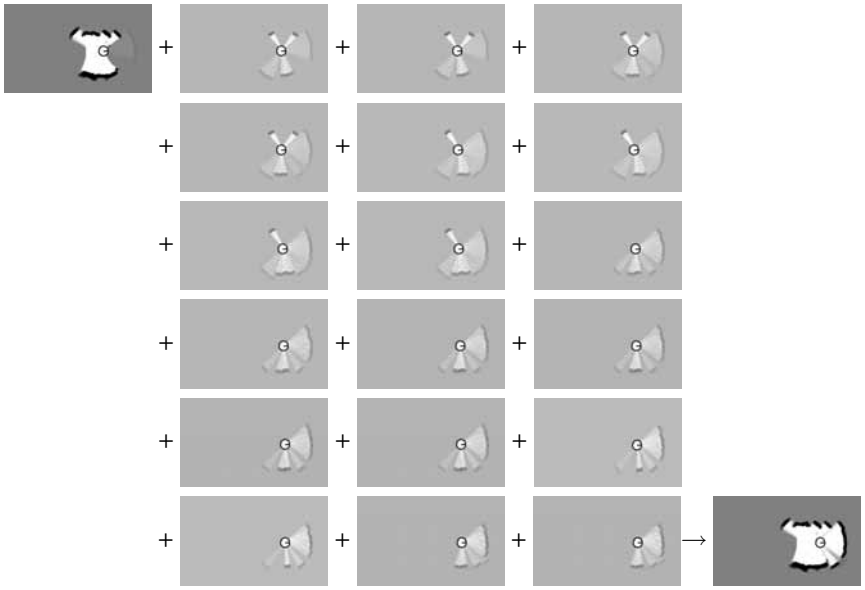


Figure 9.4 Incremental learning of an occupancy grid map using ultra-sound data in a corridor environment. The upper left image shows the initial map and the lower right image contains the resulting map. The maps in columns 2 to 4 are the local maps built from an inverse sensor model. Measurements beyond a 2.5m radius have not been considered. Each cone has an opening angle of 15 degrees. Images courtesy of Cyrill Stachniss, University of Freiburg.

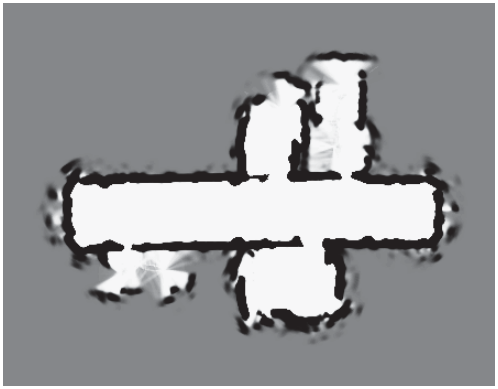


Figure 9.5 Occupancy probability map of an office environment built from sonar measurements. Courtesy of Cyrill Stachniss, University of Freiburg.

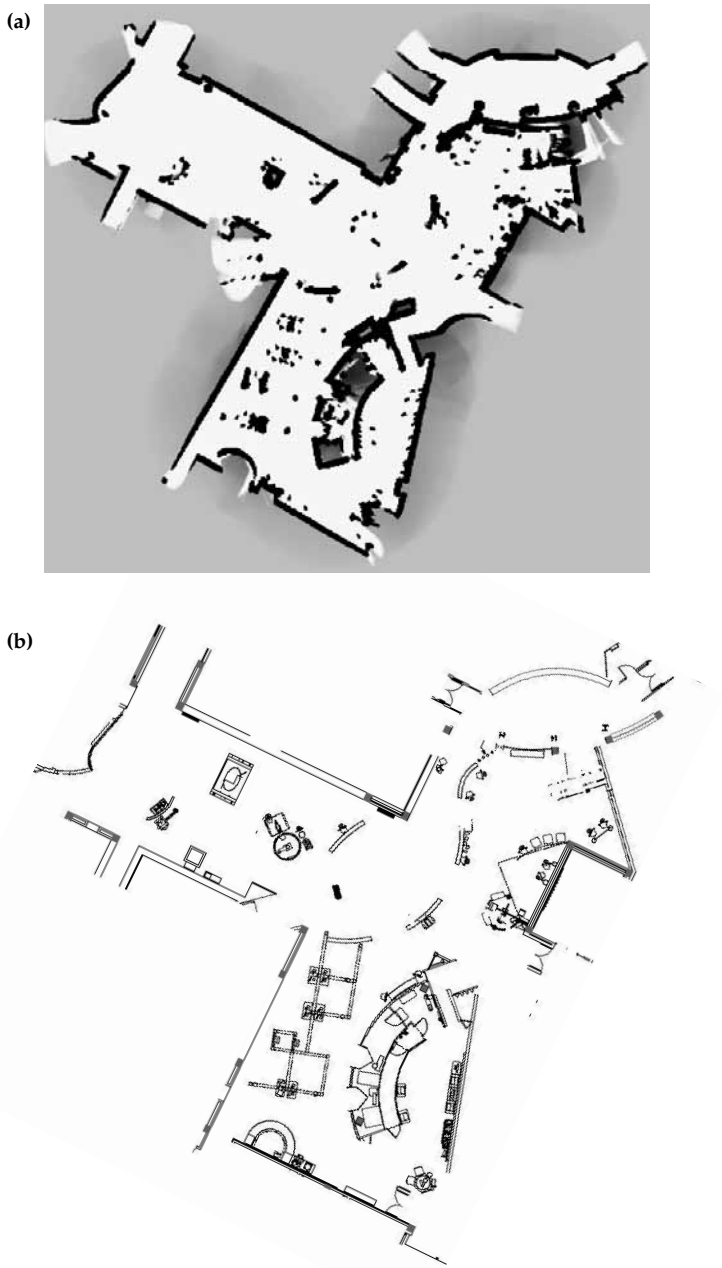


Figure 9.6 (a) Occupancy grid map and (b) architectural blue-print of a large open exhibit space. Notice that the blue-print is inaccurate in certain places.

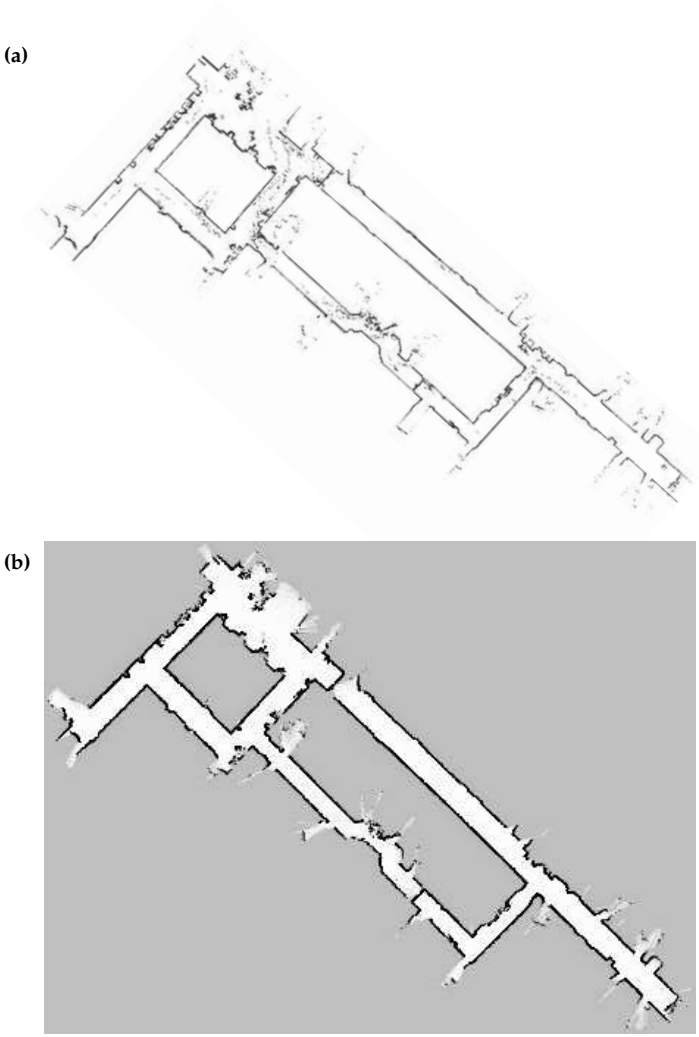


Figure 9.7 (a) Raw laser range data with corrected pose information. Each dot corresponds to a detection of an obstacle. Most obstacles are static (walls etc.), but some were dynamic, since people walked near the robot during data acquisition. (b) Occupancy grid map built from the data. The gray-scale indicates the posterior probability: Black corresponds to occupied with high certainty, and white to free with high certainty. The gray background color represents the prior. Figure (a) courtesy of Steffen Gutmann.

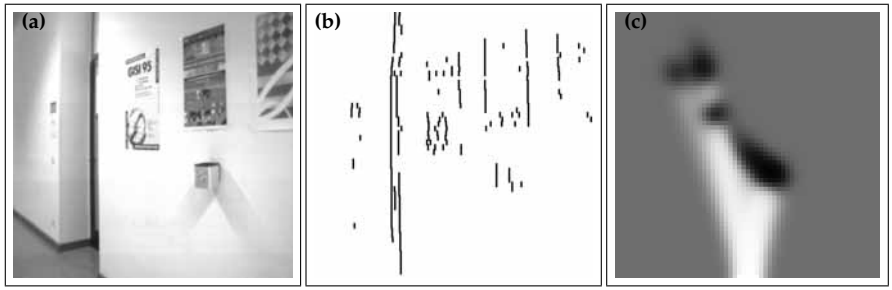


Figure 9.8 Estimation of occupancy maps using stereo vision: (a) camera image, (b) sparse disparity map, (c) occupancy map by projecting the disparity image onto the 2-D plane and convolving the result with a Gaussian. Images courtesy of Thorsten Fröhlinghaus.

ble at the height of the laser. Through a careful inspection, the reader may uncover some small discrepancies between the blueprint and the actual environment configuration.

Figure 9.7 compares a raw dataset with the occupancy grid maps generated from this data. The data in Panel (a) was preprocessed by a SLAM algorithm, so that the poses align. Some of the data is corrupted by the presence of people; the occupancy grid map filters out people quite nicely. This makes occupancy grid maps much better suited for robot navigation than sets of scan endpoint data: A planner fed the raw sensor endpoints would have a hard time finding a path through such scattered obstacles, even if the evidence that the corresponding cell is free outweighed that of it being occupied.

We note that our algorithm makes occupancy decisions exclusively based on sensor measurements. An alternative source of information is the space claimed by the robot itself: When the robot's pose is x_t, the region surrounding x_t must be navigable. Our inverse measurement algorithm in Table 9.2 can easily be modified to incorporate this information, by returning a large negative number for all grid cells occupied by a robot when at x_t. In practice, it is a good idea to incorporate the robot's volume when generating maps, especially if the environment is populated during mapping.

9.2.1 Multi-Sensor Fusion

Robots are often equipped with more than one type of sensor. Hence, a natural objective is to integrate information from more than one sensor into a single map. This question as to how to best integrate data from multiple sensors is particularly interesting if the sensors have different characteristics. For example, Figure 9.8 shows occupancy maps built with a stereo vision system, in which disparities are projected onto the plane and convolved with a Gaussian. Clearly, the characteristics of stereo are different from that of a sonar-based range finder. They are sensitive to different types of obstacles.

Unfortunately, fusing data from multiple sensors with Bayes filters is not an easy endeavor. A naive solution is to execute algorithm **occupancy_grid_mapping** in Table 9.1 with different sensor modalities. However, such an approach has a clear drawback. If different sensors detect different types of obstacles, the result of Bayes filtering is ill-defined. Consider, for example, an obstacle that can be recognized by one sensor type but not by another. Then these two sensor types will generate conflicting information, and the resulting map will depend on the amount of evidence brought by every sensor system. This is generally undesirable, since whether or not a cell is considered occupied depends on the relative frequency at which different sensors are polled.

A popular approach to integrating information from multiple sensors is to build separate maps for each sensor type, and integrate them using an appropriate combination function. Let $m^k = \{\mathbf{m}_i^k\}$ denote the map built by the k-th sensor type. If the measurements of the sensors are independent of each other we can directly combine them using *De Morgan's law*

$$p(\mathbf{m}_i) = 1 - \prod_k \left(1 - p(\mathbf{m}_i^k)\right) \tag{9.9}$$

Alternatively, one can compute the maximum

$$p(\mathbf{m}_i) = \max_k \; p(\mathbf{m}_i^k) \tag{9.10}$$

of all maps, which yields the most pessimistic estimates of its components. If any of the sensor-specific maps show that a grid cell is occupied, so will the combined map.

9.3 Learning Inverse Measurement Models

9.3.1 Inverting the Measurement Model

The occupancy grid mapping algorithm requires a marginalized *inverse measurement model*, $p(\mathbf{m}_i \mid x, z)$. This probability is called "inverse" since it reasons from effects to causes: it provides information about the world conditioned on a measurement caused by this world. It is marginalized for the i-th grid cell; a full inverse would be of the type $p(m \mid x, z)$. In our exposition of the basic algorithm, we already provided an ad hoc procedure in Table 9.2 for implementing such an inverse model. This raises the question as to whether we can obtain an inverse model in a more principled manner, starting at the conventional measurement model.

The answer is positive but less straightforward than one might assume at first glance. Bayes rule suggests

$$\begin{aligned} p(m \mid x, z) &= \frac{p(z \mid x, m)\, p(m \mid x)}{p(z \mid x)} \\ &= \eta\, p(z \mid x, m)\, p(m) \end{aligned} \tag{9.11}$$

Here we silently assume $p(m \mid x) = p(m)$, hence the pose of the robot tells us nothing about the map—an assumption that we will adopt for sheer convenience. If our goal was to calculate the inverse model for the entire map at-a-time, we would now be done. However, our occupancy grid mapping algorithm approximates the posterior over maps by its marginals, one for each grid cell $\mathbf{m}_i$. The inverse model for the i-th grid cell is obtained by selecting the marginal for the i-th grid cell:

$$p(\mathbf{m}_i \mid x, z) = \eta \sum_{m : m(i) = \mathbf{m}_i} p(z \mid x, m)\, p(m) \tag{9.12}$$

This expression sums over all maps m for which the occupancy value of grid cell i equals $\mathbf{m}_i$. Clearly, this sum cannot be computed, since the space of all maps is too large.

We will now describe an algorithm for approximating this expression. The algorithm involves generating samples from the measurement model, and approximating the inverse using a *supervised learning algorithm*, such as *logistic regression* or a *neural network*.

SUPERVISED LEARNING ALGORITHM

9.3.2 Sampling from the Forward Model

The basic idea is simple and quite universal: If we can generate random triplets of poses $x_t^{[k]}$, measurements $z_t^{[k]}$, and map occupancy values $\mathbf{m}_i^{[k]}$ for any grid cell $\mathbf{m}_i$, we can learn a function that accepts a pose x and measurement z as an input, and outputs the probability of occupancy for $\mathbf{m}_i$.

A sample of the form $(x_t^{[k]} \;\; z_t^{[k]} \;\; \mathbf{m}_i^{[k]})$ can be generated by the following procedure.

1. Sample a random map $m^{[k]} \sim p(m)$. For example, one might already have a database of maps that represents $p(m)$ and randomly draws a map from the database.

2. Sample a pose $x_t^{[k]}$ inside the map. One may safely assume that poses are uniformly distributed.

3. Sample a measurement $z_t^{[k]} \sim p(z \mid x_t^{[k]}, m^{[k]})$. This sampling step is reminiscent of a robot simulator that stochastically simulates a sensor measurement.

4. Extract the desired "true" occupancy value $\mathbf{m}_i$ for the target grid cell from the map m.

The result is a sampled pose $x_t^{[k]}$, a measurement $z_t^{[k]}$, and the occupancy value of the grid cell $\mathbf{m}_i$. Repeated application of this sampling step yields a data set

$$\begin{array}{cccc} x_t^{[1]} & z_t^{[1]} & \longrightarrow & \mathrm{occ}(\mathbf{m}_i)^{[1]} \\ x_t^{[2]} & z_t^{[2]} & \longrightarrow & \mathrm{occ}(\mathbf{m}_i)^{[2]} \\ x_t^{[3]} & z_t^{[3]} & \longrightarrow & \mathrm{occ}(\mathbf{m}_i)^{[3]} \\ \vdots & \vdots & & \vdots \end{array}$$

TRAINING EXAMPLES

These triplets may serve as *training examples* for the supervised learning algorithm, which approximates the desired conditional probability $p(\mathbf{m}_i \mid z, x)$. Here the measurements z and the pose x are input variables, and the occupancy value $\mathrm{occ}(\mathbf{m}_i)$ is a target for the output of the learning algorithm.

This approach is somewhat inefficient, since it fails to exploit a number of properties that we know to be the case for the inverse sensor model.

- Measurements should carry no information about grid cells far outside their perceptual range. This observation has two implications: First, we can focus our sample generation process on triplets where the cell $\mathbf{m}_i$ is

indeed inside the measurement cone. And second, when making a prediction for this cell, we only have to include a subset of the data in a measurement z (e.g., nearby beams) as input to the learning algorithm.

- The characteristics of a sensor are invariant with respect to the absolute coordinates of the robot or the grid cell when taking a measurement. Only the relative coordinates matter. If we denote the robot pose by $x_t = (x \;\; y \;\; \theta)^T$ and the coordinates of the grid cell by $\mathbf{m}_i = (x_{\mathbf{m}_i} \;\; y_{\mathbf{m}_i})^T$, the coordinates of the grid cell are mapped into the robot's local reference frame via the following translation and rotation:

  $$\begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x_{\mathbf{m}_i} - x \\ y_{\mathbf{m}_i} - y \end{pmatrix}$$

  In robots with circular arrays of range finders, it makes sense to encode the relative location of a grid cell using the familiar polar coordinates (range and bearing).

- Nearby grid cells should have a similar interpretation under the inverse sensor model. This smoothness suggests that it may be beneficial to learn a single function in which the coordinates of the grid cell function as an input, rather than learning a separate function for each grid cell.

- If the robot possesses functionally identical sensors, the inverse sensor model should be interchangeable for different sensors. For robots equipped with a circular array of range sensors, any of the resulting sensor beams are characterized by the same inverse sensor model.

The most basic way to enforce these invariances is to constrain the learning algorithm by choosing appropriate input variables. A good choice is to use relative pose information, so that the learning algorithm cannot base its decision on absolute coordinates. It is also a good idea to omit sensor measurements known to be irrelevant to occupancy predictions, and to confine the prediction to grid cells inside the perceptual field of a sensor. By exploiting these invariances, the training set size can be reduced significantly.

9.3.3 The Error Function

To train the learning algorithm, we need an approximate error function. A popular example are artificial neural networks trained with the Backpropagation algorithm. Backpropagation trains *neural networks* by *gradient descent*

BACKPROPAGATION

in parameter space. Given an error function that measures the "mismatch" between the network's actual and desired output, Backpropagation calculates the first derivative of the target function and the parameters of the neural network, and then adapts the parameters in opposite direction of the gradient so as to diminish the mismatch. This raises the question as to what error function to use.

A common approach is to train the learning algorithm so as to maximize the log-likelihood of the training data. More specifically we are given a training set of the form

$$\begin{array}{ccc} \mathrm{input}^{[1]} & \longrightarrow & \mathrm{occ}(\mathbf{m}_i)^{[1]} \\ \mathrm{input}^{[2]} & \longrightarrow & \mathrm{occ}(\mathbf{m}_i)^{[2]} \\ \mathrm{input}^{[3]} & \longrightarrow & \mathrm{occ}(\mathbf{m}_i)^{[3]} \\ \vdots & & \vdots \end{array} \tag{9.13}$$

$occ(\mathbf{m}_i)^{[k]}$ is the k-th sample of the desired conditional probability, and $\mathrm{input}^{[k]}$ is the corresponding input to the learning algorithm. Clearly, the exact form of the input may vary as a result of the encoding known invariances, but the exact nature of this vector will play no role in the form of the error function.

Let us denote the parameters of the learning algorithm by W. Assuming that each individual item in the training data has been generated independently, the likelihood of the training data is now

$$\prod_i p(\mathbf{m}_i^{[k]} \mid \mathrm{input}^{[k]}, W) \tag{9.14}$$

and its negative logarithm is

$$J(W) \quad = \quad -\sum_i \log p(\mathbf{m}_i^{[k]} \mid \mathrm{input}^{[k]}, W) \tag{9.15}$$

Here J defines the function we seek to minimize during training.

Let us denote the learning algorithm by $f(\mathrm{input}^{[k]}, W)$. The output of this function is a value in the interval $[0; 1]$. After training, we want the learning algorithm to output the probability of occupancy:

$$p(\mathbf{m}_i^{[k]} \mid \mathrm{input}^{[k]}, W) \quad = \quad \begin{cases} f(\mathrm{input}^{[k]}, W) & \text{if } \mathbf{m}_i^{[k]} = 1 \\ 1 - f(\mathrm{input}^{[k]}, W) & \text{if } \mathbf{m}_i^{[k]} = 0 \end{cases} \tag{9.16}$$

Thus, we seek an error function that adjusts W so as to minimize the deviation of this predicted probability and the one communicated by the training

example. To find such an error function, we re-write (9.16) as follows:

$$p(\mathbf{m}_i^{[k]} \mid \text{input}^{[k]}, W) = f(\text{input}^{[k]}, W)^{\mathbf{m}_i^{[k]}} \; (1 - f(\text{input}^{[k]}, W))^{1-\mathbf{m}_i^{[k]}} \tag{9.17}$$

It is easy to see that this product and Expression (9.16) are identical. In the product, one of the terms is always 1, since its exponent is zero. Substituting the product into (9.15) and multiplying the result by minus one gives us the following function:

$$\begin{aligned} J(W) &= -\sum_i \log \left[f(\text{input}^{[k]}, W)^{\mathbf{m}_i^{[k]}} \; (1 - f(\text{input}^{[k]}, W))^{1-\mathbf{m}_i^{[k]}} \right] \\ &= -\sum_i \mathbf{m}_i^{[k]} \log f(\text{input}^{[k]}, W) + (1 - \mathbf{m}_i^{[k]}) \log(1 - f(\text{input}^{[k]}, W)) \end{aligned} \tag{9.18}$$

$J(W)$ is the error function to minimize when training the learning algorithm. It is easily folded into any algorithm that uses gradient descent to adjusts its parameters.

9.3.4 Examples and Further Considerations

Figure 9.9 shows the result of an artificial neural network trained to mimic the inverse sensor model. The robot in this example is equipped with a circular array of sonar range sensors mounted at approximate table height. The input to the network are the relative range and bearing of a target cell, along with the set of five adjacent range measurements. The output is a probability of occupancy: the darker a cell, the more likely it is occupied. As this example illustrates, the approach correctly learns to distinguish freespace from occupied space. The uniformly gray area behind obstacles matches the prior probability of occupancy, which leads to no change when used in the occupancy grid mapping algorithm. Figure 9.9b contains a faulty short reading on the bottom left. Here a single reading seems to be insufficient to predict an obstacle with high probability.

We note that there exists a number of ways to train a function approximator using actual data collected by a robot, instead of the simulated data from the forward model. In general, this is the most accurate data one can use for learning, since the measurement model is necessarily just an approximation. One such way involves a robot operating in a known environment with a known map. With Markov localization, we can localize the robot, and then use its actual recorded measurements and the known map occupancy to assemble training examples. It is even possible to start with an approxi-

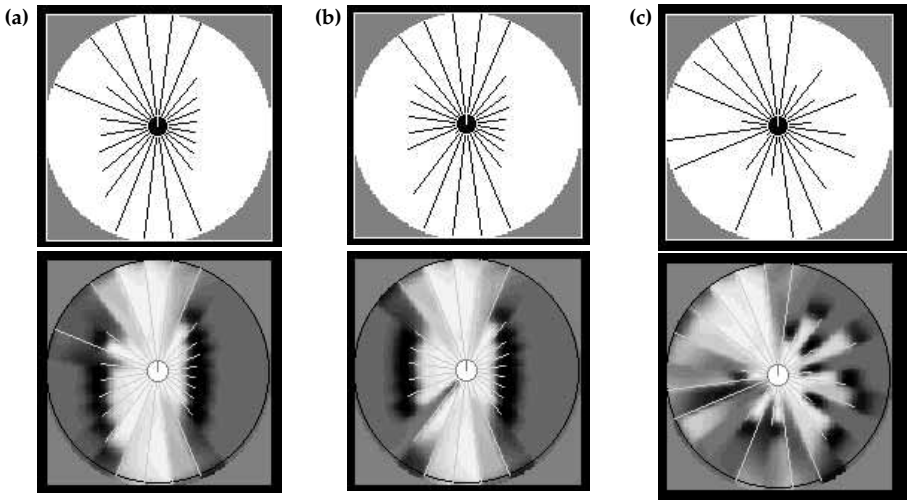


Figure 9.9 Inverse sensor model learned from data: Three sample sonar scans (top row) and local occupancy maps (bottom row), as generated by the neural network. Bright regions indicate free-space, and dark regions indicate walls and obstacles (enlarged by a robot diameter).

mate map, use the learned sensor model to generate a better map, and use the procedure just outlined to improve the inverse measurement model.

9.4 Maximum A Posteriori Occupancy Mapping

9.4.1 The Case for Maintaining Dependencies

In the remainder of this chapter, we will return to one of the very basic assumptions of the occupancy grid mapping algorithm. In Chapter 9.2, we assumed that we can safely decompose the map inference problem defined over high-dimensional space of all maps, into a collection of single-cell mapping problems. This assumption culminated into the factorization in (9.4):

$$p(m \mid z_{1:t}, x_{1:t}) \quad = \quad \prod_i p(\mathbf{m}_i \mid z_{1:t}, x_{1:t}) \tag{9.19}$$

This raises the question as to how faithful we should be in the result of any algorithm that relies on such a strong decomposition.

Figure 9.10 illustrates a problem that arises directly as a result of this factor-

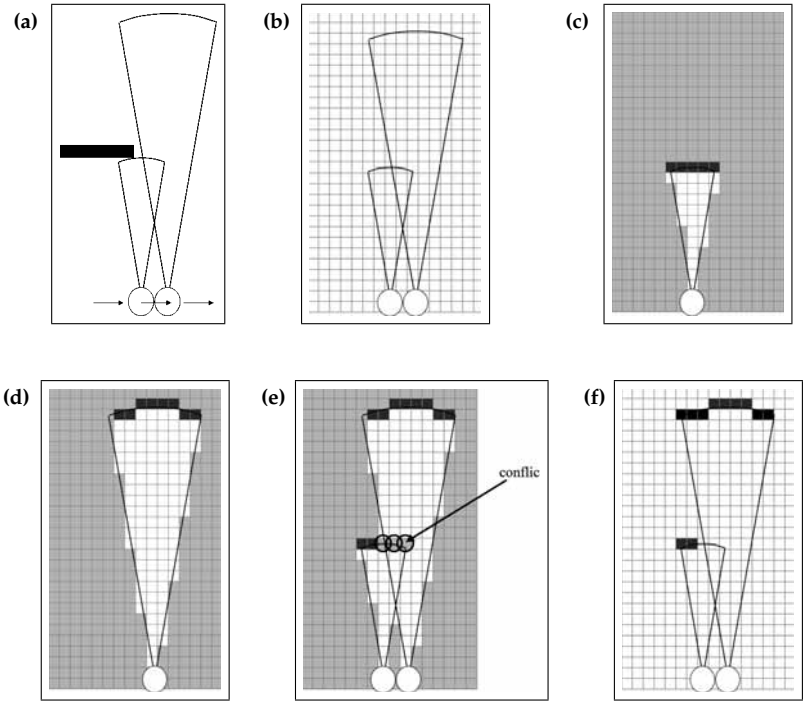


Figure 9.10 The problem with the standard occupancy grid mapping algorithm in Chapter 9.2: For the environment shown in Figure (a), a passing robot might receive the (noise-free) measurement shown in (b). The factorial approach maps these beams into probabilistic maps separately for each grid cell and each beam, as shown in (c) and (d). Combining both interpretations yields the map shown in (e). Obviously, there is a conflict in the overlap region, indicated by the circles in (e). The interesting insight is: There exist maps, such as the one in diagram (f), that perfectly explain the sensor measurement without any such conflict. For a sensor reading to be explained, it suffices to assume an obstacle *somewhere* in the cone of a measurement, and not everywhere.

ization. Shown there is a situation in which the robot facing a wall receives two noise-free sonar range measurements. Because the factored approach predicts an object along the entire arc at the measured range, the occupancy values of all grid cells along this arc are increased. When combining the two different measurements shown in Figure 9.10c&d, a conflict is created, as shown in Figure 9.10e. The standard occupancy grid mapping algorithm

```
Algorithm MAP_occupancy_grid_mapping(x_{1:t}, z_{1:t}):
    set m = {0}
    repeat until convergence
        for all cells m_i do
            m_i = argmax_{k=0,1} k l_0 + Σ_t log
                    measurement_model(z_t, x_t, m with m_i = k)
        endfor
    endrepeat
    return m
```

Table 9.3 The maximum a posteriori occupancy grid algorithm, which uses conventional measurement models instead of inverse models.

"resolves" this conflict by summing up positive and negative evidence for occupancy; however, the result will reflect the relative frequencies of the two types of measurements, which is undesirable.

However, there exist maps, such as the one in Figure 9.10f, that perfectly explain the sensor measurements without any such conflict. This is because for a sensor reading to be explained, it suffices to assume an obstacle *somewhere* in its measurement cone. Put differently, the fact that cones sweep over multiple grid cells induces important dependencies between neighboring grid cells. When decomposing the mapping into thousands of individual grid cell estimation problems, we lose the ability to consider these dependencies.

9.4.2 Occupancy Grid Mapping with Forward Models

These dependencies are incorporated by an algorithm that outputs the mode of the posterior, instead of the full posterior. The mode is defined as the maximum of the logarithm of the map posterior:

$$m^* = \operatorname*{argmax}_m \log p(m \mid z_{1:t}, x_{1:t}) \tag{9.20}$$

The map posterior factors into a map prior and a measurement likelihood (c.f., Equation (9.11)):

$$\log p(m \mid z_{1:t}, x_{1:t}) \quad = \quad \text{const.} \; + \; \log p(z_{1:t} \mid x_{1:t}, m) \; + \; \log p(m) \tag{9.21}$$

The log-likelihood $\log p(z_{1:t} \mid x_{1:t}, m)$ decomposes into a sum of individual measurement log-likelihoods:

$$\log p(z_{1:t} \mid x_{1:t}, m) \quad = \quad \sum \log p(z_t \mid x_t, m) \tag{9.22}$$

Further, the log-prior also decomposes. To see, we note that the prior probability of any map m is given by the following product:

$$\begin{aligned} p(m) &= \prod_i p(\mathbf{m})^{\mathbf{m}_i} \; (1 - p(\mathbf{m}))^{1-\mathbf{m}_i} \\ &= (1 - p(\mathbf{m}))^N \; \prod_i p(\mathbf{m})^{\mathbf{m}_i} \; (1 - p(\mathbf{m}))^{-\mathbf{m}_i} \\ &= \eta \; \prod_i p(\mathbf{m})^{\mathbf{m}_i} \; (1 - p(\mathbf{m}))^{-\mathbf{m}_i} \end{aligned} \tag{9.23}$$

Here $p(\mathbf{m})$ is the prior probability of occupancy (e.g., $p(\mathbf{m}) = 0.5$), and N is the number of grid cells in the map. The expression $(1 - p(\mathbf{m}))^N$ is simply a constant, which is replaced by our generic symbol η as usual.

This implies for the log version of the prior:

$$\begin{aligned} \log p(m) &= \text{const.} + \sum_i \mathbf{m}_i \; \log p(\mathbf{m}) \; - \mathbf{m}_i \; \log(1 - p(\mathbf{m})) \\ &= \text{const.} + \sum_i \mathbf{m}_i \; \log \frac{p(\mathbf{m})}{1 - p(\mathbf{m})} \\ &= \text{const.} + \sum_i \mathbf{m}_i \; l_0 \end{aligned} \tag{9.24}$$

The constant l_0 is adopted from (9.7). The term $M \log(1 - p(\mathbf{m}_i))$ is obviously independent of the map. Hence it suffices to optimize the remaining expression and the data log-likelihood:

$$m^* \quad = \quad \operatorname*{argmax}_m \sum_t \log p(z_t \mid x_t, m) + l_0 \; \sum_i \mathbf{m}_i \tag{9.25}$$

A hill-climbing algorithm for maximizing this log-probability is provided in Table 9.3. This algorithm starts with the all-free map (line 2). It "flips" the occupancy value of a grid cell when such a flip increases the likelihood of the data (lines 4-6). For this algorithm it is essential that the prior of occupancy

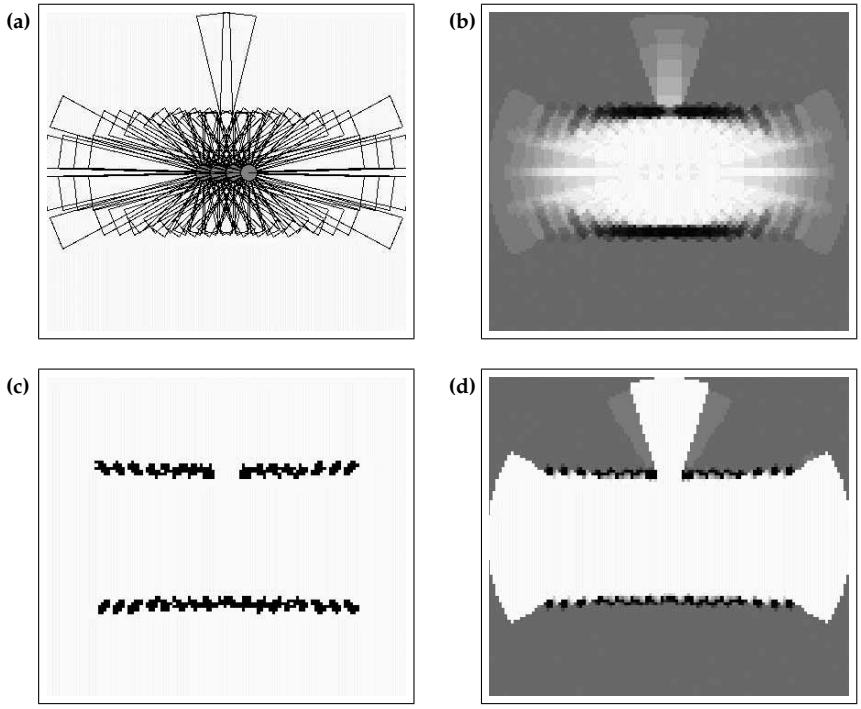


Figure 9.11 (a) Sonar range measurements from a noise-free simulation. (b) Results of the standard occupancy mapper, lacking the open door. (c) A maximum a posterior map. (d) The residual uncertainty in this map, obtained by measuring the sensitivity of the map likelihood function with respect to individual grid cells. This map clearly shows the door, and it also contains flatter walls at both ends.

$p(\mathbf{m}_i)$ is not too close to 1; otherwise it might return an all-occupied map. As any hill climbing algorithm, this approach is only guaranteed to find a local maximum. In practice, there are usually very few, if any, local maxima.

Figure 9.11 illustrates the effect of the MAP occupancy grid algorithm. Figure 9.11a depicts a noise-free data set of a robot passing by an open door. Some of the sonar measurements detect the open door, while others are reflected at the door post. The standard occupancy mapping algorithm with inverse models fails to capture the opening, as shown in Figure 9.11b. The mode of the posterior is shown in Figure 9.11c. This map models the open door correctly, hence it is better suited for robot navigation than the standard

occupancy grid map algorithm. Figure 9.11d shows the residual uncertainty of this map. This diagram is the result of a cell-wise sensitivity analysis: The magnitude by which flipping a grid cell decreases the log-likelihood function is illustrated by the grayness of a cell. This diagram, similar in appearance to the regular occupancy grid map, suggests maximum uncertainty for grid cells behind obstacles.

There exists a number of limitations of the algorithm **MAP_occupancy_grid_mapping**, and it can be improved in multiple ways. The algorithm is a maximum a posteriori approach, and as such returns no notion of uncertainty in the residual map. Our sensitivity analysis approximates this uncertainty, but this approximation is overconfident, since sensitivity analysis only inspects the mode locally. Further, the algorithm is a batch algorithm and cannot be executed incrementally. In fact, the MAP algorithm requires that all data is kept in memory. At the computational end, the algorithm can be sped up by initializing it with the result of the regular occupancy grid mapping approach, instead of an empty map. Finally, we note that only a small number of measurements are affected by flipping a grid cell in line 5 of Table 9.3. While each sum is potentially huge, only a small number of elements has to be inspected when calculating the argmax. This property can be exploited in the basic algorithm, to increase its computational efficiency.

9.5 Summary

This chapter introduced algorithms for learning occupancy grids. All algorithms in this chapter require exact pose estimates for the robot, hence they do not solve the general mapping problem.

- The standard occupancy mapping algorithm estimates for each grid cell individually the posterior probability of occupancy. It is an adaptation of the binary Bayes filter for static environments.
- Data from multiple sensors can be fused into a single map in two ways: By maintaining a single map using Bayes filters, and by maintaining multiple maps, one for each sensor modality. The latter approach is usually preferable when different sensors are sensitive to different types of obstacles.
- The standard occupancy grid mapping algorithm relies on inverse measurement models, which reason from effects (measurements) to causes (occupancy). This differs from previous applications of Bayes filters in the

context of localization, where the Bayes filter was based on a conventional measurement model that reasons from causes to effects.

- It is possible to learn inverse sensor models from the conventional measurement model, which models the sensor from causes to effects. To do so, one has to generate samples and learn an inverse model using a supervised learning algorithm.
- The standard occupancy grid mapping algorithm does not maintain dependencies in the estimate of occupancy. This is a result of decomposing the map posterior estimation problem into a large number of single-cell posterior estimation problem.
- The full map posterior is generally not computable, due to the large number of maps that can be defined over a grid. However, it can be maximized. Maximizing it leads to maps that are more consistent with the data than the occupancy grid algorithm using Bayes filters. However, the maximization requires the availability of all data, and the resulting maximum a posterior map does not capture the residual uncertainty in the map.

Without a doubt, occupancy grid maps and their various extensions are vastly popular in robotics. This is because they are extremely easy to acquire, and they capture important elements for robot navigation.

9.6 Bibliographical Remarks

Occupancy grid maps are due to Elfes (1987), whose Ph.D. thesis (1989) defined the field. A well-referenced article by Moravec (1988) provides a highly accessible introduction into this topic, and lays out the basic probabilistic approach that forms the core of this chapter. In unpublished work, Moravec and Martin (1994) extended occupancy grid maps to 3-D, using stereo as the primary sensor. Multi-sensor fusion in occupancy grids were introduced in Thrun et al. (1998a). The results for learning inverse sensor models described in this chapter can be found in Thrun (1998b). The forward modeling approach, also described in this chapter, is based on a similar algorithm in Thrun (2003).

Occupancy maps have been used for a number of different purposes. Borenstein and Koren (1991) were the first to adopt occupancy grid maps for collision avoidance. A number of authors have used occupancy grid maps for localization, by cross-matching two occupancy grid maps. Such "map matching" algorithms are discussed in detail in Chapter 7. Biswas et al. (2002) used occupancy grid maps to learn shape models of movable objects in dynamic environments. This approach was later extended to learning hierarchical class models of dynamic objects, all represented with occupancy grid maps (Anguelov et al. 2002). Occupancy grid maps have also extensively been used in the context of the simultaneous localization and mapping problem. Those applications will be discussed in later chapters.

Figure 9.12 Mobile indoor robot of the type RWI B21, with 24 sonar sensors mounted on a circular array around the robot.

The idea of representing space by grids is only one out of many ideas explored in the mobile robotics literature. Classical work on motion planning often assumes that the environment is represented by polygons, but leaves open as to how those models are being acquired from data (Schwartz et al. 1987). An early proposal on learning polygonal maps is due to Chatila and Laumond (1985). A first implementation using Kalman filters for fitting lines from sonar data was done by Crowley (1989). In more recent work, Anguelov et al. (2004) devised techniques for identifying straight-line doors from raw sensor data, and learn visual attributes for improving the door detection rate.

An early paradigm in spatial representation is the topological paradigm, in which space is represented by a set of local relations, often corresponding to specific actions a robot may have to take to navigate between adjacent locations. Examples of topological mapping algorithm include Kuipers and Levitt's (1988) work on their *Spatial Semantic hierarchy* (see also Kuipers et al. (2004)); Matarić's (1990) M.Sc. thesis work, Kortenkamp and Weymouth's (1994) work on topological graphs contracted from sonar and vision data, and Shatkay and Kaelbling's (1997) approach on spatial HMMs with arc length information. Occupancy grid maps are members of the complimentary paradigm: metric representations. Metric representations directly describe the robot environment, in some absolute coordinate system. A second example of a metric approach is the EKF SLAM algorithm, which will be discussed in the subsequent chapter.

SPATIAL SEMANTIC HIERARCHY

There is a history of attempts to generate mapping algorithms that harvest the best of both paradigms, topological and metric. Tomatis et al. (2002) uses topological representations to close loops consistently, then converts to metric maps. Thrun (1998b) first builds a metric occupancy grid map, then extracts a topological skeleton to facilitate fast motion planning. In Chapter 11, we will study techniques that bridge both paradigms, metric and topological.

9.7 Exercises

1. Change the basic occupancy grid algorithm in Table 9.1 to include a provision for the change of occupancy over time. To accommodate such change, evidence collected Δt time steps in the past should be decayed by a factor of $\alpha^{\Delta t}$, for some value of $\alpha < 1$ (e.g., $\alpha = 0.99$). Such a rule is called *exponential decay*. State the exponential decay occupancy grid mapping algorithm in log odds form and argue its correctness. If you cannot find an exact algorithm, state an approximation and argue why it is a suitable approximation. For simplicity, you might want to assume a prior $p(\mathbf{m}_i) = 0.5$ for occupancy.

EXPONENTIAL DECAY

2. The binary Bayes filter assumes that a cell is either occupied or unoccupied, and the sensor provides noisy evidence for the correct hypothesis. In this question, you will be asked to build an alternative estimator for a grid cell: Suppose the sensor can only measure "0 = unoccupied" or "1 = occupied", and it receives a sequence

   0, 0, 1, 0, 1, 1, 1, 0, 1, 0.

   What is the maximum likelihood probability p for the next reading to be 1? Provide an incremental formula for a general maximum likelihood estimator for this probability p. Discuss the difference of this estimator to the binary Bayes filter (all for a single cell only).

3. We study a common sensor configuration in indoor robotics. Suppose an indoor robot uses sonar sensors with a 15 degree opening cone, mounted at a fixed height so that they point out horizontally and parallel to the ground. Figure 9.12 shows such a robot. Discuss, what happens when the robot faces an obstacle whose height is just below the height of the sensor (for example, 15 cm below). Specifically, answer the following questions.

   (a) Under what conditions will the robot detect the obstacle? Under what conditions will it fail to detect it? Be concise.

   (b) What implications does this all have for the binary Bayes filter and the underlying Markov assumption? How can you make the occupancy grid algorithm fail?

   (c) Based on your answer to the previous question, can you provide an improved occupancy grid mapping algorithm that will detect the obstacle more reliably than the plain occupancy grid mapping algorithm?

4. In this question, you are being asked to design a simple sensor model. Suppose you are given binary occupancy measurements for the following four cells:

cell number	type	measurement sequence						
cell 1	occupied	1	1	0	1	0	1	1
cell 2	occupied	0	1	1	1	0	0	1
cell 3	free	0	0	0	0	0	0	0
cell 4	free	1	0	0	1	0	0	0

What is the maximum likelihood measurement model $p(z \mid \mathbf{m}_i)$? (Hint: $\mathbf{m}_i$ is a binary occupancy variable, and z is a binary measurement variable.)

5. For the table in Exercise 4, implement the basic occupancy grid algorithm.

   (a) What is the posterior $p(\mathbf{m}_i \mid z_{1:7})$ for the four different cases, assuming a prior $p(\mathbf{m}_i) = 0.5$?

   (b) Devise a tuning algorithm for your sensor model that makes the output of your occupancy grid mapping algorithm as close as possible to the ground truth, for the four cases in Exercise 4. What do you find? (For this question, you will have to come up with a suitable closeness measure.)

6. The standard occupancy grid mapping algorithm is implemented using the log odds form, even though it would have equally been implementable using probabilities.

   (a) Derive an update rule that represented occupancy probabilities directly, without the detour of the log odds representation.

   (b) For an implementation in a common programming language such as C++, give an example in which the probability implementation yields *different* results form the log odds implementation, due to numerical truncation. Explain your example, and argue whether you judge this to be a problem in practice.

10 *Simultaneous Localization and Mapping*

10.1 Introduction

This and the following chapters address one of the most fundamental problems in robotics, the *simultaneous localization and mapping problem*. This problem is commonly abbreviated as *SLAM*, and is also known as *Concurrent Mapping and Localization*, or *CML*. SLAM problems arise when the robot does not have access to a map of the environment, nor does it know its own pose. Instead, all it is given are measurements $z_{1:t}$ and controls $u_{1:t}$. The term "simultaneous localization and mapping" describes the resulting problem: In SLAM, the robot acquires a map of its environment while simultaneously localizing itself relative to this map. SLAM is significantly more difficult than all robotics problems discussed thus far. It is more difficult than localization in that the map is unknown and has to be estimated along the way. It is more difficult than mapping with known poses, since the poses are unknown and have to be estimated along the way.

From a probabilistic perspective, there are two main forms of the SLAM problem, which are both of equal practical importance. One is known as the *online SLAM problem*: It involves estimating the posterior over the momentary pose along with the map:

ONLINE SLAM PROBLEM

$$p(x_t, m \mid z_{1:t}, u_{1:t}) \tag{10.1}$$

Here x_t is the pose at time t, m is the map, and $z_{1:t}$ and $u_{1:t}$ are the measurements and controls, respectively. This problem is called the online SLAM problem since it only involves the estimation of variables that persist at time t. Many algorithms for the online SLAM problem are incremental: they discard past measurements and controls once they have been processed. The graphical model of online SLAM is depicted in Figure 10.1.

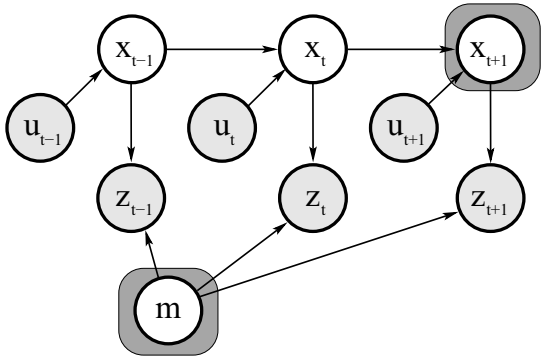


Figure 10.1 Graphical model of the online SLAM problem. The goal of online SLAM is to estimate a posterior over the current robot pose along with the map.

FULL SLAM PROBLEM

The second SLAM problem is called the *full SLAM problem*. In full SLAM, we seek to calculate a posterior over the entire path $x_{1:t}$ along with the map, instead of just the current pose x_t (see also Figure 10.2):

$$p(x_{1:t}, m \mid z_{1:t}, u_{1:t}) \tag{10.2}$$

This subtle difference between online and full SLAM has ramifications in the type of algorithms that can be brought to bear. In particular, the online SLAM problem is the result of integrating out past poses from the full SLAM problem:

$$p(x_t, m \mid z_{1:t}, u_{1:t}) \quad = \quad \int\int\cdots\int p(x_{1:t}, m \mid z_{1:t}, u_{1:t})\; dx_1\; dx_2 \ldots\; dx_{t-1} \tag{10.3}$$

In online SLAM, these integrations are typically performed one-at-a-time. They cause interesting changes of the dependency structures in SLAM that we will fully explore in the next chapter.

A second key characteristic of the SLAM problem has to do with the nature of the estimation problem. SLAM problems possess a continuous and a discrete component. The continuous estimation problem pertains to the location of the objects in the map and the robot's own pose variables. Objects may be landmarks in feature-based representation, or they might be object patches detected by range finders. The discrete nature has to do with correspondence: When an object is detected, a SLAM algorithm must reason about the relation of this object to previously detected objects. This reasoning is typically discrete: Either the object is the same as a previously detected one, or it is not.

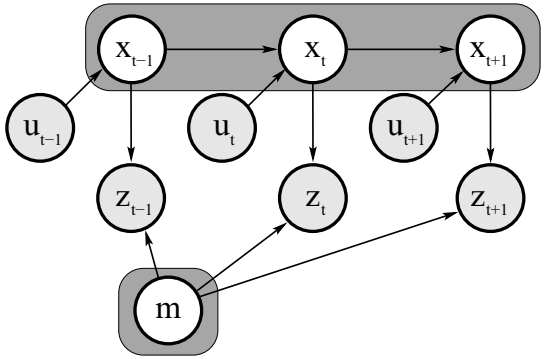


Figure 10.2 Graphical model of the full SLAM problem. Here, we compute a joint posterior over the whole path of the robot and the map.

We already encountered similar continuous-discrete estimation problems in previous chapters. For example, EKF localization in Chapter 7.4 estimates the robot pose, which is continuous. But to do so it also estimates the correspondences of measurements and landmarks in the map, which are discrete. In this and the subsequent chapters, we will discuss a number of different techniques to deal with the continuous and the discrete aspects of the SLAM problem.

At times, it will be useful to make the correspondence variables explicit, as we did in Chapter 7 on localization. The online SLAM posterior is then given by

$$p(x_t, m, c_t \mid z_{1:t}, u_{1:t}) \tag{10.4}$$

and the full SLAM posterior by

$$p(x_{1:t}, m, c_{1:t} \mid z_{1:t}, u_{1:t}) \tag{10.5}$$

The online posterior is obtained from the full posterior by integrating out past robot poses and summing over all past correspondences:

$$\begin{aligned} & p(x_t, m, c_t \mid z_{1:t}, u_{1:t}) \\ = & \int\int\cdots\int \sum_{c_1}\sum_{c_2}\cdots\sum_{c_{t-1}} p(x_{1:t}, m, c_{1:t} \mid z_{1:t}, u_{1:t})\, dx_1\, dx_2 \ldots\, dx_{t-1} \end{aligned} \tag{10.6}$$

In both versions of the SLAM problems—online and full—estimating the full posterior (10.4) or (10.5) is the gold standard of SLAM. The full posterior captures all there is to be known about the map and the pose or the path.

In practice, calculating a full posterior is usually infeasible. Problems arise from two sources: (1) the high dimensionality of the continuous parameter space, and (2) the large number of discrete correspondence variables. Many state-of-the-art SLAM algorithms construct maps with tens of thousands of features, or more. Even under known correspondence, the posterior over those maps alone involves probability distributions over spaces with 10^5 or more dimensions. This is in stark contrast to localization problems, in which posteriors were estimated over three-dimensional continuous spaces. Further, in most applications the correspondences are unknown. The number of possible assignments to the vector of all correspondence variables $c_{1:t}$ grows exponentially in the time t. Thus, practical SLAM algorithms that can cope with the correspondence problem must rely on approximations.

The SLAM problem will be discussed in a number of subsequent chapters. The remainder of this chapter develops an EKF algorithm for the online SLAM problem. Much of this material builds on Chapter 3.3, where the EKF was introduced, and Chapter 7.4, where we applied the EKF to the mobile robot localization problem. We will derive a progression of EKF algorithms that first apply EKFs to SLAM with known correspondences, and then progress to the more general case with unknown correspondences.

10.2 SLAM with Extended Kalman Filters

10.2.1 Setup and Assumptions

Historically the earliest—and perhaps the most influential—SLAM algorithm is based on the extended Kalman filter, or EKF. In a nutshell, the EKF SLAM algorithm applies the EKF to online SLAM using maximum likelihood data association. In doing so, EKF SLAM is subject to a number of approximations and limiting assumptions:

FEATURE-BASED MAPS

Maps, in EKF SLAM, are *feature-based*. They are composed of point landmarks. For computational reasons, the number of point landmarks is usually small (e.g., smaller than 1,000). Further, the EKF approach tends to work well the less ambiguous the landmarks are. For this reason, EKF SLAM requires significant engineering of feature detectors, sometimes using artificial beacons as features.

GAUSSIAN NOISE ASSUMPTION

As any EKF algorithm, EKF SLAM makes a *Gaussian noise assumption* for robot motion and perception. The amount of uncertainty in the posterior must be relatively small, since otherwise the linearization in EKFs tend to introduce intolerable errors.

POSITIVE INFORMATION

The EKF SLAM algorithm, just like the EKF localizer discussed in Chapter 7.4, can only process *positive* sightings of landmarks. It cannot process negative information that arises from the absence of landmarks in sensor measurements. This is a direct consequence of the Gaussian belief representation and was already discussed in Chapter 7.4.

10.2.2 SLAM with Known Correspondence

The SLAM algorithm for the case with known correspondence addresses the continuous portion of the SLAM problem only. Its development is in many ways parallel to the derivation of the EKF localization algorithm in Chapter 7.4, but with one key difference: In addition to estimating the robot pose x_t, the EKF SLAM algorithm also estimates the coordinates of all landmarks encountered along the way. This makes it necessary to include the landmark coordinates into the state vector.

COMBINED STATE VECTOR

For convenience, let us call the state vector comprising robot pose and the map the *combined state vector*, and denote this vector y_t. The combined vector is given by

$$
\begin{aligned}
y_t &= \begin{pmatrix} x_t \\ m \end{pmatrix} \\
&= \begin{pmatrix} x & y & \theta & m_{1,x} & m_{1,y} & s_1 & m_{2,x} & m_{2,y} & s_2 & \dots & m_{N,x} & m_{N,y} & s_N \end{pmatrix}^T
\end{aligned}
\tag{10.7}
$$

Here x, y, and θ denote the robot's coordinates at time t (not to be confused with the state variables x_t and y_t), $m_{i,x}, m_{i,y}$ are the coordinates of the i-th landmark, for $i = 1, \dots, N$, and s_i is its signature. The dimension of this state vector is $3N + 3$, where N denotes the number of landmarks in the map. Clearly, for any reasonable number of N, this vector is significantly larger than the pose vector that is being estimated in Chapter 7.4, which introduced the EKF localization algorithm. EKF SLAM calculates the online posterior $p(y_t \mid z_{1:t}, u_{1:t})$.

The *EKF SLAM algorithm* is depicted in Table 10.1—notice the similarity to the EKF localization algorithm in Table 7.2. Lines 2 through 5 apply the motion update, whereas lines 6 through 20 incorporate the measurement vector.

Lines 3 and 5 manipulate the mean and covariance of the belief in accordance to the motion model. This manipulation only affects those elements of the belief distribution concerned with the robot pose. It leaves all mean and covariance variables for the map unchanged, along with the pose-map covariances. Lines 7 through 20 iterate through all measurements. The test

Algorithm EKF_SLAM_known_correspondences$(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t, c_t)$**:**

$$F_x = \begin{pmatrix} 1 & 0 & 0 & 0 \cdots 0 \\ 0 & 1 & 0 & 0 \cdots 0 \\ 0 & 0 & 1 & \underbrace{0 \cdots 0}_{3N} \end{pmatrix}$$

$$\bar{\mu}_t = \mu_{t-1} + F_x^T \begin{pmatrix} -\frac{v_t}{\omega_t} \sin \mu_{t-1,\theta} + \frac{v_t}{\omega_t} \sin(\mu_{t-1,\theta} + \omega_t \Delta t) \\ \frac{v_t}{\omega_t} \cos \mu_{t-1,\theta} - \frac{v_t}{\omega_t} \cos(\mu_{t-1,\theta} + \omega_t \Delta t) \\ \omega_t \Delta t \end{pmatrix}$$

$$G_t = I + F_x^T \begin{pmatrix} 0 & 0 & -\frac{v_t}{\omega_t} \cos \mu_{t-1,\theta} + \frac{v_t}{\omega_t} \cos(\mu_{t-1,\theta} + \omega_t \Delta t) \\ 0 & 0 & -\frac{v_t}{\omega_t} \sin \mu_{t-1,\theta} + \frac{v_t}{\omega_t} \sin(\mu_{t-1,\theta} + \omega_t \Delta t) \\ 0 & 0 & 0 \end{pmatrix} F_x$$

$$\bar{\Sigma}_t = G_t \, \Sigma_{t-1} \, G_t^T + F_x^T \, R_t \, F_x$$

$$Q_t = \begin{pmatrix} \sigma_r^2 & 0 & 0 \\ 0 & \sigma_\phi^2 & 0 \\ 0 & 0 & \sigma_s^2 \end{pmatrix}$$

for all observed features $z_t^i = (r_t^i \;\; \phi_t^i \;\; s_t^i)^T$ *do*

$j = c_t^i$

if landmark j *never seen before*

$$\begin{pmatrix} \bar{\mu}_{j,x} \\ \bar{\mu}_{j,y} \\ \bar{\mu}_{j,s} \end{pmatrix} = \begin{pmatrix} \bar{\mu}_{t,x} \\ \bar{\mu}_{t,y} \\ s_t^i \end{pmatrix} + \begin{pmatrix} r_t^i \cos(\phi_t^i + \bar{\mu}_{t,\theta}) \\ r_t^i \sin(\phi_t^i + \bar{\mu}_{t,\theta}) \\ 0 \end{pmatrix}$$

endif

$$\delta = \begin{pmatrix} \delta_x \\ \delta_y \end{pmatrix} = \begin{pmatrix} \bar{\mu}_{j,x} - \bar{\mu}_{t,x} \\ \bar{\mu}_{j,y} - \bar{\mu}_{t,y} \end{pmatrix}$$

$$q = \delta^T \delta$$

$$\hat{z}_t^i = \begin{pmatrix} \sqrt{q} \\ \operatorname{atan2}(\delta_y, \delta_x) - \bar{\mu}_{t,\theta} \\ \bar{\mu}_{j,s} \end{pmatrix}$$

$$F_{x,j} = \begin{pmatrix} 1 & 0 & 0 & 0 \cdots 0 & 0 & 0 & 0 & 0 \cdots 0 \\ 0 & 1 & 0 & 0 \cdots 0 & 0 & 0 & 0 & 0 \cdots 0 \\ 0 & 0 & 1 & 0 \cdots 0 & 0 & 0 & 0 & 0 \cdots 0 \\ 0 & 0 & 0 & 0 \cdots 0 & 1 & 0 & 0 & 0 \cdots 0 \\ 0 & 0 & 0 & 0 \cdots 0 & 0 & 1 & 0 & 0 \cdots 0 \\ 0 & 0 & 0 & \underbrace{0 \cdots 0}_{3j-3} & 0 & 0 & 1 & \underbrace{0 \cdots 0}_{3N-3j} \end{pmatrix}$$

$$H_t^i = \frac{1}{q} \begin{pmatrix} -\sqrt{q}\delta_x & -\sqrt{q}\delta_y & 0 & +\sqrt{q}\delta_x & \sqrt{q}\delta_y & 0 \\ \delta_y & -\delta_x & -q & -\delta_y & +\delta_x & 0 \\ 0 & 0 & 0 & 0 & 0 & q \end{pmatrix} F_{x,j}$$

$$K_t^i = \bar{\Sigma}_t \, H_t^{iT} (H_t^i \, \bar{\Sigma}_t \, H_t^{iT} + Q_t)^{-1}$$

$$\bar{\mu}_t = \bar{\mu}_t + K_t^i (z_t^i - \hat{z}_t^i)$$

$$\bar{\Sigma}_t = (I - K_t^i \, H_t^i) \, \bar{\Sigma}_t$$

endfor

$$\mu_t = \bar{\mu}_t$$

$$\Sigma_t = \bar{\Sigma}_t$$

return μ_t, Σ_t

Table 10.1 The EKF algorithm for the SLAM problem (with known correspondences).

TESLA EXHIBIT 1011, Page 314

in line 9 returns true only for landmarks for which we have no initial location estimate. For those, line 10 initializes the location of such a landmark by the projected location obtained from the corresponding range and bearing measurement. As we shall discuss below, this step is important for the linearization in EKFs; it would not be needed in linear Kalman filters. For each measurement, an "expected" measurement is computed in line 14, and the corresponding *Kalman gain* is computed in line 17. Notice that the Kalman gain is a matrix of size 3 by $3N + 3$. This matrix is usually not sparse. Information is propagated through the entire state estimate. The filter update then occurs in lines 18 and 19, where the innovation is folded back into the robot's belief.

The fact that the Kalman gain is fully populated for all state variables—and not just the observed landmark and the robot pose—is important. In SLAM, observing a landmark does not just improve the position estimate of this very landmark, but that of other landmarks as well. This effect is mediated by the robot pose: Observing a landmark improves the robot pose estimate, and as a result it eliminates some of the uncertainty of landmarks previously seen by the same robot. The amazing effect here is that we do not have to model past poses explicitly—which would put us into the realm of the full SLAM problem and make the EKF a non-realtime algorithm. Instead, this dependence is captured in the Gaussian posterior, more specifically, in the off-diagonal covariance elements of the matrix Σ_t.

Figure 10.3 illustrates the EKF SLAM algorithm for an artificial example. The robot navigates from a start pose that serves as the origin of its coordinate system. As it moves, its own pose uncertainty increases, as indicated by uncertainty ellipses of growing diameter. It also senses nearby landmarks and maps them with an uncertainty that combines the fixed measurement uncertainty with the increasing pose uncertainty. As a result, the uncertainty in the landmark locations grows over time. In fact, it parallels that of the pose uncertainty at the time a landmark is observed. The interesting transition happens in Figure 10.3d: Here the robot observes the landmark it saw in the very beginning of mapping, and whose location is relatively well known. Through this observation, the robot's pose error is reduced, as indicated in Figure 10.3d—notice the very small error ellipse for the final robot pose! This observation also reduces the uncertainty for other landmarks in the map. This phenomenon arises from a correlation that is expressed in the covariance matrix of the Gaussian posterior. Since most of the uncertainty in earlier landmarks is caused by the robot pose, and since this very uncertainty persists over time, the location estimates of those landmarks are

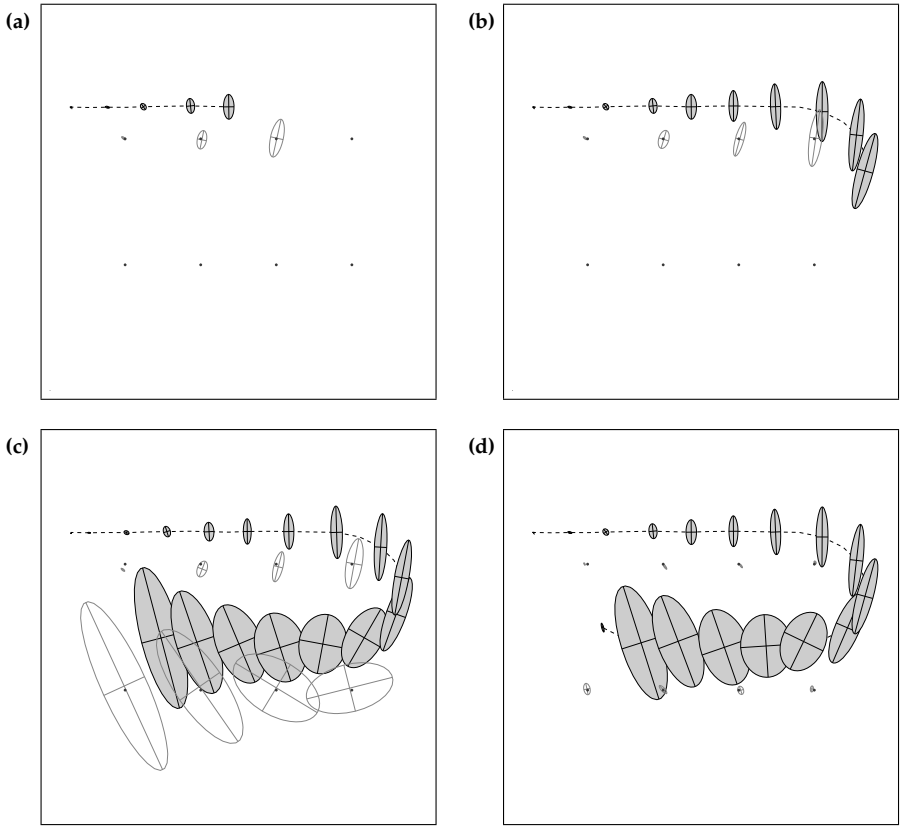


Figure 10.3 EKF applied to the online SLAM problem. The robot's path is a dotted line, and its estimates of its own position are shaded ellipses. Eight distinguishable landmarks of unknown location are shown as small dots, and their location estimates are shown as white ellipses. In (a)–(c) the robot's positional uncertainty is increasing, as is its uncertainty about the landmarks it encounters. In (d) the robot senses the first landmark again, and the uncertainty of *all* landmarks decreases, as does the uncertainty of its current pose. Image courtesy of Michael Montemerlo, Stanford University.

correlated. When gaining information on the robot's pose, this information spreads to previously observed landmarks. This effect is probably the most important characteristic of the SLAM posterior. Information that helps localize the robot is propagated through map, and as a result improves the

localization of other landmarks in the map.

10.2.3 Mathematical Derivation of EKF SLAM

The derivation of the EKF SLAM algorithm for the case with known correspondences largely parallels that of the EKF localizer in Chapter 7.4. The key difference is the augmented state vector, which now includes the locations of all landmarks in addition to the robot pose.

In SLAM, the initial pose is taken to be the origin of the coordinate system. This definition is somewhat arbitrary, in that it can be replaced by any coordinate. None of the landmark locations are known initially. The following initial mean and covariance express this belief:

$$\mu_0 = \begin{pmatrix} 0 & 0 & 0 & \dots & 0 \end{pmatrix}^T \tag{10.8}$$

$$\Sigma_0 = \begin{pmatrix} 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \infty & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \infty \end{pmatrix} \tag{10.9}$$

The covariance matrix is of size $(3N+3) \times (3N+3)$. It is composed of a small 3×3 matrix of zeros for the robot pose variables. All other covariance values are infinite.

As the robot moves, the state vector changes according to the standard noise-free velocity model (see Equations (5.13) and (7.4)). In SLAM, this motion model is extended to the augmented state vector:

$$y_t = y_{t-1} + \begin{pmatrix} -\frac{v_t}{\omega_t}\sin\theta + \frac{v_t}{\omega_t}\sin(\theta + \omega_t \Delta t) \\ \frac{v_t}{\omega_t}\cos\theta - \frac{v_t}{\omega_t}\cos(\theta + \omega_t \Delta t) \\ \omega_t \Delta t + \gamma_t \Delta t \\ 0 \\ \vdots \\ 0 \end{pmatrix} \tag{10.10}$$

The variables x, y, and θ denote the robot pose in y_{t-1}. Because the motion only affects the robot's pose and all landmarks remain where they are, only the first three elements in the update are non-zero. This enables us to write

the same equation more compactly:

$$y_t = y_{t-1} + F_x^T \begin{pmatrix} -\frac{v_t}{\omega_t}\sin\theta + \frac{v_t}{\omega_t}\sin(\theta+\omega_t\Delta t) \\ \frac{v_t}{\omega_t}\cos\theta - \frac{v_t}{\omega_t}\cos(\theta+\omega_t\Delta t) \\ \omega_t\Delta t + \gamma_t\Delta t \end{pmatrix} \tag{10.11}$$

Here F_x is a matrix that maps the 3-dimensional state vector into a vector of dimension $3N+3$.

$$F_x = \begin{pmatrix} 1 & 0 & 0 & 0 \cdots 0 \\ 0 & 1 & 0 & 0 \cdots 0 \\ 0 & 0 & 1 & \underbrace{0 \cdots 0}_{3N \text{ columns}} \end{pmatrix} \tag{10.12}$$

The full motion model with noise is then as follows

$$y_t = \underbrace{y_{t-1} + F_x^T \begin{pmatrix} -\frac{v_t}{\omega_t}\sin\theta + \frac{v_t}{\omega_t}\sin(\theta+\omega_t\Delta t) \\ \frac{v_t}{\omega_t}\cos\theta - \frac{v_t}{\omega_t}\cos(\theta+\omega_t\Delta t) \\ \omega\Delta t \end{pmatrix}}_{g(u_t, y_{t-1})} + \mathcal{N}(0, F_x^T\ R_t\ F_x) \tag{10.13}$$

where $F_x^T R_t F_x$ extends the covariance matrix to the dimension of the full state vector squared.

As usual in EKFs, the motion function g is approximated using a first degree Taylor expansion

$$g(u_t, y_{t-1}) \approx g(u_t, \mu_{t-1}) + G_t\ (y_{t-1} - \mu_{t-1}) \tag{10.14}$$

where the Jacobian $G_t = g'(u_t, \mu_{t-1})$ is the derivative of g with respect to y_{t-1} at u_t and μ_{t-1}, as in Equation (7.7).

Obviously, the additive form in (10.13) enables us to decompose this Jacobian into an identity matrix of dimension $(3N+3)\times(3N+3)$ (the derivative of y_{t-1}) plus a low-dimensional Jacobian g_t that characterizes the change of the robot pose:

$$G_t = I + F_x^T\ g_t\ F_x \tag{10.15}$$

with

$$g_t = \begin{pmatrix} 0 & 0 & -\frac{v_t}{\omega_t}\cos\mu_{t-1,\theta} + \frac{v_t}{\omega_t}\cos(\mu_{t-1,\theta}+\omega_t\Delta t) \\ 0 & 0 & -\frac{v_t}{\omega_t}\sin\mu_{t-1,\theta} + \frac{v_t}{\omega_t}\sin(\mu_{t-1,\theta}+\omega_t\Delta t) \\ 0 & 0 & 0 \end{pmatrix} \tag{10.16}$$

Plugging these approximations into the standard EKF algorithm gives us lines 2 through 5 of Table 10.1. Obviously, several of the matrices multiplied

in line 5 are sparse, which should be exploited when implementing this algorithm. The result of this update are the mean $\bar{\mu}_t$ and the covariance $\bar{\Sigma}_t$ of the estimate at time t after updating the filter with the control u_t, but before integrating the measurement z_t.

The derivation of the measurement update is similar to the one in Chapter 7.4. In particular, we are given the following measurement model

$$z_t^i = \underbrace{\begin{pmatrix} \sqrt{(m_{j,x}-x)^2+(m_{j,y}-y)^2} \\ \operatorname{atan2}(m_{j,y}-y, m_{j,x}-x)-\theta \\ m_{j,s} \end{pmatrix}}_{h(y_t,j)} + \mathcal{N}(0, \underbrace{\begin{pmatrix} \sigma_r & 0 & 0 \\ 0 & \sigma_\phi & 0 \\ 0 & 0 & \sigma_s \end{pmatrix}}_{Q_t}) \tag{10.17}$$

Here x, y, and θ denotes the pose of the robot, i is the index of an individual landmark observation in z_t, and $j = c_t^i$ is the index of the observed landmark at time t. The variable r denotes the range to a landmark, ϕ is the bearing to a landmark, and s the landmark signature; the terms σ_r, σ_ϕ, and σ_s are the corresponding measurement noise covariances.

This expression is approximated by the linear function

$$h(y_t, j) \approx h(\bar{\mu}_t, j) + H_t^i\,(y_t - \bar{\mu}_t) \tag{10.18}$$

Here H_t^i is the derivative of h with respect to the full state vector y_t. Since h depends only on two elements of that state vector, the robot pose x_t and the location of the j-th landmark m_j, the derivative factors into a low-dimensional Jacobian h_t^i and a matrix $F_{x,j}$, which maps h_t^i into a matrix of the dimension of the full state vector:

$$H_t^i = h_t^i\, F_{x,j} \tag{10.19}$$

Here h_t^i is the Jacobian of the function $h(y_t, j)$ at $\bar{\mu}_t$, calculated with respect to the state variables x_t and m_j:

$$h_t^i = \begin{pmatrix} \frac{\bar{\mu}_{t,x}-\bar{\mu}_{j,x}}{\sqrt{q_t}} & \frac{\bar{\mu}_{t,y}-\bar{\mu}_{j,y}}{\sqrt{q_t}} & 0 & \frac{\bar{\mu}_{j,x}-\bar{\mu}_{t,x}}{\sqrt{q_t}} & \frac{\bar{\mu}_{j,y}-\bar{\mu}_{t,y}}{\sqrt{q_t}} & 0 \\ \frac{\bar{\mu}_{j,y}-\bar{\mu}_{t,y}}{q_t} & \frac{\bar{\mu}_{t,x}-\bar{\mu}_{j,x}}{q_t} & -1 & \frac{\bar{\mu}_{t,y}-\bar{\mu}_{j,y}}{q_t} & \frac{\bar{\mu}_{j,x}-\bar{\mu}_{t,x}}{q_t} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{10.20}$$

The scalar $q_t = (\bar{\mu}_{j,x} - \bar{\mu}_{t,x})^2 + (\bar{\mu}_{j,y} - \bar{\mu}_{t,y})^2$, and as before, $j = c_t^i$ is the landmark that corresponds to the measurement z_t^i. The matrix $F_{x,j}$ is of dimension $6 \times (3N+3)$. It maps the low-dimensional matrix h_t^i into a matrix

of dimension $3 \times (3N+3)$:

$$F_{x,j} = \begin{pmatrix} 1 & 0 & 0 & 0\cdots0 & 0 & 0 & 0 & 0\cdots0 \\ 0 & 1 & 0 & 0\cdots0 & 0 & 0 & 0 & 0\cdots0 \\ 0 & 0 & 1 & 0\cdots0 & 0 & 0 & 0 & 0\cdots0 \\ 0 & 0 & 0 & 0\cdots0 & 1 & 0 & 0 & 0\cdots0 \\ 0 & 0 & 0 & 0\cdots0 & 0 & 1 & 0 & 0\cdots0 \\ 0 & 0 & 0 & \underbrace{0\cdots0}_{3j-3} & 0 & 0 & 1 & \underbrace{0\cdots0}_{3N-3j} \end{pmatrix} \tag{10.21}$$

These expressions make up for the gist of the Kalman gain calculation in lines 8 through 17 in our EKF SLAM algorithm in Table 10.1, with one important extension. When a landmark is observed for the first time, its initial pose estimate in Equation (10.8) leads to a poor linearization. This is because with the default initialization in (10.8), the point about which h is being linearized is $(\hat{\mu}_{j,x} \;\; \hat{\mu}_{j,y} \;\; \hat{\mu}_{j,s})^T = (0 \;\; 0 \;\; 0)^T$, which is a poor estimator of the actual landmark location. A better landmark estimator is given in line 10 of Table 10.1. Here we initialize the landmark estimate $(\hat{\mu}_{j,x} \;\; \hat{\mu}_{j,y} \;\; \hat{\mu}_{j,s})^T$ with the expected position. This expected position is derived from the expected robot pose and the measurement variables for this landmark

$$\begin{pmatrix} \bar{\mu}_{j,x} \\ \bar{\mu}_{j,y} \\ \bar{\mu}_{j,s} \end{pmatrix} = \begin{pmatrix} \bar{\mu}_{t,x} \\ \bar{\mu}_{t,y} \\ s_t^i \end{pmatrix} + \begin{pmatrix} r_t^i \;\cos(\phi_t^i + \bar{\mu}_{t,\theta}) \\ r_t^i \;\sin(\phi_t^i + \bar{\mu}_{t,\theta}) \\ 0 \end{pmatrix} \tag{10.22}$$

We note that this initialization is only possible because the measurement function h is bijective. Measurements are two-dimensional, as are landmark locations. In cases where a measurement is of lower dimensionality than the coordinates of a landmark, h is a true projection and it is impossible to calculate a meaningful expectation for $(\bar{\mu}_{j,x} \;\; \bar{\mu}_{j,y} \;\; \bar{\mu}_{j,s})^T$ from a single measurement only. This is, for example, the case in computer vision implementations of SLAM, since cameras often calculate the angle to a landmark but not the range. SLAM is then usually performed by integrating multiple sightings and applying triangulation to determine an appropriate initial location estimate. In the SLAM literature, such a problem is known as *bearing only SLAM* and will be further discussed in one of the exercises (page 334).

BEARING ONLY SLAM

Finally, we note that the EKF algorithm requires memory that is quadratic in N, the number of landmarks in the map. Its update time is also quadratic in N. The quadratic update complexity stems from the matrix multiplications that take place at various locations in the EKF.

Algorithm EKF_SLAM($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t, N_{t-1}$):

$N_t = N_{t-1}$

$$F_x = \begin{pmatrix} 1 & 0 & 0 & 0 \cdots 0 \\ 0 & 1 & 0 & 0 \cdots 0 \\ 0 & 0 & 1 & 0 \cdots 0 \end{pmatrix}$$

$$\bar{\mu}_t = \mu_{t-1} + F_x^T \begin{pmatrix} -\frac{v_t}{\omega_t} \sin \mu_{t-1,\theta} + \frac{v_t}{\omega_t} \sin(\mu_{t-1,\theta} + \omega_t \Delta t) \\ \frac{v_t}{\omega_t} \cos \mu_{t-1,\theta} - \frac{v_t}{\omega_t} \cos(\mu_{t-1,\theta} + \omega_t \Delta t) \\ \omega_t \Delta t \end{pmatrix}$$

$$G_t = I + F_x^T \begin{pmatrix} 0 & 0 & -\frac{v_t}{\omega_t} \cos \mu_{t-1,\theta} + \frac{v_t}{\omega_t} \cos(\mu_{t-1,\theta} + \omega_t \Delta t) \\ 0 & 0 & -\frac{v_t}{\omega_t} \sin \mu_{t-1,\theta} + \frac{v_t}{\omega_t} \sin(\mu_{t-1,\theta} + \omega_t \Delta t) \\ 0 & 0 & 0 \end{pmatrix} F_x$$

$$\bar{\Sigma}_t = G_t \, \Sigma_{t-1} \, G_t^T + F_x^T \, R_t \, F_x$$

$$Q_t = \begin{pmatrix} \sigma_r & 0 & 0 \\ 0 & \sigma_\phi & 0 \\ 0 & 0 & \sigma_s \end{pmatrix}$$

for all observed features $z_t^i = (r_t^i \;\; \phi_t^i \;\; s_t^i)^T$ *do*

$$\begin{pmatrix} \bar{\mu}_{N_t+1,x} \\ \bar{\mu}_{N_t+1,y} \\ \bar{\mu}_{N_t+1,s} \end{pmatrix} = \begin{pmatrix} \bar{\mu}_{t,x} \\ \bar{\mu}_{t,y} \\ s_t^i \end{pmatrix} + r_t^i \begin{pmatrix} \cos(\phi_t^i + \bar{\mu}_{t,\theta}) \\ \sin(\phi_t^i + \bar{\mu}_{t,\theta}) \\ 0 \end{pmatrix}$$

for $k = 1$ *to* $N_t{+}1$ *do*

$$\delta_k = \begin{pmatrix} \delta_{k,x} \\ \delta_{k,y} \end{pmatrix} = \begin{pmatrix} \bar{\mu}_{k,x} - \bar{\mu}_{t,x} \\ \bar{\mu}_{k,y} - \bar{\mu}_{t,y} \end{pmatrix}$$

$$q_k = \delta_k^T \delta_k$$

see next page for continuation

continued from the previous page

$$\hat{z}_t^k = \begin{pmatrix} \sqrt{q_k} \\ \text{atan2}(\delta_{k,y}, \delta_{k,x}) - \bar{\mu}_{t,\theta} \\ \bar{\mu}_{k,s} \end{pmatrix}$$

$$F_{x,k} = \begin{pmatrix} 1 & 0 & 0 & 0 \cdots 0 & 0 & 0 & 0 & 0 \cdots 0 \\ 0 & 1 & 0 & 0 \cdots 0 & 0 & 0 & 0 & 0 \cdots 0 \\ 0 & 0 & 1 & 0 \cdots 0 & 0 & 0 & 0 & 0 \cdots 0 \\ 0 & 0 & 0 & 0 \cdots 0 & 1 & 0 & 0 & 0 \cdots 0 \\ 0 & 0 & 0 & 0 \cdots 0 & 0 & 1 & 0 & 0 \cdots 0 \\ 0 & 0 & 0 & 0 \cdots 0 & 0 & 0 & 1 & 0 \cdots 0 \end{pmatrix}$$

$$H_t^k = \frac{1}{q_k} \begin{pmatrix} -\sqrt{q}_k \delta_{k,x} & -\sqrt{q}_k \delta_{k,y} & 0 & \sqrt{q}_k \delta_{k,x} & \sqrt{q}_k \delta_{k,y} & 0 \\ \delta_{k,y} & -\delta_{k,x} & -1 & -\delta_{k,y} & \delta_{k,x} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} F_{x,k}$$

$$\Psi_k = H_t^k \, \bar{\Sigma}_t \, [H_t^k]^T + Q_t$$

$$\pi_k = (z_t^i - \hat{z}_t^k)^T \, \Psi_k^{-1} \, (z_t^i - \hat{z}_t^k)$$

endfor

$\pi_{N_t+1} = \alpha$

$j(i) = \underset{k}{\text{argmin}} \; \pi_k$

$N_t = \max\{N_t, j(i)\}$

$K_t^i = \bar{\Sigma}_t \, [H_t^{j(i)}]^T \Psi_{j(i)}^{-1}$

$\bar{\mu}_t = \bar{\mu}_t + K_t^i \, (z_t^i - \hat{z}_t^{j(i)})$

$\bar{\Sigma}_t = (I - K_t^i \, H_t^{j(i)}) \, \bar{\Sigma}_t$

endfor

$\mu_t = \bar{\mu}_t$

$\Sigma_t = \bar{\Sigma}_t$

return μ_t, Σ_t

Table 10.2 The EKF SLAM algorithm with ML correspondences, shown here with outlier rejection.

10.3 EKF SLAM with Unknown Correspondences

10.3.1 The General EKF SLAM Algorithm

MAXIMUM LIKELIHOOD CORRESPONDENCE

The EKF SLAM algorithm with known correspondences is now extended into the general EKF SLAM algorithm, which uses an incremental *maximum likelihood* (ML) estimator to determine correspondences. Table 10.2 depicts the algorithm for unknown correspondences.

Since the correspondence is unknown, the input to the algorithm **EKF_SLAM** lacks a correspondence variable c_t. Instead, it includes the momentary size of the map, N_{t-1}. The motion update in lines 3 through 6 is equivalent to the one in **EKF_SLAM_known_correspondences** in Table 10.1. The measurement update loop, however, is different. Starting in line 8, it first creates the hypothesis of a new landmark with index $N_t + 1$; this index is one larger than the landmarks in the map at this point in time. The new landmark's location is initialized in line 9, by calculating its expected location given the estimate of the robot pose and the range and bearing in the measurement. Line 9 also assigns the observed signature value to this new landmark. Next, various update quantities are then computed in lines 10 through 18 for all $N_t + 1$ possible landmarks, including the "new" landmark. Line 19 sets the threshold for the creation of a new landmark: A new landmark is created if the Mahalanobis distance to all existing landmarks in the map exceeds the value α. The ML correspondence is then selected in line 20. If the measurement is associated with a previously unseen landmark, the landmark counter is incremented in line 21, and various vectors and matrices are enlarged accordingly—this somewhat tedious step is not made explicit in Table 10.2. The update of the EKF finally takes place in lines 23 and 24. The algorithm **EKF_SLAM** returns the new number of landmarks N_t along with the mean μ_t and the covariance Σ_t.

The derivation of this EKF SLAM follows directly from previous derivations. In particular, the initialization in line 9 is identical to the one in line 10 in **EKF_SLAM_known_correspondences**, Table 10.1. Lines 10 through 18 parallel lines 12 through 17 in **EKF_SLAM_known_correspondences**, with the added variable π_k needed for calculating the ML correspondence. The selection of the ML correspondence in line 20, and the definition of the Mahalanobis distance in line 17, is analogous to the ML correspondence discussed in Chapter 7.5; in particular, the algorithm **EKF_localization** in Table 7.3 on page 217 used an analogous equation to determine the most likely landmark (line 16). The measurement updates in lines 23 and 24 of Table 10.2 are also

analogous to those in the EKF algorithm with known correspondences, assuming that the participating vectors and matrices are of the appropriate dimension in case the map has just been extended.

Our example implementation of **EKF_SLAM** can be made more efficient by restricting the landmarks considered in lines 10 through 18 to those that are near the robot. Further, many of the values and matrices calculated in this inner loop can safely be cached away when looping through more than one feature measurement vector z_t^i. In practice, a good management of features in the map and a tight optimization of this loop can greatly reduce the running time.

10.3.2 Examples

Figure 10.4 shows the EKF SLAM algorithm—here with known correspondence—applied in simulation. The left panel of each of the three diagrams plots the posterior distributions, marginalized for the individual landmarks and the robot pose. The right side depicts the correlation matrix for the augmented state vector y_t; the correlation is the normalized covariance. As is easily seen from the result in Figure 10.4c, over time all x- and y-coordinate estimates become fully correlated. This means the map becomes known in relative terms, up to an uncertain global location that cannot be reconciled. This highlights an important characteristic of the SLAM problem: The absolute coordinates of a map relative to the coordinate system defined by the initial robot pose can only be determined in approximation, whereas the relative coordinates can be determined asymptotically with certainty.

In practice, EKF SLAM has been applied successfully to a large range of navigation problems, involving airborne, underwater, indoor, and various other vehicles. Figure 10.5 shows an example result obtained using the underwater robot Oberon, developed at the University of Sydney, Australia, and shown in Figure 10.6. This vehicle is equipped with a pencil sonar, a sonar that can scan at very high resolutions and detect obstacles up to 50 meters away. To facilitate the mapping problem, researchers have deposited long, small vertical objects in the water, which can be extracted from the sonar scans with relative ease. In this specific experiment, there is a row of such objects, spaced approximately 10 meters apart. In addition, a more distant cliff offers additional point features that can be detected using the pencil sonar.

In the experiment shown in Figure 10.5, the robot moves by these land-

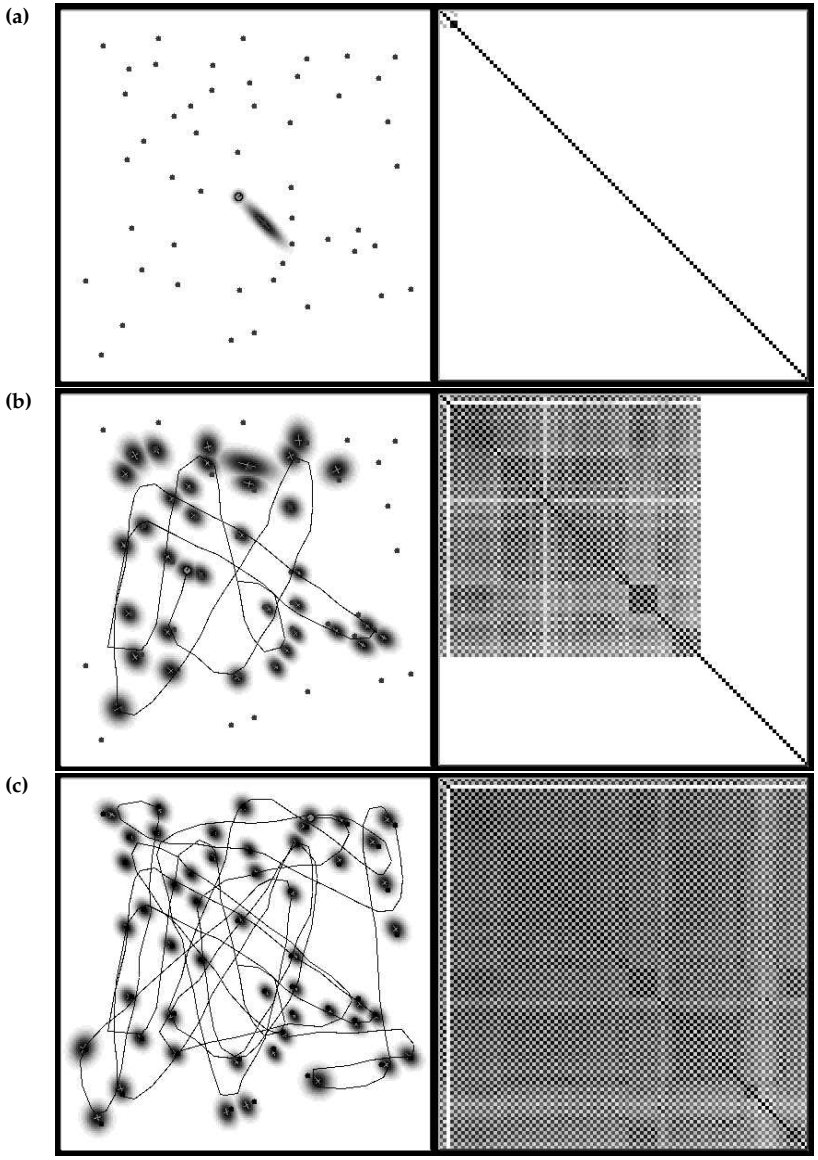


Figure 10.4 EKF SLAM with known data association in a simulated environment. The map is shown on the left, with the gray-level corresponding to the uncertainty of each landmark. The matrix on the right is the correlation matrix, which is the normalized covariance matrix of the posterior estimate. After some time, all x- and all y-coordinate estimates become fully correlated.

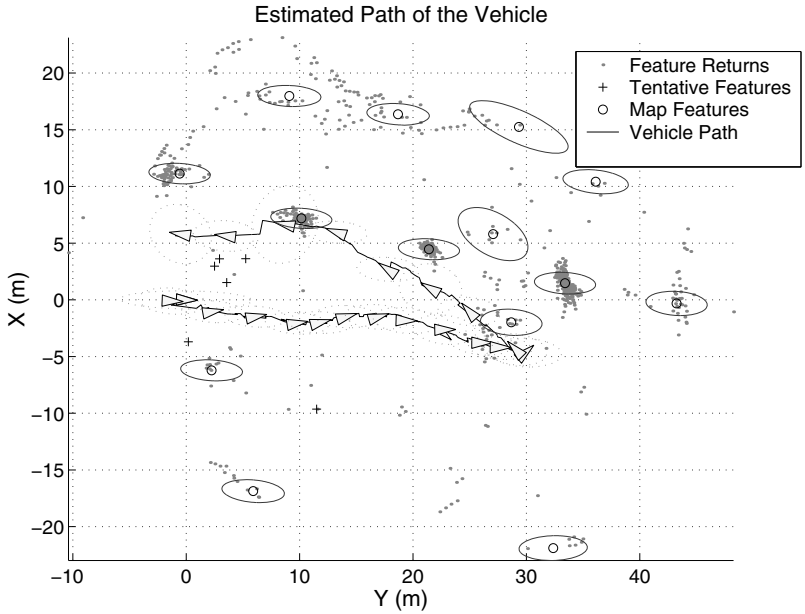
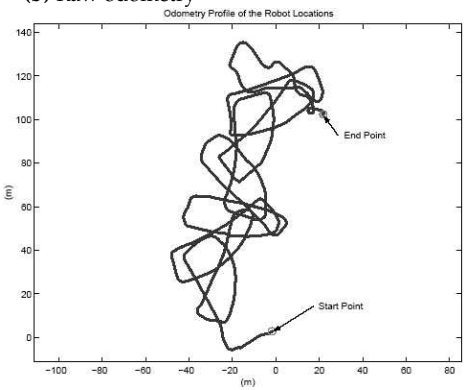


Figure 10.5 Example of Kalman filter estimation of the map and the vehicle pose. Image courtesy of Stefan Williams and Hugh Durrant-Whyte, Australian Centre for Field Robotics.

Figure 10.6 Underwater vehicle Oberon, developed at the University of Sydney. Image courtesy of Stefan Williams and Hugh Durrant-Whyte, Australian Centre for Field Robotics.

(a) RWI B21 Mobile robot and testing environment

(b) Raw odometry



(c) Result of EKF SLAM with ground truth

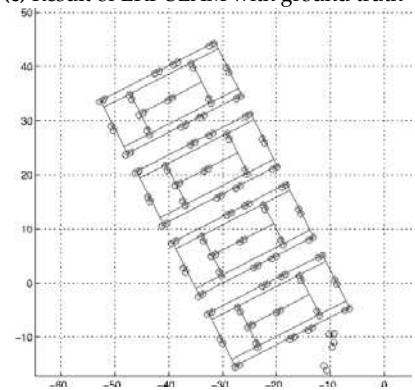


Figure 10.7 (a) The MIT B21 mobile robot in a calibrated testing facility. (b) Raw odometry of the robot, as it is manually driven through the environment. (c) The result of EKF SLAM is a highly accurate map. The image shows the estimated map overlayed on a manually constructed map. All images and results are courtesy of John Leonard and Matthew Walter, MIT.

marks, then turns around and moves back. While doing so, it measures and integrates landmarks into its map using the EKF SLAM algorithm described in this chapter.

The map shown in Figure 10.5 shows the robot's path, marked by the triangles connected by a line. Around each triangle one can see an ellipse, which corresponds to the covariance matrix of the Kalman filter estimate projected into the robot's x-y position. The ellipse shows the variance; the larger it is, the less certain the robot is about its current pose. Various small dots in

Figure 10.5 show landmark sightings, obtained by searching the sonar scan for small and highly reflective objects. The majority of these sightings are rejected, using a mechanism described in the next section. However, some are believed to correspond to a landmark and are added to the map. At the end of the run, the robot has classified 14 such objects as landmarks, each of which is plotted with the projected uncertainty ellipse in Figure 10.5. These landmarks include the artificial landmarks put out by the researchers, but they also include various other terrain features in the vicinity of the robot. The residual pose uncertainty is small.

Figure 10.7 shows the result of another EKF SLAM implementation. Panel (a) shows MIT's RWI B21 mobile robot, situated in a testing environment. The testing environment is a tennis court; obstacles are hurdles whose position was measured manually with centimeter accuracy for evaluation purposes. Panel (b) of Figure 10.7 shows the raw odometry path. The result of EKF SLAM is shown in Panel (c), overlayed with the manually constructed map. The reader should verify that this is indeed an accurate map.

10.3.3 Feature Selection and Map Management

Making EKF SLAM robust in practice often requires additional techniques for *map management*. Many of them pertain to the fact that the Gaussian noise assumption is unrealistic, and many spurious measurements occur in the far tail end of the noise distribution. Such spurious measurements can cause the creation of fake landmarks in the map which, in turn, negatively affect the localization of the robot.

OUTLIERS

Many state-of-the-art techniques possess mechanisms to deal with *outliers* in the measurement space. Such outliers are defined as spurious landmark sightings outside the uncertainty range of any landmark in the map. The most simple technique to reject such outliers is to maintain a *provisional landmark list*. Instead of augmenting the map by a new landmark once a measurement indicates the existence of a new landmark, such a new landmark is first added to a provisional list of landmarks. This list is just like the map, but landmarks on this list are *not* used to adjust the robot pose (the corresponding gradients in the measurement equations are set to zero). Once a landmark has consistently been observed and its uncertainty ellipse has shrunk, it is transitioned into the regular map.

PROVISIONAL LANDMARK LIST

In practical implementations, this mechanism tends to reduce the number of landmarks in the map by a significant factor, while still retaining all physical landmarks with high probability. A further step, also commonly found

LANDMARK EXISTENCE PROBABILITY

in state-of-the-art implementations, is to maintain a *landmark existence probability*. Such a posterior probability may be implemented as log odds ratio and be denoted o_j for the j-th landmark in the map. Whenever the j-th landmark m_j is observed, o_j is incremented by a fixed value. Not observing m_j when it would be in the perceptual range of the robot's sensors leads to a decrement of o_j. Since it can never be known with certainty whether a landmark is within a robot's perceptual range, the decrement may factor in the probability of such an event. Landmarks are removed from the map when the value o_j drops below a threshold. Such techniques lead to much leaner maps in the face of non-Gaussian measurement noise.

NUMERICAL INSTABILITY OF EKF SLAM

When initializing the estimate for a new landmark starting with a covariance with very large elements—as suggested in Equation (10.9)—may induce *numerical instabilities*. This is because the very first covariance update step will change this value by several orders of magnitude, too many perhaps for generating a matrix that is still positive definite. A better strategy involves an explicit initialization step for any feature that has not been observed before. In particular, such a step would initialize the covariance Σ_t directly with the actual landmark uncertainty, instead of executing line 24 in Table 10.2 (same with the mean in line 23).

As noted previously, the maximum likelihood approach to data association has a clear limitation, which arises from the fact that the maximum likelihood approach deviates from the idea of full posterior estimation in probabilistic robotics. Instead of maintaining a joint posterior over augmented states and data associations, it reduces the data association problem to a deterministic determination, which is treated as if the maximum likelihood association was always correct. This limitation makes EKF brittle with regards to landmark confusion, which in turn can lead to wrong results. In practice, researchers often remedy the problem by choosing one of the following two methods, both of which reduce the chances of confusing landmarks:

- **Spatial arrangement.** The further apart landmarks are, the smaller the chance to accidentally confuse them. It is therefore common practice to choose landmarks that are sufficiently far away from each other so that the probability of confusing one with another becomes small. This introduces an interesting trade-off: a large number of landmarks increases the danger of confusing them. Too few landmarks makes it more difficult to localize the robot, which in turn also increases the chances of confusing landmarks. Little is currently known about the optimal density of landmarks, and researchers often use intuition when selecting specific land-

marks.

- **Signatures.** When selecting appropriate landmarks, it is essential to maximize the perceptual distinctiveness of landmarks. For example, doors might possess different colors, or corridors might have different widths. The resulting signatures are essential for successful SLAM.

With these additions, the EKF SLAM algorithm has indeed been applied successfully to a wide range of practical mapping problems, involving robotic vehicles in the air, on the ground, and underwater.

A key limitation of EKF SLAM lies in the necessity to select appropriate landmarks. By reducing the sensor stream to the presence and absence of landmarks, a lot of sensor data is usually discarded. This results in an information loss relative to a SLAM algorithm that can exploit sensors without extensive pre-filtering. Further, the quadratic update time of the EKF limits this algorithm to relatively scarce maps with less than 1,000 features. In practice, one often seeks maps with 10^6 features or more, in which case the EKF ceases to be applicable.

The relatively low dimensionality of the map tends to create a harder data association problem. This is easily verified: When you open your eyes and look at the full room you are in, you probably have no difficulty in recognizing where you are! However, if you are only told the location of a small number of landmarks—e.g., the location of all light sources—the decision is much harder. As a result, data association in EKF SLAM is more difficult than in some of the SLAM algorithms discussed in subsequent chapters, and capable of handling orders of magnitude more features. This culminates into the *fundamental dilemma of EKF SLAM*: While incremental maximum likelihood data association might work well with dense maps with hundreds of millions of features, it tends to be brittle with scarce maps. However, EKFs require sparse maps because of the quadratic update complexity. In subsequent chapters, we will discuss SLAM algorithms that are more efficient and can handle much larger maps. We will also discuss more robust data association techniques. For its many limitations, the value of the EKF SLAM algorithm presented in this chapter is mostly historical.

FUNDAMENTAL DILEMMA EKF SLAM

10.4 Summary

This chapter described the general SLAM problem and introduced the EKF approach.

- The SLAM problem is defined as a concurrent localization and mapping problem, in which a robot seeks to acquire a map of the environment while simultaneously seeking to localize itself relative to this map.

- The SLAM problem comes in two versions: online and global. Both problems involve the estimation of the map. The online problem seeks to estimate the momentary robot pose, whereas the global problem seeks to determine all poses. Both problems are of equal importance in practice, and have found equal coverage in the literature.

- The EKF SLAM algorithm is arguably the earliest SLAM algorithm. It applies the extended Kalman filter to the online SLAM problem. With known correspondences, the resulting algorithm is incremental. Updates require time quadratic in the number of landmarks in the map.

- When correspondences are unknown, the EKF SLAM algorithm applies an incremental maximum likelihood estimator to the correspondence problem. The resulting algorithm works well when landmarks are sufficiently distinct.

- Additional techniques were discussed for managing maps. Two common strategies for identifying outliers include a provisional list for landmarks that are not yet observed sufficiently often, and a landmark evidence counter that calculates the posterior evidence of the existence of a landmark.

- EKF SLAM has been applied with considerable success in a number of robotic mapping problems. Its main drawback is the need for sufficiently distinct landmarks, and the computational complexity required for updating the filter.

In practice, EKF SLAM has been applied with some success. When landmarks are sufficiently distinct, the approach approximates the posterior well. The advantage of calculating a full posterior are manifold: It captures all residual uncertainty and enables the robot to reason about its control taking its true uncertainty into account. However, the EKF SLAM algorithm suffers from its enormous update complexity, and the limitation to sparse maps. This, in turn, makes the data association problem a difficult one, and EKF SLAM tends to work poorly in situations where landmarks are highly ambiguous. Further brittleness is due to the fact that the EKF SLAM algorithm relies on an incremental maximum likelihood data association tech-

nique. This technique makes it impossible to revise past data associations, and can induce failure when the ML data association is incorrect.

The EKF SLAM algorithm applies to the online SLAM problem; it is inapplicable to the full SLAM problem. In the full SLAM problem, the addition of a new pose to the state vector at each time step would make both the state vector and the covariance grow without bounds. Updating the covariance would therefore require an ever-increasing amount of time, and the approach would quickly run out of computational time no matter how fast the processor.

10.5 Bibliographical Remarks

The problem of SLAM predates the invention of modern robots by many centuries. The problem of modeling a physical structure from a moving sensor platform is at the core of a number of fields, such as *geosciences*, *photogrammetry*, and *computer vision*. Many of the mathematical techniques that form the core of the SLAM work today were first developed for calculating planetary orbits. For example, the least squares method can be traced back to Johann Carl Friedrich Gauss (1809). SLAM is essentially a geographic surveying problem. Teleported to a robot it creates challenges that human surveyors rarely face, such as the correspondence problem and the problem of finding appropriate features.

In robotics, the EKF to the SLAM problem was introduced through a series of seminal papers by Cheeseman and Smith (1986); Smith and Cheeseman (1986); Smith et al. (1990). These papers were the first to describe the EKF approach discussed in this chapter. Just like us in this book, Smith et al. discussed the EKF in the context of feature-based mapping with point landmarks, and known data association. The first implementations of EKF SLAM were due to Moutarlier and Chatila (1989a,b) and Leonard and Durrant-Whyte (1991), some using artificial beacons as landmarks. The EKF became fashionable at a time when many authors investigated alternative techniques for maintaining accurate pose estimates during mapping (Cox 1991). Early work by Dickmanns and Graefe (1988) on estimation road curvature in autonomous cars is highly related; see (Dickmanns 2002) for a survey.

SLAM is a highly active field of research, as a recent workshop indicates (Leonard et al. 2002b). An extensive literature for the field of SLAM—or CML for *concurrent mapping and localization* as Leonard and Durrant-Whyte (1991) call it—can be found in Thrun (2002). The importance of maintaining correlations in the map was pointed out by Csorba (1997), who in his Ph.D. thesis, who also established some basic convergence results. Since then, a number of authors have extended the basic paradigm in many different ways. The feature management techniques described in this chapter are due to Dissanayake et al. (2001, 2002); see also Bailey (2002). Williams et al. (2001) developed the idea of provisional feature lists in SLAM, to reduce the effect of feature detection errors. Feature initialization is discussed in Leonard et al. (2002a), who explicitly maintains an estimate of previous poses to accommodate sensors that provide incomplete data on feature coordinates. A representation that avoids singularities by explicitly factoring "perturbations" out of the posterior was devised by Castellanos et al. (1999), who reports improved numerical stability over the vanilla EKF. Jensfelt et al. (2002) found significant improvements in indoor SLAM when utilizing basic geometric constraints, such as the fact that most walls are parallel or orthogonal. Early work on SLAM with sonars goes back to Rencken

(1993); a state-of-the-art system for SLAM with sonar sensors can be found in Tardós et al. (2002). Castellanos et al. (2004) provided a critical consistency analysis for the EKF. An empirical comparison of multiple algorithms can be found in Vaganay et al. (2004). A few open questions are discussed by Salichs and Moreno (2000). Research on the important data association problem will be reviewed in a later chapter (see page 481).

As noted, a key limitation of the EKF solution to the SLAM problem lies in the quadratic nature of the covariance matrix. This "flaw" has not remained unnoticed. In the past few years, a number of researchers have proposed EKF SLAM algorithms that gain remarkable scalability through decomposing the map into submaps, for which covariances are maintained separately. Some of the original work in this field is by Leonard and Feder (1999), Guivant and Nebot (2001), and Williams (2001). Leonard and Feder's (1999) *decoupled stochastic mapping* algorithm decomposes the map into collections of smaller, more manageable submaps. This approach is computationally efficient, but does not provide a mechanism to propagate information through the network of local maps (Leonard and Feder 2001). Guivant and Nebot (2001, 2002), in contrast, provided an approximate factorization of the covariance matrix which reduced the actual complexity of EKF updating by a significant factor. Williams (2001) and Williams et al. (2002) proposed the *constrained local submap filter (CLSF)*, which relies on creating independent local submaps of the features in the immediate vicinity of the vehicle. Williams et al. (2002) provides results for underwater mapping (see Figure 10.5 for some of his early work). The *sequential map joining techniques* described in Tardós et al. (2002) is a related decomposition. Bailey (2002) devises a similar technique for representing SLAM maps hierarchically. Folkesson and Christensen (2003) describes a technique by which frequent updates are limited to a small region near the robot, whereas the remainder of the map is updated at much lower frequencies. All these techniques achieve the same rate of convergence as the full EKF solution, but incur an $O(n^2)$ computational burden. However, they scale much better to large problems with tens of thousands of features.

A number of researchers have developed hybrid SLAM techniques, which combine EKF-style SLAM techniques with volumetric techniques, such as occupancy grid maps. The *hybrid metric map (HYMM)* by Guivant et al. (2004) and Nieto et al. (2004) decomposes maps into triangular regions (LTRs) using volumetric maps such as occupancy grid maps as a basic representation for those regions. These local maps are combined using EKFs. Burgard et al. (1999b) also decomposes maps into local occupancy grid maps, but uses the *expectation maximization* (EM) algorithm (see Dempster et al. (1977)) for combining local maps into a joint global map. The work by Betgé-Brezetz et al. (1995, 1996) integrated two types of representations into a SLAM framework: bitmaps for representing outdoor terrain, and object representations for sparse outdoor objects.

Extensions of SLAM to dynamic environments can be found in Wang et al. (2003), Hähnel et al. (2003c), and Wolf and Sukhatme (2004). Wang et al. (2003) developed an algorithm called *SLAM with DATMO*, short for SLAM with the detection and tracking of moving objects. Their approach is based on the EKF, but it allows for the possible motion of features. Hähnel et al. (2003c) studied the problem of performing SLAM in environments with many moving objects. They successfully employed the EM algorithm for filtering out measurements that likely correspond to moving objects. By doing so, they were able to acquire maps in environments where conventional SLAM techniques failed. The approach in Wolf and Sukhatme (2004) maintains two coupled occupancy grids of the environment, one for the static map, and one for moving objects. SLAM-style localization is achieved by a regular landmark-based SLAM algorithm.

SLAM systems have been brought to bear in a number of deployed systems. Rikoski et al. (2004) applied SLAM to sonar odometry of submarine, providing a new approach for "auditory odometry." SLAM in abandoned mines is described in Nüchter et al. (2004), who extended the

paradigm to full 6-D pose estimation. Extensions to the multi-robot SLAM problem have been proposed by a number of researchers. Some of the earlier work is by Nettleton et al. (2000), who developed a technique by which vehicles maintain local EKF maps, but fuse them using the information representation of the posterior. An alternative technique is due to Rekleitis et al. (2001a), who use a team of stationary and moving robots to reduce the localization error when performing SLAM. Fenwick et al. (2002) provides a comprehensive theoretical investigation of multi vehicle map fusion, specifically for landmark based SLAM. Techniques for fusing scans were developed in Konolige et al. (1999); Thrun et al. (2000b); Thrun (2001).

A number of researchers have developed SLAM systems for specific sensor types. An important sensor is a camera; however, cameras only provide bearing to features. This problem is well-studied in the computer vision literature as *structure from motion* (SFM) (Tomasi and Kanade 1992; Soatto and Brockett 1998; Dellaert et al. 2003), and in the field of photogrammetry (Konecny 2002). Within SLAM, the seminal work on bearing only SLAM is due to Deans and Hebert (2000, 2002). Their approach recursively estimates features of the environment that are invariant to the robot pose, so as to decouple the pose error from the map error. A great number of researchers has applied SLAM using cameras as the primary sensor (Neira et al. 1997; Cid et al. 2002; Davison 2003). Davison (1998) provides active vision techniques in the context of SLAM. Dudek and Jegessur's (2000) work relies on place recognition based on appearance, whereas Hayet et al. (2002) and Bouguet and Perona (1995) use visual landmarks. Diebel et al. (2004) developed a filter for SLAM with an active stereo sensor that accounts for the nonlinear noise distribution of a stereo range finder. Sensor fusion techniques for SLAM were developed by Devy and Bulata (1996). Castellanos et al. (2001) found empirically that fusing laser and camera outperformed each sensor modality in isolation.

STRUCTURE FROM MOTION

SLAM has also been extended to the problem of building dense 3-D models. Early systems for acquiring 3-D models with indoor mobile robots can be found in Reed and Allen (1997); Iocchi et al. (2000); Thrun et al. (2004b). Devy and Parra (1998) acquire 3-D models using parametric curves. Zhao and Shibasaki (2001), Teller et al. (2001), and Frueh and Zakhor (2003) have developed impressive systems for building large textured 3-D maps of urban environments. Neither of these systems addresses the full SLAM problem due to the availability of outdoor GPS, but they are highly related to the mathematical basis of SLAM. These techniques blend smoothly with a rich body of work on aerial reconstruction of urban environments (Jung and Lacroix 2003; Thrun et al. 2003).

The following chapters discuss alternatives to the plain EKFs. The techniques described there share many of the intuitions with the extensions discussed here, as the boundary between different types of filters has become almost impossible to draw. The literature review will be continued after the next chapter, when discussing a SLAM algorithm using information-theoretic representations.

10.6 Exercises

1. What is the computational complexity of the motion update in EKF SLAM? Use the $O(\ )$ notation. Compare this with the worst case complexity for EKFs over a feature vector of the same size.

BEARING ONLY SLAM

2. *Bearing only SLAM* refers to the SLAM problem when the sensors can only measure the bearing of a landmark but not its range. As noted, bearing

only SLAM is closely related to Structure from Motion (SFM) in Computer Vision. One problem in bearing only SLAM with EKFs concerns the initialization of landmark location estimates, even if the correspondences are known. Discuss why, and devise a technique for initializing the landmark location estimates (means and covariances) that can be applied in bearing only SLAM.

3. On page 329, we remarked that the EKF algorithm in Table 10.2 can become numerically instable. Devise a method for setting μ_t and Σ_t directly when a new feature is observed for the first time. Such a technique would not require initializing the covariance with very large values. Show that the result is mathematically equivalent to lines 23 and 24 when the covariance is initialized as in Equation (10.9).

4. The text suggests using a binary Bayes filter to compute the probability that a landmark represented in the posterior actually exists in the physical world.

   (a) In the first part of this exercise, you are asked to design such a binary Bayes filter.

   (b) Now extend your filter to a situation in which landmarks sporadically disappear with a probability p^*.

   (c) Picture a situation where for a well-established landmark, no information is received for a long time with regards to its existence (no positive and no negative information). To what value will your filter converge? Prove your answer.

5. The EKF SLAM algorithm, as presented here, is unable to cope with the data association problem in a sound statistical way. Lay out an algorithm (and a statistical framework) for posterior estimation with unknown data association that represents the posterior by mixtures of Gaussians, and characterize its advantages and disadvantages. How does the complexity of the posterior grow over time?

6. Based on the previous problem, develop an approximate method for posterior estimation with unknown data association where the time needed for each incremental update step does not grow over time (assuming a fixed number of landmarks).

7. Develop a Kalman filter algorithm that uses local occupancy grid maps as its basic components, instead of landmarks. Among other things, prob-

lems that have to be solved are how to relate local grids to each other, and how to deal with the ever-growing number of local grids.

11 *The GraphSLAM Algorithm*

11.1 Introduction

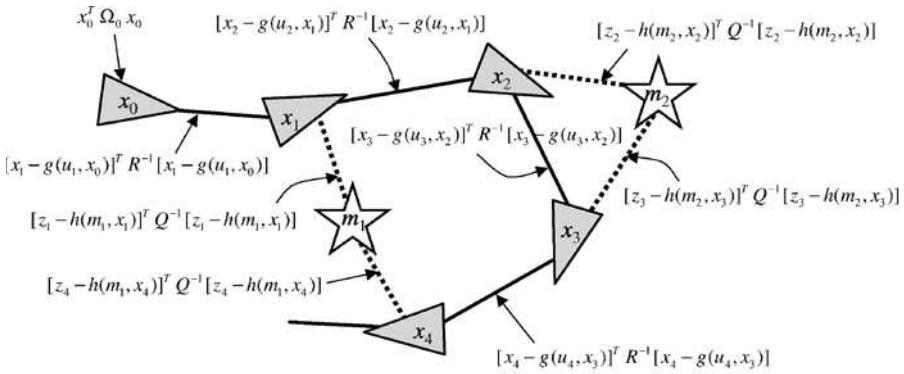
The EKF SLAM algorithm described in the previous chapter is subject to a number of limitations. One of them is its quadratic update complexity; another is the linearization technique in EKFs, which is only performed once in the EKF for each nonlinear term. In this chapter, we introduce an alternative SLAM algorithm, called *GraphSLAM*. In contrast to the EKF, GraphSLAM solves the *full SLAM problem*. It calculates a solution for the offline problem defined over all poses and all features in the map. As we will show in this chapter, the posterior of the *full SLAM problem* naturally forms a *sparse graph*. This graph leads to a sum of nonlinear quadratic constraints. Optimizing these constraints yields a maximum likelihood map and a corresponding set of robot poses. Historically, this idea can be found in a large number of SLAM publications. The name "GraphSLAM" has been chosen because it captures the essence of this approach.

SPARSE GRAPH

Figure 11.1 illustrates the GraphSLAM algorithm. Shown there is the graph that GraphSLAM extracts from five poses labeled $x_0, \ldots, x_4$, and two map features m_1, m_2. Arcs in this graph come in two types: motion arcs and measurement arcs. *Motion arcs* link any two consecutive robot poses, and *measurement arcs* link poses to features that were measured there. Each edge in the graph corresponds to a nonlinear constraint. As we shall see later, these constraints represent the negative log likelihood of the measurement and the motion models, hence are best thought of as *information constraints*. Adding such a constraint to the graph shall prove to be trivial for GraphSLAM; it involves no significant computation. The sum of all constraints results in a nonlinear *least squares problem*, as stated in Figure 11.1.

LEAST SQUARES

To compute a map posterior, GraphSLAM linearizes the set of constraints.



Sum of all constraints:

$$J_{\text{GraphSLAM}} = x_0^T \Omega_0 x_0 + \sum_t [x_t - g(u_t, x_{t-1})]^T R^{-1} [x_t - g(u_t, x_{t-1})] + \sum_t [z_t - h(m_{c_t}, x_t)]^T Q^{-1} [z_t - h(m_{c_t}, x_t)]$$

Figure 11.1 GraphSLAM illustration, with 4 poses and two map features. Nodes in the graphs are robot poses and feature locations. The graph is populated by two types of edges: Solid edges link consecutive robot poses, and dashed edges link poses with features sensed while the robot assumes that pose. Each link in GraphSLAM is a non-linear quadratic constraint. Motion constraints integrate the motion model; measurement constraints the measurement model. The target function of GraphSLAM is the sum of these constraints. Minimizing it yields the most likely map and the most likely robot path.

The result of linearization is an information matrix and an information vector of essentially the same form as already encountered in Chapter 3, when we discussed the information filter. However, the information matrix inherits the sparseness from the graph constructed by GraphSLAM. This sparseness enables GraphSLAM to apply the variable elimination algorithm, thereby transforming the graph into a much smaller one only defined over robot poses. The path posterior map is then calculated using standard inference techniques. GraphSLAM also computes a map and certain marginal posteriors over the map; the full map posterior is of course quadratic in the size of the map and hence is usually not recovered.

In may ways, EKF SLAM and GraphSLAM are extreme ends of the spectrum of SLAM algorithms. A primary difference between EKF SLAM and GraphSLAM pertains to the representation of information. While EKF

SLAM represents information through a covariance matrix and a mean vector, GraphSLAM represents the information as a graph of soft constraints. Updating the covariance in an EKF is computationally expensive; whereas growing the graph is cheap!

Such savings come at a price. GraphSLAM requires additional inference when recovering the map and the path, whereas EKF maintains its best estimate of the map and the robot pose at all times. The build-up of the graph is followed by a separate computational phase in which this information is transformed into an estimate of the state. No such phase is required for EKF SLAM.

PROACTIVE SLAM

LAZY SLAM

Consequently, one may think of EKF as a *proactive SLAM* algorithm, in the sense that it resolves any new piece of information immediately into an improved estimate of the state of the world. GraphSLAM, in contrast, is more like a *lazy SLAM* technique, which simply accumulates information into its graph without resolving it. This difference is significant. GraphSLAM can acquire maps that are many orders of magnitude larger than EKFs can handle.

There are further differences between EKF SLAM and GraphSLAM. As a solution to the full SLAM problem, GraphSLAM calculates posteriors over robot paths, hence is not an incremental algorithm. This approach is different from EKF SLAM, which, as a filter, only maintains a posterior over the momentary pose of the robot. EKF SLAM enables a robot to update its map forever, whereas GraphSLAM is best suited for problems where one seeks a map from a data set of fixed size. EKF SLAM can maintain a map over the entire lifetime of a robot without having to worry about the total number of time steps elapsed since the beginning of data acquisition.

Because GraphSLAM has access to the full data when building the map, it can apply improved linearization and data association techniques. In EKF SLAM, the linearization and the correspondence for a measurement at time t are calculated based on the data up to time t. In GraphSLAM *all* data can be used to linearize and to calculate correspondences. Put differently, GraphSLAM can revise past data association, and it can linearize more than once. In fact, GraphSLAM iterates the three crucial steps in mapping: the construction of the map, the calculation of correspondence variables, and the linearization of the measurement and motion models—so as to obtain the best estimate of all of those quantities. As a result of all this, GraphSLAM tends to produce maps that are superior in accuracy to maps generated by EKFs.

However, GraphSLAM is not without limitations when compared to the

EKF approach. One was already discussed: The size of the graph grows linearly over time, whereas the EKF shows no such time dependence in the amount of memory allocated to its estimate. Another pertains to data association. Whereas in EKF SLAM data association probabilities can easily be obtained from the posterior's covariance matrix, computing the same probabilities in GraphSLAM requires inference. This difference will be elucidated below, where we define an explicit algorithm for computing correspondences in GraphSLAM. Thus, which method is preferable is very much a question of the application, as there is no single method that would be superior in all critical dimensions.

This chapter first describes the intuition behind GraphSLAM and its basic updates steps. We then derive the various update steps mathematically and prove its correctness relative to specific linear approximations. A technique for data association will also be devised, followed by a discussion of actual implementations of the GraphSLAM algorithm.

11.2 Intuitive Description

The basic intuition behind GraphSLAM is remarkably simple: GraphSLAM extracts from the data a set of soft constraints, represented by a sparse graph. It obtains the map and the robot path by resolving these constraints into a globally consistent estimate. The constraints are generally nonlinear, but in the process of resolving them they are linearized and transformed into an information matrix. Thus, GraphSLAM is essentially an information-theoretic technique. We will describe GraphSLAM both as a technique for building a sparse graph of nonlinear constraints, and as a technique for populating a sparse information matrix of linearized constraints.

11.2.1 Building Up the Graph

Suppose we are given a set of measurements $z_{1:t}$ with associated correspondence variables $c_{1:t}$, and a set of controls $u_{1:t}$. GraphSLAM turns this data into a graph. The nodes in the graph are the robot poses $x_{1:t}$ and the features in the map $m = \{m_j\}$. Each edge in the graph corresponds to an event: a motion event generates an edge between two robot poses, and a measurement event creates a link between a pose and a feature in the map. Edges represent soft constraints between poses and features in GraphSLAM.

For a linear system, these constraints are equivalent to entries in an information matrix and an information vector of a large system of equations. As

usual, we will denote the information matrix by Ω and the information vector by ξ. As we shall see below, each measurement and each control leads to a *local* update of Ω and ξ, which corresponds to a local addition of an edge to the graph in GraphSLAM. In fact, the rule for incorporating a control or a measurement into Ω and ξ is a local addition, paying tribute to the important fact that information is an additive quantity.

Figure 11.2 illustrates the process of constructing the graph along with the corresponding information matrix. First consider a measurement z_t^i. This measurement provides information between the location of the feature $j = c_t^i$ and the robot pose x_t at time t. In GraphSLAM, this information is mapped into a constraint between x_t and m_j. We can think of this edge as a "spring" in a spring-mass model. As we shall see below, the constraint is of the type:

$$(z_t^i - h(x_t, m_j))^T \; Q_t^{-1} \; (z_t^i - h(x_t, m_j)) \tag{11.1}$$

Here h is the familiar measurement function, and Q_t is the covariance of the measurement noise. Figure 11.2a shows the addition of such a link into the graph maintained by GraphSLAM.

Now consider robot motion. The control u_t provides information about the relative value of the robot pose at time $t-1$ and the pose at time t. Again, this information induces a constraint in the graph, which will be of the form

$$(x_t - g(u_t, x_{t-1}))^T \; R_t^{-1} \; (x_t - g(u_t, x_{t-1})) \tag{11.2}$$

Here g is the familiar kinematic motion model of the robot, and R_t is the covariance of the motion noise.

Figure 11.2b illustrates the addition of such a link in the graph. It also shows the addition of a new element in the information matrix, between the pose x_t and the measurement z_t^i. This update is again additive. As before, the magnitude of these values reflects the residual uncertainty R_t due to the measurement noise; the less noisy the sensor, the larger the value added to Ω and ξ.

After incorporating all measurements $z_{1:t}$ and controls $u_{1:t}$, we obtain a sparse graph of soft constraints. The number of constraints in the graph is linear in the time elapsed, hence the graph is sparse. The sum of all constraints in the graph will be of the form

$$\begin{aligned} J_{\text{GraphSLAM}} \;=\; & x_0^T \; \Omega_0 \; x_0 \; + \; \sum_t (x_t - g(u_t, x_{t-1}))^T \; R_t^{-1} \; (x_t - g(u_t, x_{t-1})) \\ & + \sum_t \sum_i (z_t^i - h(y_t, c_t^i))^T \; Q_t^{-1} \; (z_t^i - h(y_t, c_t^i)) \end{aligned} \tag{11.3}$$

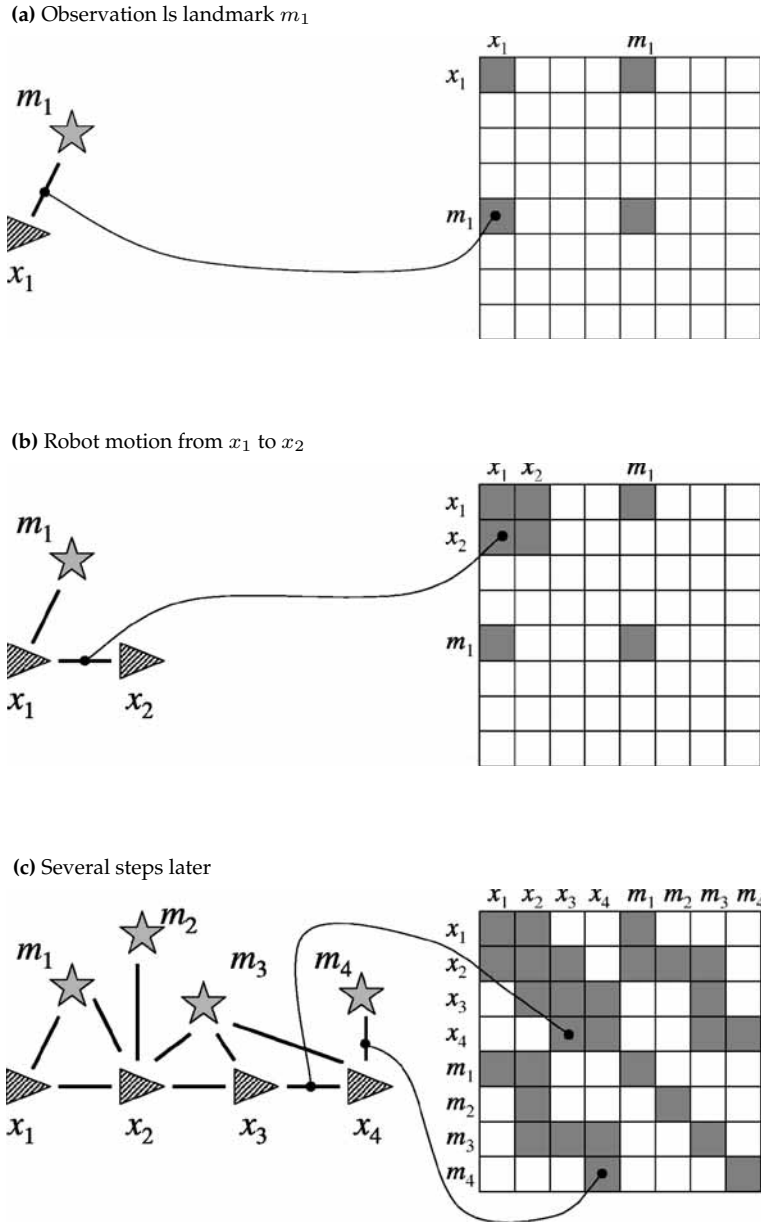


Figure 11.2 Illustration of the acquisition of the information matrix in GraphSLAM. The left diagram shows the dependence graph, the right the information matrix.

ANCHORING CONSTRAINT

It is a function defined over pose variables $x_{1:t}$ and all feature locations in the map m. Notice that this expression also features an *anchoring constraint* of the form $x_0^T \; \Omega_0 \; x_0$. This constraint anchors the absolute coordinates of the map by initializing the very first pose of the robot as $(0 \;\; 0 \;\; 0)^T$.

In the associated information matrix Ω, the off-diagonal elements are all zero with two exceptions: Between any two consecutive poses x_{t-1} and x_t will be a non-zero value that represents the information link introduced by the control u_t. Also non-zero will be any element between a map feature m_j and a pose x_t, if m_j was observed when the robot was at x_t. All elements between pairs of different features remain zero. This reflects the fact that we never received information pertaining to their relative location—all we receive in SLAM are measurements that constrain the location of a feature relative to a robot pose. Thus, the information matrix is equally sparse; all but a linear number of its elements are zero.

11.2.2 Inference

Of course, neither the graph representation nor the information matrix representation gives us what we want: the map and the path. In GraphSLAM, the map and the path are obtained from the linearized information matrix, via $\mu = \Omega^{-1}\xi$ (see Equation (3.73) on page 72). This operation requires us to solve a system of linear equations. This raises the question on how efficiently we can recover the map estimate μ and the covariance Σ.

The answer to the complexity question depends on the topology of the world. If each feature is seen only locally in time, the graph represented by the constraints is linear. Thus, Ω can be reordered so that it becomes a band-diagonal matrix, and all non-zero values occur near its diagonal. The equation $\mu = \Omega^{-1}\xi$ can then be computed in linear time. This intuition carries over to cycle-free world that is traversed once, so that each feature is seen for a short, consecutive period of time.

CYCLES

The more common case, however, involves features that are observed multiple times, with large time delays in between. This might be the case because the robot goes back and forth through a corridor, or because the world possesses *cycles*. In either situation, there will exist features m_j that are seen at drastically different time steps x_{t_1} and x_{t_2}, with $t_2 \gg t_1$. In our constraint graph, this introduces a cyclic dependence: x_{t_1} and x_{t_2} are linked through the sequence of controls $u_{t_1+1}, u_{t_1+2}, \ldots, u_{t_2}$ and through the joint observation links between x_{t_1} and m_j, and x_{t_2} and m_j, respectively. Such links make our variable reordering trick inapplicable, and recovering the map becomes

more complex. In fact, since the inverse of Ω is multiplied with a vector, the result can be computed with optimization techniques such as conjugate gradient, without explicitly computing the full inverse matrix. Since most worlds possess cycles, this is the case of interest.

FACTORIZATION

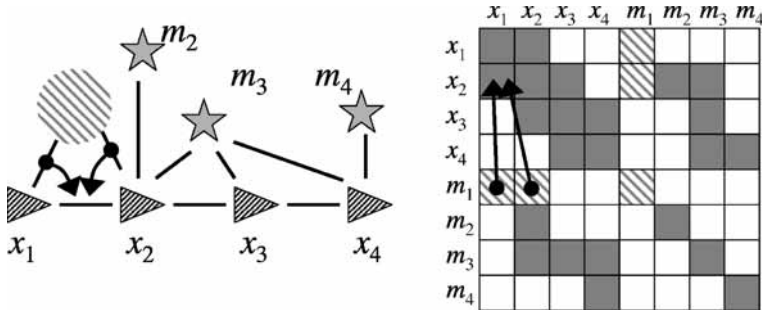
The GraphSLAM algorithm now employs an important *factorization trick*, which we can think of as propagating information through the information matrix (in fact, it is a generalization of the well-known *variable elimination algorithm* for matrix inversion). Suppose we would like to remove a feature m_j from the information matrix Ω and the information state ξ. In our spring mass model, this is equivalent to removing the node and all springs attached to this node. As we shall see below, this is possible by a remarkably simple operation: We can remove all those springs between m_j and the poses at which m_j was observed, by introducing new springs between any pair of such poses.

This process is illustrated in Figure 11.3, which shows the removal of two map features, m_1 and m_3 (the removal of m_2 and m_4 is trivial in this example). On both cases, the feature removal modifies the link between any pair of poses from which a feature was originally observed. As illustrated in Figure 11.3b, this operation may lead to the introduction of new links in the graph. In the example shown there, the removal of m_3 leads to a new link between x_2 and x_4.

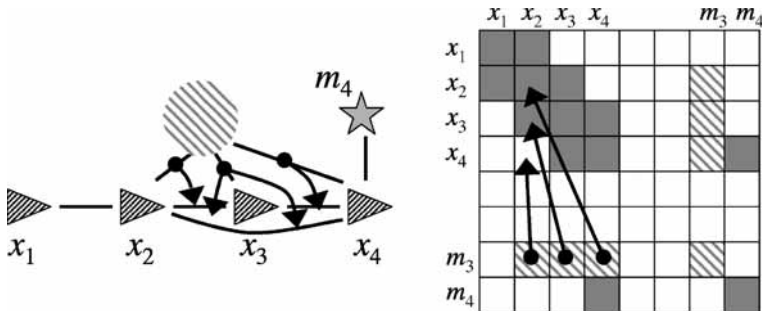
Formally, let $\tau(j)$ be the set of poses at which m_j was observed (that is: $x_t \in \tau(j) \iff \exists i : c_t^i = j$). Then we already know that the feature m_j is only linked to poses x_t in $\tau(j)$; by construction, m_j is *not* linked to any other pose, or to any feature in the map. We can now set all links between m_j and the poses $\tau(j)$ to zero by introducing a new link between any two poses $x_t, x_{t'} \in \tau(j)$. Similarly, the information vector values for all poses $\tau(j)$ are also updated. An important characteristic of this operation is that it is local: It only involves a small number of constraints. After removing all links to m_j, we can safely remove m_j from the information matrix and vector. The resulting information matrix is smaller—it lacks an entry for m_j. However, it is equivalent for the remaining variables, in the sense that the posterior defined by this information matrix is mathematically equivalent to the original posterior before removing m_j. This equivalence is intuitive: We simply have replaced springs connecting m_j to various poses in our spring mass model by a set of springs directly linking these poses. In doing so, the total force asserted by these springs remains equivalent, with the only exception that m_j is now disconnected.

The virtue of this reduction step is that we can gradually transform our

(a) The removal of m_1 changes the link between x_1 and x_2



(b) The removal of m_3 introduces a new link between x_2 and x_4



(c) Final Result after removing all map features

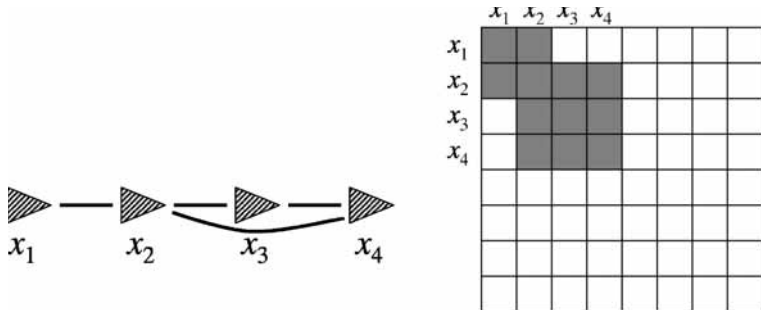


Figure 11.3 Reducing the graph in GraphSLAM: Arcs are removed to yield a network of links that only connect robot poses.

inference problem into a smaller one. By removing each feature m_j from Ω and ξ, we ultimately arrive at a much smaller information form $\tilde{\Omega}$ and $\tilde{\xi}$ defined only over the robot path variables. The reduction can be carried out in time linear in the size of the map; in fact, it generalizes the variable elimination technique for matrix inversion to the information form, in which we also maintain an information state. The posterior over the robot path is now recovered as $\tilde{\Sigma} = \tilde{\Omega}^{-1}$ and $\tilde{\mu} = \tilde{\Sigma}\xi$. Unfortunately, our reduction step does not eliminate cycles in the posterior. The remaining inference problem may still require more than linear time.

As a last step, GraphSLAM recovers the feature locations. Conceptually, this is achieved by building a new information matrix Ω_j and information vector ξ_j for each m_j. Both are defined over the variable m_j and the poses $\tau(j)$ at which m_j were observed. It contains the original links between m_j and $\tau(j)$, but the poses $\tau(j)$ are set to the values in $\tilde{\mu}$, without uncertainty. From this information form, it is now simple to calculate the location of m_j, using the common matrix inversion trick. Clearly, Ω_j contains only elements that connect to m_j; hence the inversion takes time linear in the number of poses in $\tau(j)$.

It should be apparent why the graph representation is such a natural representation. The full SLAM problem is solved by locally adding information into a large information graph, one edge at-a-time for each measurement z_t^i and each control u_t. To turn such information into an estimate of the map and the robot path, it is first linearized, then information between poses and features is gradually shifted to information between pairs of poses. The resulting structure only constraints the robot poses, which are then calculated using matrix inversion. Once the poses are recovered, the feature locations are calculated one-after-another, based on the original feature-to-pose information.

11.3 The GraphSLAM Algorithm

We will now make the various computational steps of the GraphSLAM precise. The *full GraphSLAM algorithm* will be described in a number of steps. The main difficulty in implementing the simple additive information algorithm pertains to the conversion of a conditional probability of the form $p(z_t^i \mid x_t, m)$ and $p(x_t \mid u_t, x_{t-1})$ into a link in the information matrix. The information matrix elements are all linear; hence this step involves linearizing $p(z_t^i \mid x_t, m)$ and $p(x_t \mid u_t, x_{t-1})$. In EKF SLAM, this linearization was

Algorithm GraphSLAM_initialize($u_{1:t}$):

$$\begin{pmatrix} \mu_{0,x} \\ \mu_{0,y} \\ \mu_{0,\theta} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

for all controls $u_t = (v_t \;\; \omega_t)^T$ *do*

$$\begin{pmatrix} \mu_{t,x} \\ \mu_{t,y} \\ \mu_{t,\theta} \end{pmatrix} = \begin{pmatrix} \mu_{t-1,x} \\ \mu_{t-1,y} \\ \mu_{t-1,\theta} \end{pmatrix} + \begin{pmatrix} -\frac{v_t}{\omega_t}\sin\mu_{t-1,\theta} + \frac{v_t}{\omega_t}\sin(\mu_{t-1,\theta} + \omega_t\Delta t) \\ \frac{v_t}{\omega_t}\cos\mu_{t-1,\theta} - \frac{v_t}{\omega_t}\cos(\mu_{t-1,\theta} + \omega_t\Delta t) \\ \omega_t\Delta t \end{pmatrix}$$

endfor

return $\mu_{0:t}$

Table 11.1 Initialization of the mean pose vector $\mu_{1:t}$ in the GraphSLAM algorithm.

Algorithm GraphSLAM_linearize($u_{1:t}, z_{1:t}, c_{1:t}, \mu_{0:t}$):

set $\Omega = 0, \xi = 0$

add $\begin{pmatrix} \infty & 0 & 0 \\ 0 & \infty & 0 \\ 0 & 0 & \infty \end{pmatrix}$ *to* Ω *at* x_0

for all controls $u_t = (v_t \;\; \omega_t)^T$ *do*

$$\hat{x}_t = \mu_{t-1} + \begin{pmatrix} -\frac{v_t}{\omega_t}\sin\mu_{t-1,\theta} + \frac{v_t}{\omega_t}\sin(\mu_{t-1,\theta} + \omega_t\Delta t) \\ \frac{v_t}{\omega_t}\cos\mu_{t-1,\theta} - \frac{v_t}{\omega_t}\cos(\mu_{t-1,\theta} + \omega_t\Delta t) \\ \omega_t\Delta t \end{pmatrix}$$

$$G_t = \begin{pmatrix} 1 & 0 & -\frac{v_t}{\omega_t}\cos\mu_{t-1,\theta} + \frac{v_t}{\omega_t}\cos(\mu_{t-1,\theta} + \omega_t\Delta t) \\ 0 & 1 & -\frac{v_t}{\omega_t}\sin\mu_{t-1,\theta} + \frac{v_t}{\omega_t}\sin(\mu_{t-1,\theta} + \omega_t\Delta t) \\ 0 & 0 & 1 \end{pmatrix}$$

see next page for continuation

continued from the previous page

add $\begin{pmatrix} -G_t^T \\ 1 \end{pmatrix} R_t^{-1} \begin{pmatrix} -G_t & 1 \end{pmatrix}$ *to* Ω *at* x_t *and* x_{t-1}

add $\begin{pmatrix} -G_t^T \\ 1 \end{pmatrix} R_t^{-1} [\hat{x}_t - G_t \, \mu_{t-1}]$ *to* ξ *at* x_t *and* x_{t-1}

endfor

for all measurements z_t *do*

$$Q_t = \begin{pmatrix} \sigma_r^2 & 0 & 0 \\ 0 & \sigma_\phi^2 & 0 \\ 0 & 0 & \sigma_s^2 \end{pmatrix}$$

for all observed features $z_t^i = (r_t^i \;\; \phi_t^i \;\; s_t^i)^T$ *do*

$j = c_t^i$

$$\delta = \begin{pmatrix} \delta_x \\ \delta_y \end{pmatrix} = \begin{pmatrix} \mu_{j,x} - \mu_{t,x} \\ \mu_{j,y} - \mu_{t,y} \end{pmatrix}$$

$q = \delta^T \delta$

$$\hat{z}_t^i = \begin{pmatrix} \sqrt{q} \\ \operatorname{atan2}(\delta_y, \delta_x) - \mu_{t,\theta} \\ s_j \end{pmatrix}$$

$$H_t^i = \frac{1}{q} \begin{pmatrix} -\sqrt{q}\delta_x & -\sqrt{q}\delta_y & 0 & +\sqrt{q}\delta_x & \sqrt{q}\delta_y & 0 \\ \delta_y & -\delta_x & -q & -\delta_y & +\delta_x & 0 \\ 0 & 0 & 0 & 0 & 0 & q \end{pmatrix}$$

add $H_t^{iT} \, Q_t^{-1} \, H_t^i$ *to* Ω *at* x_t *and* m_j

add $H_t^{iT} \, Q_t^{-1} \left[z_t^i - \hat{z}_t^i + H_t^i \begin{pmatrix} \mu_{t,x} \\ \mu_{t,y} \\ \mu_{t,\theta} \\ \mu_{j,x} \\ \mu_{j,y} \\ \mu_{j,s} \end{pmatrix} \right]$ *to* ξ *at* x_t *and* m_j

endfor

endfor

return Ω, ξ

Table 11.2 Calculation of Ω and ξ in GraphSLAM.

Algorithm GraphSLAM_reduce(Ω, ξ):

$\tilde{\Omega} = \Omega$
$\tilde{\xi} = \xi$
for each feature j do
  let $\tau(j)$ be the set of all poses x_t at which j was observed
  subtract $\tilde{\Omega}_{\tau(j),j}\ \tilde{\Omega}_{j,j}^{-1}\ \xi_j$ from $\tilde{\xi}$ at $x_{\tau(j)}$ and m_j
  subtract $\tilde{\Omega}_{\tau(j),j}\ \tilde{\Omega}_{j,j}^{-1}\ \tilde{\Omega}_{j,\tau(j)}$ from $\tilde{\Omega}$ at $x_{\tau(j)}$ and m_j
  remove from $\tilde{\Omega}$ and $\tilde{\xi}$ all rows/columns corresponding to j
endfor
return $\tilde{\Omega}, \tilde{\xi}$

Table 11.3 Algorithm for reducing the size of the information representation of the posterior in GraphSLAM.

Algorithm GraphSLAM_solve($\tilde{\Omega}, \tilde{\xi}, \Omega, \xi$):

$\Sigma_{0:t} = \tilde{\Omega}^{-1}$
$\mu_{0:t} = \Sigma_{0:t}\ \tilde{\xi}$
for each feature j do
  set $\tau(j)$ to the set of all poses x_t at which j was observed
  $\mu_j = \Omega_{j,j}^{-1}\ (\xi_j + \Omega_{j,\tau(j)}\ \tilde{\mu}_{\tau(j)})$
endfor
return $\mu, \Sigma_{0:t}$

Table 11.4 Algorithm for updating the posterior μ.

```
Algorithm GraphSLAM_known_correspondence(u_{1:t}, z_{1:t}, c_{1:t}):
    μ_{0:t} = GraphSLAM_initialize(u_{1:t})
    repeat
        Ω, ξ = GraphSLAM_linearize(u_{1:t}, z_{1:t}, c_{1:t}, μ_{0:t})
        Ω̃, ξ̃ = GraphSLAM_reduce(Ω, ξ)
        μ, Σ_{0:t} = GraphSLAM_solve(Ω̃, ξ̃, Ω, ξ)
    until convergence
    return μ
```

Table 11.5 The GraphSLAM algorithm for the full SLAM problem with known correspondence.

found by calculating a Jacobian at the estimated mean poses $\mu_{0:t}$. To build our initial information matrix Ω and ξ, we need an initial estimate $\mu_{0:t}$ for all poses $x_{0:t}$.

There exist a number of solutions to the problem of finding an initial mean μ suitable for linearization. For example, we can run an EKF SLAM and use its estimate for linearization. For the sake of this chapter, we will use an even simpler technique: Our initial estimate will simply be provided by chaining together the motion model $p(x_t \mid u_t, x_{t-1})$. Such an algorithm is outlined in Table 11.1, and called there **GraphSLAM_initialize**. This algorithm takes the controls $u_{1:t}$ as input, and outputs sequence of pose estimates $\mu_{0:t}$. It initializes the first pose by zero, and then calculates subsequent poses by recursively applying the velocity motion model. Since we are only interested in the mean poses vector $\mu_{0:t}$, **GraphSLAM_initialize** only uses the deterministic part of the motion model. It also does not consider any measurement in its estimation.

Once an initial $\mu_{0:t}$ is available, the GraphSLAM algorithm constructs the full SLAM information matrix Ω and the corresponding information vector ξ. This is achieved by linearizing the links in the graph. The algorithm **GraphSLAM_linearize** is depicted in Table 11.2. This algorithm contains a good amount of mathematical notation, much of which shall become clear in our derivation of the algorithm further below. **GraphSLAM_linearize** accepts as an input the set of controls, $u_{1:t}$, the measurements $z_{1:t}$ and associated corre-

spondence variables $c_{1:t}$, and the mean pose estimates $\mu_{0:t}$. It then gradually constructs the information matrix Ω and the information vector ξ through linearization, by locally adding submatrices in accordance with the information obtained from each measurement and each control.

In particular, line 2 in **GraphSLAM_linearize** initializes the information elements. The "infinite" information entry in line 3 fixes the initial pose x_0 to $(0 \;\; 0 \;\; 0)^T$. It is necessary, since otherwise the resulting matrix becomes singular, reflecting the fact that from relative information alone we cannot recover absolute estimates.

Controls are integrated in lines 4 through 9 of **GraphSLAM_linearize**. The pose $\hat{x}$ and the Jacobian G_t calculated in lines 5 and 6 represent the linear approximation of the non-linear control function g. As obvious from these equations, this linearization step utilizes the pose estimates $\mu_{0:t-1}$, with $\mu_0 = (0 \;\; 0 \;\; 0)^T$. This leads to the updates for Ω, and ξ, calculated in lines 7, and 8, respectively. Both terms are added into the corresponding rows and columns of Ω and ξ. This addition realizes the inclusion of a new constraint into the SLAM posterior, very much along the lines of the intuitive description in the previous section.

Measurements are integrated in lines 10 through 21 of **GraphSLAM_linearize**. The matrix Q_t calculated in line 11 is the familiar measurement noise covariance. Lines 13 through 17 compute the Taylor expansion of the measurement function, here stated for the feature-based measurement model defined in Chapter 6.6. Attention has to be payed to the implementation of line 16, since the angular expressions can be shifted arbitrarily by 2π. This calculation culminates into the computation of the measurement update in lines 18 and 19. The matrix that is being added to Ω in line 18 is of dimension 6×6. To add it, we decompose it into a matrix of dimension 3×3 for the pose x_t, a matrix of dimension 3×3 for the feature m_j, and two matrices of dimension 3×3 and 3×3 for the link between x_t and m_j. Those are added to Ω at the corresponding rows and columns. Similarly, the vector added to the information vector ξ is of vertical dimension 5. It is also chopped into two vectors of size 3 and 2, and added to the elements corresponding to x_t and m_j, respectively. The result of **GraphSLAM_linearize** is an information vector ξ and a matrix Ω. We already noted that Ω is sparse. It contains only non-zero submatrices along the main diagonal, between subsequent poses, and between poses and features in the map. The running time of this algorithm is linear in t, the number of time steps at which data was accrued.

The next step of the GraphSLAM algorithm pertains to reducing the di-

mensionality of the information matrix/vector. This is achieved through the algorithm **GraphSLAM_reduce** in Table 11.3. This algorithm takes as input Ω and ξ defined over the full space of map features and poses, and outputs a reduced matrix $\tilde{\Omega}$ and vectors $\tilde{\xi}$ defined over the space of all poses (but not the map!). This transformation is achieved by removing features m_j one-at-a-time, in lines 4 through 9 of **GraphSLAM_reduce**. The bookkeeping of the exact indexes of each item in $\tilde{\Omega}$ and $\tilde{\xi}$ is a bit tedious, hence Table 11.3 only provides an intuitive account.

Line 5 calculates the set of poses $\tau(j)$ at which the robot observed feature j. It then extracts two submatrices from the present $\tilde{\Omega}$: $\tilde{\Omega}_{j,j}$ and $\tilde{\Omega}_{\tau(j),j}$. $\tilde{\Omega}_{j,j}$ is the quadratic submatrix between m_j and m_j, and $\tilde{\Omega}_{\tau(j),j}$ is composed of the off-diagonal elements between m_j and the pose variables $\tau(j)$. It also extracts from the information state vector $\tilde{\xi}$ the elements corresponding to the j-th feature, denoted here as ξ_j. It then subtracts information from $\tilde{\Omega}$ and $\tilde{\xi}$ as stated in lines 6 and 7. After this operation, the rows and columns for the feature m_j are zero. These rows and columns are then removed, reducing the dimension on $\tilde{\Omega}$ and $\tilde{\xi}$ accordingly. This process is iterated until all features have been removed, and only pose variables remain in $\tilde{\Omega}$ and $\tilde{\xi}$. The complexity of **GraphSLAM_reduce** is once again linear in t.

The last step in the GraphSLAM algorithm computes the mean and covariance for all poses in the robot path, and a mean location estimate for all features in the map. This is achieved through **GraphSLAM_solve** in Table 11.4. Lines 2 and 3 compute the path estimates $\mu_{0:t}$, by inverting the reduced information matrix $\tilde{\Omega}$ and multiplying the resulting covariance with the information vector. Subsequently, **GraphSLAM_solve** computes the location of each feature in lines 4 through 7. The return value of **GraphSLAM_solve** contains the mean for the robot path and all features in the map, but only the covariance for the robot path. We note that there exist other, more efficient ways to compute $\mu_{0:t}$ that bypass the matrix inversion step. Those will be discussed towards the end of this chapter, when applying standard optimization techniques to GraphSLAM.

The quality of the solution calculated by the GraphSLAM algorithm depends on the goodness of the initial mean estimates, calculated by **GraphSLAM_initialize**. The x- and y- components of these estimates affect the respective models in a linear way, hence the linearization does not depend on these values. Not so for the orientation variables in $\mu_{0:t}$. Errors in these initial estimates affect the accuracy of the Taylor approximation, which in turn affects the result.

To reduce potential errors due to the Taylor approximation in the lin-

earization, the procedures **GraphSLAM_linearize**, **GraphSLAM_reduce**, and **GraphSLAM_solve** are run multiple times over the same data set. Each iteration takes as an input an estimated mean vector $\mu_{0:t}$ from the previous iteration, and outputs a new, improved estimate. The iteration of the GraphSLAM optimization are only necessary when the initial pose estimates have high error (e.g., more than 20 degrees orientation error). A small number of iterations (e.g., 3) is usually sufficient.

Table 11.5 summarizes the resulting algorithm. It initializes the means, then repeats the construction step, the reduction step, and the solution step. Typically, two or three iterations suffice for convergence. The resulting mean μ is our best guess of the robot's path and the map.

11.4 Mathematical Derivation of GraphSLAM

The derivation of the GraphSLAM algorithm begins with a derivation of a recursive formula for calculating the full SLAM posterior, represented in information form. We then investigate each term in this posterior, and derive from them the additive SLAM updates through Taylor expansions. From that, we will derive the necessary equations for recovering the path and the map.

11.4.1 The Full SLAM Posterior

As in the discussion of EKF SLAM, it will be beneficial to introduce a variable for the augmented state of the full SLAM problem. We will use y to denote state variables that combine one or more poses x with the map m. In particular, we define $y_{0:t}$ to be a vector composed of the path $x_{0:t}$ and the map m, whereas y_t is composed of the momentary pose at time t and the map m:

$$y_{0:t} = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_t \\ m \end{pmatrix} \quad \text{and} \quad y_t = \begin{pmatrix} x_t \\ m \end{pmatrix} \tag{11.4}$$

The posterior in the full SLAM problem is $p(y_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t})$, where $z_{1:t}$ are the familiar measurements with correspondences $c_{1:t}$, and $u_{1:t}$ are the controls. Bayes rule enables us to factor this posterior:

$$p(y_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) \tag{11.5}$$

$$= \quad \eta\, p(z_t \mid y_{0:t}, z_{1:t-1}, u_{1:t}, c_{1:t})\; p(y_{0:t} \mid z_{1:t-1}, u_{1:t}, c_{1:t})$$

where η is the familiar normalizer. The first probability on the right-hand side can be reduced by dropping irrelevant conditioning variables:

$$p(z_t \mid y_{0:t}, z_{1:t-1}, u_{1:t}, c_{1:t}) \quad = \quad p(z_t \mid y_t, c_t) \tag{11.6}$$

Similarly, we can factor the second probability by partitioning $y_{0:t}$ into x_t and $y_{0:t-1}$, and obtain

$$\begin{aligned} & p(y_{0:t} \mid z_{1:t-1}, u_{1:t}, c_{1:t}) \\ & \quad = \quad p(x_t \mid y_{0:t-1}, z_{1:t-1}, u_{1:t}, c_{1:t})\; p(y_{0:t-1} \mid z_{1:t-1}, u_{1:t}, c_{1:t}) \\ & \quad = \quad p(x_t \mid x_{t-1}, u_t)\; p(y_{0:t-1} \mid z_{1:t-1}, u_{1:t-1}, c_{1:t-1}) \end{aligned} \tag{11.7}$$

Putting these expressions back into (11.5) gives us the recursive definition of the full SLAM posterior:

$$\begin{aligned} & p(y_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) \\ & \quad = \quad \eta\, p(z_t \mid y_t, c_t)\; p(x_t \mid x_{t-1}, u_t)\; p(y_{0:t-1} \mid z_{1:t-1}, u_{1:t-1}, c_{1:t-1}) \end{aligned} \tag{11.8}$$

The closed form expression is obtained through induction over t. Here $p(y_0)$ is the prior over the map m and the initial pose x_0.

$$\begin{aligned} p(y_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) \quad &= \quad \eta\, p(y_0) \prod_t p(x_t \mid x_{t-1}, u_t)\; p(z_t \mid y_t, c_t) \\ &= \quad \eta\, p(y_0) \prod_t \left[p(x_t \mid x_{t-1}, u_t) \prod_i p(z_t^i \mid y_t, c_t^i) \right] \end{aligned} \tag{11.9}$$

Here, as before, z_t^i is the i-th measurement in the measurement vector z_t at time t. The prior $p(y_0)$ factors into two independent priors, $p(x_0)$ and $p(m)$. In SLAM, we usually have no prior knowledge about the map m. We simply replace $p(y_0)$ by $p(x_0)$ and subsume the factor $p(m)$ into the normalizer η.

11.4.2 The Negative Log Posterior

The information form represents probabilities in logarithmic form. The log-SLAM posterior follows directly from the previous equation:

$$\begin{aligned} & \log p(y_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) \\ & \quad = \quad \text{const.} + \log p(x_0) \;+\; \sum_t \left[\log p(x_t \mid x_{t-1}, u_t) \;+\; \sum_i \log p(z_t^i \mid y_t, c_t^i) \right] \end{aligned} \tag{11.10}$$

Just as in Chapter 10, we assume the outcome of robot motion is distributed normally according to $\mathcal{N}(g(u_t, x_{t-1}), R_t)$, where g is the deterministic motion function, and R_t is the covariance of the motion error. Similarly, measurements z_t^i are generated according to $\mathcal{N}(h(y_t, c_t^i), Q_t)$, where h is the familiar measurement function and Q_t is the measurement error covariance. In equations, we have

$$p(x_t \mid x_{t-1}, u_t) = \eta \exp\left\{-\tfrac{1}{2}(x_t - g(u_t, x_{t-1}))^T R_t^{-1} (x_t - g(u_t, x_{t-1}))\right\} \tag{11.11}$$

$$p(z_t^i \mid y_t, c_t^i) = \eta \exp\left\{-\tfrac{1}{2}(z_t^i - h(y_t, c_t^i))^T Q_t^{-1} (z_t^i - h(y_t, c_t^i))\right\} \tag{11.12}$$

The prior $p(x_0)$ in (11.10) is also easily expressed by a Gaussian-type distribution. It *anchors* the initial pose x_0 to the origin of the global coordinate system: $x_0 = (0\ 0\ 0)^T$:

$$p(x_0) = \eta \exp\left\{-\tfrac{1}{2}\, x_0^T\, \Omega_0\, x_0\right\} \tag{11.13}$$

with

$$\Omega_0 = \begin{pmatrix} \infty & 0 & 0 \\ 0 & \infty & 0 \\ 0 & 0 & \infty \end{pmatrix} \tag{11.14}$$

For now, it shall not concern us that the value of ∞ cannot be implemented, as we can easily substitute ∞ with a large positive number. This leads to the following quadratic form of the negative log-SLAM posterior in (11.10):

$$\begin{aligned} & -\log p(y_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) \\ &= \text{const.} + \tfrac{1}{2}\Bigg[x_0^T\, \Omega_0\, x_0 + \sum_t (x_t - g(u_t, x_{t-1}))^T R_t^{-1} (x_t - g(u_t, x_{t-1})) \\ &\qquad + \sum_t \sum_i (z_t^i - h(y_t, c_t^i))^T Q_t^{-1} (z_t^i - h(y_t, c_t^i))\Bigg] \end{aligned} \tag{11.15}$$

This is essentially the same as $J_{\text{GraphSLAM}}$ in Equation (11.3), with a few differences pertaining to the omission of normalization constants (including a multiplication with -1). Equation (11.15) highlights an essential characteristic of the full SLAM posterior in the information form: It is composed of a number of quadratic terms, one for the prior, and one for each control and each measurement.

11.4.3 Taylor Expansion

The various terms Equation (11.15) are quadratic in the functions g and h, not in the variables we seek to estimate (poses and the map). GraphSLAM alle-

LINEARIZATION viates this problem by *linearizing* g and h via Taylor expansion—completely analogously to Equations (10.14) and (10.18) in the derivation of the EKF. In particular, we have:

$$g(u_t, x_{t-1}) \approx g(u_t, \mu_{t-1}) + G_t(x_{t-1} - \mu_{t-1}) \tag{11.16}$$

$$h(y_t, c_t^i) \approx h(\mu_t, c_t^i) + H_t^i\ (y_t - \mu_t) \tag{11.17}$$

Here μ_t is the current estimate of the state vector y_t, and $H_t^i = h_t^i\ F_{x,j}$ as defined already in Equation (10.19).

This linear approximation turns the log-likelihood (11.15) into a function that is quadratic in $y_{0:t}$. In particular, we obtain

$$\begin{aligned}\log p(y_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) &= \text{const.} - \tfrac{1}{2} \\ &\Bigg\{ x_0^T\ \Omega_0\ x_0 + \sum_t [x_t - g(u_t, \mu_{t-1}) - G_t(x_{t-1} - \mu_{t-1})]^T \\ &\qquad R_t^{-1}\ [x_t - g(u_t, \mu_{t-1}) - G_t(x_{t-1} - \mu_{t-1})] \\ &+ \sum_i [z_t^i - h(\mu_t, c_t^i) - H_t^i(y_t - \mu_t)]^T\ Q_t^{-1}\ [z_t^i - h(\mu_t, c_t^i) - H_t^i(y_t - \mu_t)] \Bigg\}\end{aligned} \tag{11.18}$$

This function is indeed a quadratic in $y_{0:t}$, and it shall prove convenient to reorder its terms, omitting several constant terms.

$$\begin{aligned}\log p(y_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) &= \text{const.} \\ &-\tfrac{1}{2} \underbrace{x_0^T\ \Omega_0\ x_0}_{\text{quadratic in } x_0} - \tfrac{1}{2} \sum_t \underbrace{x_{t-1:t}^T \begin{pmatrix} -G_t^T \\ 1 \end{pmatrix} R_t^{-1}\ (-G_t\ \ 1)\ x_{t-1:t}}_{\text{quadratic in } x_{t-1:t}} \\ &\quad + \underbrace{x_{t-1:t}^T \begin{pmatrix} -G_t^T \\ 1 \end{pmatrix} R_t^{-1}\ [g(u_t, \mu_{t-1}) - G_t\ \mu_{t-1}]}_{\text{linear in } x_{t-1:t}} \\ &-\tfrac{1}{2} \sum_i \underbrace{y_t^T\ H_t^{iT}\ Q_t^{-1}\ H_t^i\ y_t}_{\text{quadratic in } y_t} + \underbrace{y_t^T\ H_t^{iT}\ Q_t^{-1}\ [z_t^i - h(\mu_t, c_t^i) + H_t^i \mu_t]}_{\text{linear in } y_t}\end{aligned} \tag{11.19}$$

Here $x_{t-1:t}$ denotes the state vector concatenating x_{t-1} and x_t; hence we can write $(x_t - G_t\ x_{t-1})^T = x_{t-1:t}^T\ (-G_t\ \ 1)^T = x_{t-1:t}^T \begin{pmatrix} -G_t^T \\ 1 \end{pmatrix}$.

If we collect all quadratic terms into the matrix Ω, and all linear terms into a vector ξ, we see that expression (11.19) is of the form

$$\log p(y_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) = \text{const.} - \tfrac{1}{2}\ y_{0:t}^T\ \Omega\ y_{0:t} + y_{0:t}^T\ \xi \tag{11.20}$$

11.4.4 Constructing the Information Form

We can read off these terms directly from (11.19), and verify that they are indeed implemented in the algorithm **GraphSLAM_linearize** in Table 11.2:

- **Prior.** The initial pose prior manifests itself by a quadratic term Ω_0 over the initial pose variable x_0 in the information matrix. Assuming appropriate extension of the matrix Ω_0 to match the dimension of $y_{0:t}$, we have

$$\Omega \longleftarrow \Omega_0 \tag{11.21}$$

This initialization is performed in lines 2 and 3 of the algorithm **GraphSLAM_linearize**.

- **Controls.** From (11.19), we see that each control u_t adds to Ω and ξ the following terms, assuming that the matrices are rearranged so as to be of matching dimensions:

$$\Omega \longleftarrow \Omega + \begin{pmatrix} -G_t^T \\ 1 \end{pmatrix} R_t^{-1} \begin{pmatrix} -G_t & 1 \end{pmatrix} \tag{11.22}$$

$$\xi \longleftarrow \xi + \begin{pmatrix} -G_t^T \\ 1 \end{pmatrix} R_t^{-1} \left[g(u_t, \mu_{t-1}) - G_t \, \mu_{t-1} \right] \tag{11.23}$$

This is realized in lines 4 through 9 in **GraphSLAM_linearize**.

- **Measurements.** According to Equation (11.19), each measurement z_t^i transforms Ω and ξ by adding the following terms, once again assuming appropriate adjustment of the matrix dimensions:

$$\Omega \longleftarrow \Omega + H_t^{iT} \, Q_t^{-1} \, H_t^i \tag{11.24}$$

$$\xi \longleftarrow \xi + H_t^{iT} \, Q_t^{-1} \left[z_t^i - h(\mu_t, c_t^i) + H_t^i \mu_t \right] \tag{11.25}$$

This update occurs in lines 10 through 21 in **GraphSLAM_linearize**.

This proves the correctness of the construction algorithm **GraphSLAM_linearize**, relative to our Taylor expansion approximation.

We also note that the steps above only affect off-diagonal elements that involve at least one pose. Thus, all between-feature elements are zero in the resulting information matrix.

Marginals of a multivariate Gaussian. Let the probability distribution $p(x, y)$ over the random vectors x and y be a Gaussian represented in the information form

$$\Omega = \begin{pmatrix} \Omega_{xx} & \Omega_{xy} \\ \Omega_{yx} & \Omega_{yy} \end{pmatrix} \qquad \text{and} \qquad \xi = \begin{pmatrix} \xi_x \\ \xi_y \end{pmatrix}$$

If Ω_{yy} is invertible, the marginal $p(x)$ is a Gaussian whose information representation is

$$\bar{\Omega}_{xx} = \Omega_{xx} - \Omega_{xy}\, \Omega_{yy}^{-1}\, \Omega_{yx} \qquad \text{and} \qquad \bar{\xi}_x = \xi_x - \Omega_{xy}\, \Omega_{yy}^{-1}\, \xi_y$$

Proof. The marginal for a Gaussian in its moments parameterization

$$\Sigma = \begin{pmatrix} \Sigma_{xx} & \Sigma_{xy} \\ \Sigma_{yx} & \Sigma_{yy} \end{pmatrix} \qquad \text{and} \qquad \mu = \begin{pmatrix} \mu_x \\ \mu_y \end{pmatrix}$$

is $\mathcal{N}(\mu_x, \Sigma_{xx})$. By definition, the information matrix of this Gaussian is therefore Σ_{xx}^{-1}, and the information vector is $\Sigma_{xx}^{-1}\ \mu_x$. We show $\Sigma_{xx}^{-1} = \bar{\Omega}_{xx}$ via the Inversion Lemma from Table 3.2 on page 50. Let $P = (0\ \ 1)^T$, and let $[\infty]$ be a matrix of the same size as Ω_{yy} but whose entries are all infinite (with $[\infty]^{-1} = 0$). This gives us

$$(\Omega + P[\infty]P^T)^{-1} = \begin{pmatrix} \Omega_{xx} & \Omega_{xy} \\ \Omega_{yx} & [\infty] \end{pmatrix}^{-1} \overset{(*)}{=} \begin{pmatrix} \Sigma_{xx}^{-1} & 0 \\ 0 & 0 \end{pmatrix}$$

The same expression can also be expanded by the inversion lemma into:

$$\begin{aligned}
&(\Omega + P[\infty]P^T)^{-1} \\
&\quad = \Omega - \Omega\, P([\infty]^{-1} + P^T\, \Omega\, P)^{-1}\, P^T\, \Omega \\
&\quad = \Omega - \Omega\, P(0 + P^T\, \Omega\, P)^{-1}\, P^T\, \Omega \\
&\quad = \Omega - \Omega\, P(\Omega_{yy})^{-1}\, P^T\, \Omega \\
&\quad = \begin{pmatrix} \Omega_{xx} & \Omega_{xy} \\ \Omega_{yx} & \Omega_{yy} \end{pmatrix} - \begin{pmatrix} \Omega_{xx} & \Omega_{xy} \\ \Omega_{yx} & \Omega_{yy} \end{pmatrix} \begin{pmatrix} 0 & 0 \\ 0 & \Omega_{yy}^{-1} \end{pmatrix} \begin{pmatrix} \Omega_{xx} & \Omega_{xy} \\ \Omega_{yx} & \Omega_{yy} \end{pmatrix} \\
&\quad \overset{(*)}{=} \begin{pmatrix} \Omega_{xx} & \Omega_{xy} \\ \Omega_{yx} & \Omega_{yy} \end{pmatrix} - \begin{pmatrix} 0 & \Omega_{xy}\, \Omega_{yy}^{-1} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \Omega_{xx} & \Omega_{xy} \\ \Omega_{yx} & \Omega_{yy} \end{pmatrix} \\
&\quad = \begin{pmatrix} \Omega_{xx} & \Omega_{xy} \\ \Omega_{yx} & \Omega_{yy} \end{pmatrix} - \begin{pmatrix} \Omega_{xy}\, \Omega_{yy}^{-1}\, \Omega_{yx} & \Omega_{xy} \\ \Omega_{yx} & \Omega_{yy} \end{pmatrix} = \begin{pmatrix} \bar{\Omega}_{xx} & 0 \\ 0 & 0 \end{pmatrix}
\end{aligned}$$

see next page for continuation

continued from the previous page

The remaining statement, $\Sigma_{xx}^{-1}\ \mu_x = \bar{\xi}_x$, is obtained analogously, exploiting the fact that $\mu = \Omega^{-1}\xi$ (see Equation(3.73)) and the equality of the two expressions marked "$(*)$" above:

$$\begin{aligned}
\begin{pmatrix} \Sigma_{xx}^{-1}\ \mu_x \\ 0 \end{pmatrix} &= \begin{pmatrix} \Sigma_{xx}^{-1} & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \mu_x \\ \mu_y \end{pmatrix} = \begin{pmatrix} \Sigma_{xx}^{-1} & 0 \\ 0 & 0 \end{pmatrix} \Omega^{-1} \begin{pmatrix} \xi_x \\ \xi_y \end{pmatrix} \\
&\overset{(*)}{=} \left[\Omega - \begin{pmatrix} 0 & \Omega_{xy}\,\Omega_{yy}^{-1} \\ 0 & 1 \end{pmatrix} \Omega \right] \Omega^{-1} \begin{pmatrix} \xi_x \\ \xi_y \end{pmatrix} \\
&= \begin{pmatrix} \xi_x \\ \xi_y \end{pmatrix} - \begin{pmatrix} 0 & \Omega_{xy}\,\Omega_{yy}^{-1} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \xi_x \\ \xi_y \end{pmatrix} = \begin{pmatrix} \bar{\xi}_x \\ 0 \end{pmatrix}
\end{aligned}$$

Table 11.6 Lemma for marginalizing Gaussians in information form. The form of the covariance $\bar{\Omega}_{xx}$ in this lemma is also known as *Schur complement*.

Conditionals of a multivariate Gaussian. Let the probability distribution $p(x, y)$ over the random vectors x and y be a Gaussian represented in the information form

$$\Omega = \begin{pmatrix} \Omega_{xx} & \Omega_{xy} \\ \Omega_{yx} & \Omega_{yy} \end{pmatrix} \qquad \text{and} \qquad \xi = \begin{pmatrix} \xi_x \\ \xi_y \end{pmatrix}$$

The conditional $p(x \mid y)$ is a Gaussian with information matrix Ω_{xx} and information vector $\xi_x - \Omega_{xy}\ y$.

Proof. The result follows trivially from the definition of a Gaussian in information form:

$$\begin{aligned}
&p(x \mid y) \\
&= \eta\ \exp\left\{ -\tfrac{1}{2} \begin{pmatrix} x \\ y \end{pmatrix}^T \begin{pmatrix} \Omega_{xx} & \Omega_{xy} \\ \Omega_{yx} & \Omega_{yy} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} x \\ y \end{pmatrix}^T \begin{pmatrix} \xi_x \\ \xi_y \end{pmatrix} \right\} \\
&= \eta\ \exp\left\{ -\tfrac{1}{2} x^T \Omega_{xx} x - x^T \Omega_{xy} y - \tfrac{1}{2} y^T \Omega_{yy} y + x^T \xi_x + y^T \xi_y \right\} \\
&= \eta\ \exp\{ -\tfrac{1}{2} x^T \Omega_{xx} x + x^T (\xi_x - \Omega_{xy} y) \underbrace{-\tfrac{1}{2} y^T \Omega_{yy} y + y^T \xi_y}_{\text{const.}} \} \\
&= \eta\ \exp\{ -\tfrac{1}{2} x^T \Omega_{xx} x + x^T (\xi_x - \Omega_{xy} y) \}
\end{aligned}$$

Table 11.7 Lemma for conditioning Gaussians in information form.

11.4.5 Reducing the Information Form

The reduction step **GraphSLAM_reduce** is based on a factorization of the full SLAM posterior.

$$p(y_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) \quad = \quad p(x_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t})\; p(m \mid x_{0:t}, z_{1:t}, u_{1:t}, c_{1:t}) \tag{11.26}$$

Here $p(x_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) \sim \mathcal{N}(\xi, \Omega)$ is the posterior over paths alone, with the map integrated out:

$$p(x_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) \quad = \quad \int p(y_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t})\; dm \tag{11.27}$$

As we will show shortly, this probability is indeed calculated by the algorithm **GraphSLAM_reduce** in Table 11.3, since

$$p(x_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) \quad \sim \quad \mathcal{N}(\tilde{\xi}, \tilde{\Omega}) \tag{11.28}$$

In general, the integration in (11.27) will be intractable, due to the large number of variables in m. For Gaussians, this integral can be calculated in closed form. The key insight is given in Table 11.6, which states and proves the *marginalization lemma* for Gaussians.

Let us subdivide the matrix Ω and the vector ξ into submatrices, for the robot path $x_{0:t}$ and the map m:

$$\Omega \quad = \quad \begin{pmatrix} \Omega_{x_{0:t},x_{0:t}} & \Omega_{x_{0:t},m} \\ \Omega_{m,x_{0:t}} & \Omega_{m,m} \end{pmatrix} \tag{11.29}$$

$$\xi \quad = \quad \begin{pmatrix} \xi_{x_{0:t}} \\ \xi_m \end{pmatrix} \tag{11.30}$$

According to the *marginalization lemma*, the probability (11.28) is obtained as

$$\tilde{\Omega} \quad = \quad \Omega_{x_{0:t},x_{0:t}} - \Omega_{x_{0:t},m}\; \Omega_{m,m}^{-1}\; \Omega_{m,x_{0:t}} \tag{11.31}$$

$$\tilde{\xi} \quad = \quad \xi_{x_{0:t}} - \Omega_{x_{0:t},m}\; \Omega_{m,m}^{-1}\; \xi_m \tag{11.32}$$

The matrix $\Omega_{m,m}$ is block-diagonal. This follows from the way Ω is constructed, in particular the absence of any links between pairs of features. This makes the inversion efficient:

$$\Omega_{m,m}^{-1} \quad = \quad \sum_j F_j^T\; \Omega_{j,j}^{-1}\; F_j \tag{11.33}$$

where $\Omega_{j,j} \;=\; F_j \Omega F_j^T$ is the sub-matrix of Ω that corresponds to the j-th

feature in the map:

$$F_j = \begin{pmatrix} 0\cdots0 & 1\ 0\ 0 & 0\cdots0 \\ 0\cdots0 & 0\ 1\ 0 & 0\cdots0 \\ 0\cdots0 & \underbrace{0\ 0\ 1}_{j-\text{th feature}} & 0\cdots0 \end{pmatrix} \tag{11.34}$$

This insight makes it possible to decompose the implement Equations (11.31) and (11.32) into a sequential update:

$$\tilde{\Omega} = \Omega_{x_{0:t},x_{0:t}} - \sum_j \Omega_{x_{0:t},j}\ \Omega_{j,j}^{-1}\ \Omega_{j,x_{0:t}} \tag{11.35}$$

$$\tilde{\xi} = \xi_{x_{0:t}} - \sum_j \Omega_{x_{0:t},j}\ \Omega_{j,j}^{-1}\ \xi_j \tag{11.36}$$

The matrix $\Omega_{x_{0:t},j}$ is non-zero only for elements in $\tau(j)$, the set of poses at which feature j was observed. This essentially proves the correctness of the reduction algorithm **GraphSLAM_reduce** in Table 11.3. The operation performed on Ω in this algorithm can be thought of as the variable elimination algorithm for matrix inversion, applied to the feature variables but not the robot pose variables.

11.4.6 Recovering the Path and the Map

The algorithm **GraphSLAM_solve** in Table 11.4 calculates the mean and variance of the Gaussian $\mathcal{N}(\tilde{\xi}, \tilde{\Omega})$, using the standard equations, see Equations (3.72) and (3.73) on page 72:

$$\tilde{\Sigma} = \tilde{\Omega}^{-1} \tag{11.37}$$

$$\tilde{\mu} = \tilde{\Sigma}\ \tilde{\xi} \tag{11.38}$$

In particular, this operation provides us with the mean of the posterior on the robot path; it does not give us the locations of the features in the map.

It remains to recover the second factor of Equation (11.26):

$$p(m \mid x_{0:t}, z_{1:t}, u_{1:t}, c_{1:t}) \tag{11.39}$$

The *conditioning lemma*, stated and proved in Table 11.7, shows that this probability distribution is Gaussian with the parameters

$$\Sigma_m = \Omega_{m,m}^{-1} \tag{11.40}$$

$$\mu_m = \Sigma_m(\xi_m + \Omega_{m,x_{0:t}}\tilde{\xi}) \tag{11.41}$$

Here ξ_m and $\Omega_{m,m}$ are the subvector of ξ, and the submatrix of Ω, respectively, restricted to the map variables. The matrix $\Omega_{m,x_{0:t}}$ is the off-diagonal submatrix of Ω that connects the robot path to the map. As noted before, $\Omega_{m,m}$ is block-diagonal, hence we can decompose

$$p(m \mid x_{0:t}, z_{1:t}, u_{1:t}, c_{1:t}) = \prod_j p(m_j \mid x_{0:t}, z_{1:t}, u_{1:t}, c_{1:t}) \tag{11.42}$$

where each $p(m_j \mid x_{0:t}, z_{1:t}, u_{1:t}, c_{1:t})$ is distributed according to

$$\Sigma_j = \Omega_{j,j}^{-1} \tag{11.43}$$

$$\mu_j = \Sigma_j(\xi_j + \Omega_{j,x_{0:t}}\tilde{\mu}) = \Sigma_j(\xi_j + \Omega_{j,\tau(j)}\tilde{\mu}_{\tau(j)}) \tag{11.44}$$

The last transformation exploited the fact that the submatrix $\Omega_{j,x_{0:t}}$ is zero except for those pose variables $\tau(j)$ from which the j-th feature was observed.

It is important to notice that this is a Gaussian $p(m \mid x_{0:t}, z_{1:t}, u_{1:t}, c_{1:t})$ conditioned on the true path $x_{0:t}$. In practice, we do not know the path, hence one might want to know the posterior $p(m \mid z_{1:t}, u_{1:t}, c_{1:t})$ without the path in the conditioning set. This Gaussian cannot be factored in the moments parameterization, as locations of different features are correlated through the uncertainty over the robot pose. For this reason, **GraphSLAM_solve** returns the mean estimate of the posterior but only the covariance over the robot path. Luckily, we never need the full Gaussian in moments representation—which would involve a fully populated covariance matrix of massive dimensions—as all essential questions pertaining to the SLAM problem can be answered at least in approximation without knowledge of Σ.

11.5 Data Association in GraphSLAM

Data association in GraphSLAM is realized through correspondence variables, just as in EKF SLAM. GraphSLAM searches for a single best correspondence vector, instead of calculating an entire distribution over correspondences. Thus, finding a correspondence vector is a search problem. However, it shall prove convenient to define correspondences slightly differently in GraphSLAM than before: correspondences are defined over pairs of features in the map, rather than associations of measurements to features. Specifically, we say $c(j,k) = 1$ if m_j and m_k correspond to the same physical feature in the world. Otherwise, $c(j,k) = 0$. This feature-correspondence is in fact logically equivalent to the correspondence defined in the previous section, but it simplifies the statement of the basic algorithm.

The technique for searching the space of correspondences is greedy, just as in the EKF. Each step in the search of the best correspondence value leads to an improvement, as measured by the appropriate log-likelihood function. However, because GraphSLAM has access to all data at the same time, it is possible to devise correspondence techniques that are considerably more powerful than the incremental approach in the EKF. In particular:

1. At any point in the search, GraphSLAM can consider the correspondence of any set of features. There is no requirement to process the observed features sequentially.

2. Correspondence search can be combined with the calculation of the map. Assuming that two observed features correspond to the same physical feature in the world affects the resulting map. By incorporating such a correspondence hypothesis into the map, other correspondence hypotheses will subsequently look more or less likely.

3. Data association decisions in GraphSLAM can be *undone*. The goodness of a data association depends on the value of other data association decisions. What appears to be a good choice early on in the search may, at some later time in the search, turn out to be inferior. To accommodate such a situation, GraphSLAM can effectively undo a previous data association decision.

We will now describe one specific correspondence search algorithm that exploits the first two properties, but not the third. The data association algorithm will still be greedy, and it will sequentially search the space of possible correspondences to arrive at a plausible map. However, like all greedy algorithms, our approach is subject to local maxima; the true space of correspondences is of course exponential in the number of features in the map. Nevertheless, we will be content with a hill climbing algorithm and postpone the treatment of an exhaustive algorithm to the next chapter.

11.5.1 The GraphSLAM Algorithm with Unknown Correspondence

LIKELIHOOD TEST FOR CORRESPONDENCE

The key component of our algorithm is a *likelihood test for correspondence*. Specifically, GraphSLAM correspondence is based on a simple test: What is the probability that two different features in the map, m_j and m_k, correspond to the same physical feature in the world? If this probability exceeds a threshold, we will accept this hypothesis and merge both features in the map.

1: **Algorithm GraphSLAM_correspondence_test(**$\Omega, \xi, \mu, \Sigma_{0:t}, j, k$**):**

2: $\Omega_{[j,k]} = \Omega_{jk,jk} - \Omega_{jk,\tau(j,k)} \, \Sigma_{\tau(j,k),\tau(j,k)} \, \Omega_{\tau(j,k),jk}$

3: $\xi_{[j,k]} = \Omega_{[j,k]} \, \mu_{j,k}$

4: $\Omega_{\Delta j,k} = \begin{pmatrix} 1 \\ -1 \end{pmatrix}^T \Omega_{[j,k]} \begin{pmatrix} 1 \\ -1 \end{pmatrix}$

5: $\xi_{\Delta j,k} = \begin{pmatrix} 1 \\ -1 \end{pmatrix}^T \xi_{[j,k]}$

6: $\mu_{\Delta j,k} = \Omega^{-1}_{\Delta j,k} \, \xi_{\Delta j,k}$

7: *return* $|2\pi \, \Omega^{-1}_{\Delta j,k}|^{-\frac{1}{2}} \, \exp\left\{-\frac{1}{2} \, \mu^T_{\Delta j,k} \, \Omega^{-1}_{\Delta j,k} \, \mu_{\Delta j,k}\right\}$

Table 11.8 The GraphSLAM test for correspondence: It accepts as input an information representation of the SLAM posterior, along with the result of the **GraphSLAM_solve** step. It then outputs the posterior probability that m_j corresponds to m_k.

The algorithm for the correspondence test is depicted in Table 11.8: The input to the test are two feature indexes, j and k, for which we seek to compute the probability that those two features correspond to the same feature in the physical world. To calculate this probability, our algorithm utilizes a number of quantities: The information representation of the SLAM posterior, as manifest by Ω and ξ, and the result of the procedure **GraphSLAM_solve**, which is the mean vector μ and the path covariance $\Sigma_{0:t}$.

The correspondence test then proceeds in the following way: First, it computes the marginalized posterior over the two target features. This posterior is represented by the information matrix $\Omega_{[j,k]}$ and vector $\xi_{[j,k]}$ computed in lines 2 and 3 in Table 11.8. This step of the computation utilizes various subelements of the information form Ω, ξ, the mean feature locations as specified through μ, and the path covariance $\Sigma_{0:t}$. Next, it calculates the parameters of a new Gaussian random variable, whose value is the difference between m_j and m_k. Denoting the difference variable $\Delta_{j,k} = m_j - m_k$, the information parameters $\Omega_{\Delta j,k}, \xi_{\Delta j,k}$ are calculated in lines 4 and 5, and the correspond-

1: **Algorithm GraphSLAM($u_{1:t}, z_{1:t}$):**

2: *initialize all c_t^i with a unique value*
3: $\mu_{0:t} = \mathbf{GraphSLAM_initialize}(u_{1:t})$
4: $\Omega, \xi = \mathbf{GraphSLAM_linearize}(u_{1:t}, z_{1:t}, c_{1:t}, \mu_{0:t})$
5: $\tilde{\Omega}, \tilde{\xi} = \mathbf{GraphSLAM_reduce}(\Omega, \xi)$
6: $\mu, \Sigma_{0:t} = \mathbf{GraphSLAM_solve}(\tilde{\Omega}, \tilde{\xi}, \Omega, \xi)$
7: *repeat*
8: *for each pair of non-corresponding features m_j, m_k do*
9: $\pi_{j=k} = \mathbf{GraphSLAM_correspondence_test}(\Omega, \xi, \mu, \Sigma_{0:t}, j, k)$
10: *if $\pi_{j=k} > \chi$ then*
11: *for all $c_t^i = k$ set $c_t^i = j$*
12: $\Omega, \xi = \mathbf{GraphSLAM_linearize}(u_{1:t}, z_{1:t}, c_{1:t}, \mu_{0:t})$
13: $\tilde{\Omega}, \tilde{\xi} = \mathbf{GraphSLAM_reduce}(\Omega, \xi)$
14: $\mu, \Sigma_{0:t} = \mathbf{GraphSLAM_solve}(\tilde{\Omega}, \tilde{\xi}, \Omega, \xi)$
15: *endif*
16: *endfor*
17: *until no more pair m_j, m_k found with $\pi_{j=k} < \chi$*
18: *return μ*

Table 11.9 The GraphSLAM algorithm for the full SLAM problem with unknown correspondence. The inner loop of this algorithm can be made more efficient by selective probing feature pairs m_j, m_k, and by collecting multiple correspondences before solving the resulting collapsed set of equations.

ing expectation for the difference is computed in line 6. Line 7 returns the probability that the difference between m_j and m_k is zero.

The correspondence test provides us with an algorithm for performing data association search in GraphSLAM. Table 11.9 shows such an algorithm. It initializes the correspondence variables with unique values. The four steps

that follow (lines 3-7) are the same as in our GraphSLAM algorithm with known correspondence, stated in Table 11.5. However, this general SLAM algorithm then engages in the data association search. Specifically, for each pair of different features in the map, it calculates the probability of correspondence (line 9 in Table 11.9). If this probability exceeds a threshold χ, the correspondence vectors are set to the same value (line 11).

The GraphSLAM algorithm iterates the construction, reduction, and solution of the SLAM posterior (lines 12 through 14). As a result, subsequent correspondence tests factor in previous correspondence decisions though a newly constructed map. The map construction is terminated when no further features are found in its inner loop.

Clearly, the algorithm **GraphSLAM** is not particularly efficient. In particular, it tests all feature pairs for correspondence, not just nearby ones. Further, it reconstructs the map whenever a single correspondence is found; rather than processing sets of corresponding features in batch. Such modifications, however, are relatively straightforward. A good implementation of **GraphSLAM** will be more refined than our basic implementation discussed here.

11.5.2 Mathematical Derivation of the Correspondence Test

We essentially restrict our derivation to showing the correctness of the correspondence test in Table 11.8. Our first goal shall be to define a posterior probability distribution over a variable $\Delta_{j,k} = m_j - m_k$, the *difference* between the location of feature m_j and feature m_k. Two features m_j and m_k are equivalent if and only if their location is the same. Hence, by calculating the posterior probability of $\Delta_{j,k}$, we obtain the desired correspondence probability.

We obtain the posterior for $\Delta_{j,k}$ by first calculating the joint over m_j and m_k:

$$\begin{aligned} & p(m_j, m_k \mid z_{1:t}, u_{1:t}, c_{1:t}) \\ = & \int p(m_j, m_k \mid x_{1:t}, z_{1:t}, c_{1:t})\, p(x_{1:t} \mid z_{1:t}, u_{1:t}, c_{1:t})\, dx_{1:t} \end{aligned} \tag{11.45}$$

We will denote the information form of this marginal posterior by $\xi_{[j,k]}$ and $\Omega_{[j,k]}$. Note the use of the squared brackets, which distinguish these values from the submatrices of the joint information form.

The distribution (11.45) is obtained from the joint posterior over $y_{0:t}$, by applying the marginalization lemma. Specifically, Ω and ξ represent the joint posterior over the full state vector $y_{0:t}$ in information form, and $\tau(j)$ and $\tau(k)$

denote the sets of poses at which the robot observed feature j, and feature k, respectively. GraphSLAM gives us the mean pose vector $\tilde{\mu}$. To apply the marginalization lemma (Table 11.6), we shall leverage the result of the algorithm **GraphSLAM_solve**. Specifically, **GraphSLAM_solve** provides us already with a mean for the features m_j and m_k. We simply restate the computation here for the joint feature pair:

$$\mu_{[j,k]} = \Omega_{jk,jk}^{-1} \left(\xi_{jk} + \Omega_{jk,\tau(j,k)} \mu_{\tau(j,k)}\right) \tag{11.46}$$

Here $\tau(j,k) = \tau(j) \cup \tau(k)$ denotes the set of poses at which the robot observed m_j or m_k.

For the joint posterior, we also need a covariance. This covariance is *not* computed in **GraphSLAM_solve**, simply because the joint covariance over multiple features requires space quadratic in the number of features. However, for pairs of features the covariance of the joint is easily recovered.

Let $\Sigma_{\tau(j,k),\tau(j,k)}$ be the submatrix of the covariance $\Sigma_{0:t}$ restricted to all poses in $\tau(j,k)$. Here the covariance $\Sigma_{0:t}$ is calculated in line 2 of the algorithm **GraphSLAM_solve**. Then the marginalization lemma provides us with the marginal information matrix for the posterior over $(m_j \;\; m_k)^T$:

$$\Omega_{[j,k]} = \Omega_{jk,jk} - \Omega_{jk,\tau(j,k)} \; \Sigma_{\tau(j,k),\tau(j,k)} \; \Omega_{\tau(j,k),jk} \tag{11.47}$$

The information form representation for the desired posterior is now completed by the following information vector:

$$\xi_{[j,k]} = \Omega_{[j,k]} \; \mu_{[j,k]} \tag{11.48}$$

Hence we have for the joint

$$\begin{aligned} & p(m_j, m_k \mid z_{1:t}, u_{1:t}, c_{1:t}) \\ = \; & \eta \; \exp\left\{ -\tfrac{1}{2} \begin{pmatrix} m_j \\ m_k \end{pmatrix}^T \; \Omega_{[j,k]} \begin{pmatrix} m_j \\ m_k \end{pmatrix} + \begin{pmatrix} m_j \\ m_k \end{pmatrix}^T \; \xi_{[j,k]} \right\} \end{aligned} \tag{11.49}$$

These equations are identical to lines 2 and 3 in Table 11.8.

The nice thing about our representation is that it immediately lets us define the desired correspondence probability. For that, let us consider the random variable

$$\begin{aligned} \Delta_{j,k} &= m_j - m_k \\ &= \begin{pmatrix} 1 \\ -1 \end{pmatrix}^T \begin{pmatrix} m_j \\ m_k \end{pmatrix} \\ &= \begin{pmatrix} m_j \\ m_k \end{pmatrix}^T \begin{pmatrix} 1 \\ -1 \end{pmatrix} \end{aligned} \tag{11.50}$$

Plugging this into the definition of a Gaussian in information representation, we obtain:

$$
\begin{aligned}
(11.51)\quad & p(\Delta_{j,k} \mid z_{1:t}, u_{1:t}, c_{1:t}) \\
&= \eta \exp\left\{ -\tfrac{1}{2}\Delta_{j,k}^T \underbrace{\begin{pmatrix} 1 \\ -1 \end{pmatrix}^T \Omega_{[j,k]} \begin{pmatrix} 1 \\ -1 \end{pmatrix}}_{=:\,\Omega_{\Delta j,k}} \Delta_{j,k} + \Delta_{j,k}^T \underbrace{\begin{pmatrix} 1 \\ -1 \end{pmatrix}^T \xi_{[j,k]}}_{=:\,\xi_{\Delta j,k}} \right\} \\
&= \eta \ \exp\left\{ -\tfrac{1}{2}\ \Delta_{j,k}^T\ \Omega_{\Delta j,k}\ + \Delta_{j,k}^T\ \xi_{\Delta j,k} \right\}^T
\end{aligned}
$$

which is Gaussian with information matrix $\Omega_{\Delta j,k}$ and information vector $\xi_{\Delta j,k}$ as defined above. To calculate the probability that this Gaussian assumes the value of $\Delta_{j,k} = 0$, it shall be useful to rewrite this Gaussian in moments parameterization:

$$
\begin{aligned}
(11.52)\quad & p(\Delta_{j,k} \mid z_{1:t}, u_{1:t}, c_{1:t}) \\
&= |2\pi\ \Omega_{\Delta j,k}^{-1}|^{-\frac{1}{2}}\ \exp\left\{ -\tfrac{1}{2}\ (\Delta_{j,k} - \mu_{\Delta j,k})^T\ \Omega_{\Delta j,k}^{-1}\ (\Delta_{j,k} - \mu_{\Delta j,k}) \right\}
\end{aligned}
$$

where the mean is given by the obvious expression:

$$
(11.53)\quad \mu_{\Delta j,k} \quad = \quad \Omega_{\Delta j,k}^{-1}\ \xi_{\Delta j,k}
$$

These steps are found in lines 4 through 6 in Table 11.8.

The desired probability for $\Delta_{j,k} = 0$ is the result of plugging 0 into this distribution, and reading off the resulting probability:

$$
(11.54)\quad p(\Delta_{j,k} = 0 \mid z_{1:t}, u_{1:t}, c_{1:t}) \quad = \quad |2\pi\ \Omega_{\Delta j,k}^{-1}|^{-\frac{1}{2}}\ \exp\left\{ -\tfrac{1}{2}\ \mu_{\Delta j,k}^T\ \Omega_{\Delta j,k}^{-1}\ \mu_{\Delta j,k} \right\}
$$

This expression is the probability that two features in the map, m_j and m_k, correspond to the same features in the map. This calculation is implemented in line 7 in Table 11.8.

11.6 Efficiency Consideration

Practical implementations of GraphSLAM rely on a number of additional insights and techniques for improving efficiency. Possibly the biggest deficiency of GraphSLAM, as discussed thus far, is due to the fact that in the very beginning, we assume that all observed features constitute different features. Our algorithm unifies them one-by-one. For any reasonable number of features, such an approach will be unbearably slow. Further, it will neglect the

important constraint that at any point in time, the same feature can only be observed once, but not twice.

Existing implementations of the GraphSLAM idea exploit such opportunities.

Features that are immediately identified to correspond with high likelihood are often unified form the very beginning, before running the full GraphSLAM solution. For example, it is quite common to compile short segments into *local submaps*, e.g., local occupancy grid maps. GraphSLAM inference is then performed only between those local occupancy grid maps, where the match of two maps is taken as a probabilistic constraint between the relative poses of these maps. Such a hierarchical technique reduces the complexity of SLAM by orders of magnitude, while still retaining some of the key elements of GraphSLAM, specifically the ability to perform data association over large data sets.

LOCAL SUBMAPS

Many robots are equipped with sensors that observe large number of features at a time. For example, laser range finders observe dozens of features within a single scan. For any such scan, one commonly assumes that different measurements indeed correspond to different features in the environment, by virtue of the fact that each scan points in a different direction. This is known as the *mutual exclusion principle*, which was already discussed in Chapter 7. It therefore follows that $i \neq j \longrightarrow c_t^i \neq c_t^j$. No two measurements acquired within a single scan correspond to the same feature in the world.

Our pairwise data association technique above is unable to incorporate this constraint. Specifically, it may assign two measurements z_t^i and z_t^j to the same feature z_s^k for some $s \neq t$. To overcome this problem, it is common to associate entire measurement vectors z_t and z_s at the same time. This involves a calculation of a joint over all features in z_t and z_s. Such a calculation generalizes our pairwise calculation and is mathematically straightforward.

The GraphSLAM algorithm stated in this chapter does not make use of its ability to *undo a data association*. Once a data association decision is made, it cannot be reverted further down in the search. Mathematically, it is relatively straightforward to undo past data association decisions in the information framework. One can change the correspondence variables of any two measurements in arbitrary ways in our algorithm above. However, it is more difficult to test whether a data association should be undone, as there is no (obvious) test for testing whether two previously associated features should be distinct. A simple implementation involves undoing a data association in question, rebuilding the map, and testing whether our criterion above still calls for correspondence. Such an approach can be computation-

Figure 11.4 The Groundhog robot is a 1,500 pound custom-built vehicle equipped with onboard computing, laser range sensing, gas and sinkage sensors, and video recording equipment. The robot has been built to map abandoned mines.

ally involved, as it provides no means of detecting which data association to test. Mechanisms for detecting unlikely associations are outside the scope of this book, but should be considered when implementing this approach.

NEGATIVE INFORMATION

Finally, the GraphSLAM algorithm does not consider *negative information*. In practice, not seeing a feature can be as informative as seeing one. However, our simple formulation does not perform the necessary geometric computations.

In practice, whether or not we can exploit negative information depends on the nature of the sensor model, and the model of our features in the world. For example, we might have to compute probabilities of occlusion, which might be tricky for certain type sensors (e.g., range and bearing sensors for landmarks). However, contemporary implementations indeed consider negative information, but often by replacing proper probabilistic calculations through approximations. One such example will be given in the next section.

11.7 Empirical Implementation

We will now highlight empirical results for a GraphSLAM implementation. The vehicle used in our experiment is Figure 11.4; it is a robot designed to map abandoned mines.

The type map collected by the robot is shown in Figure 11.5. This map is an occupancy grid map, using effectively pairwise scan matching for recovering the robot's poses. Pairwise scan matching can be thought of as a version of

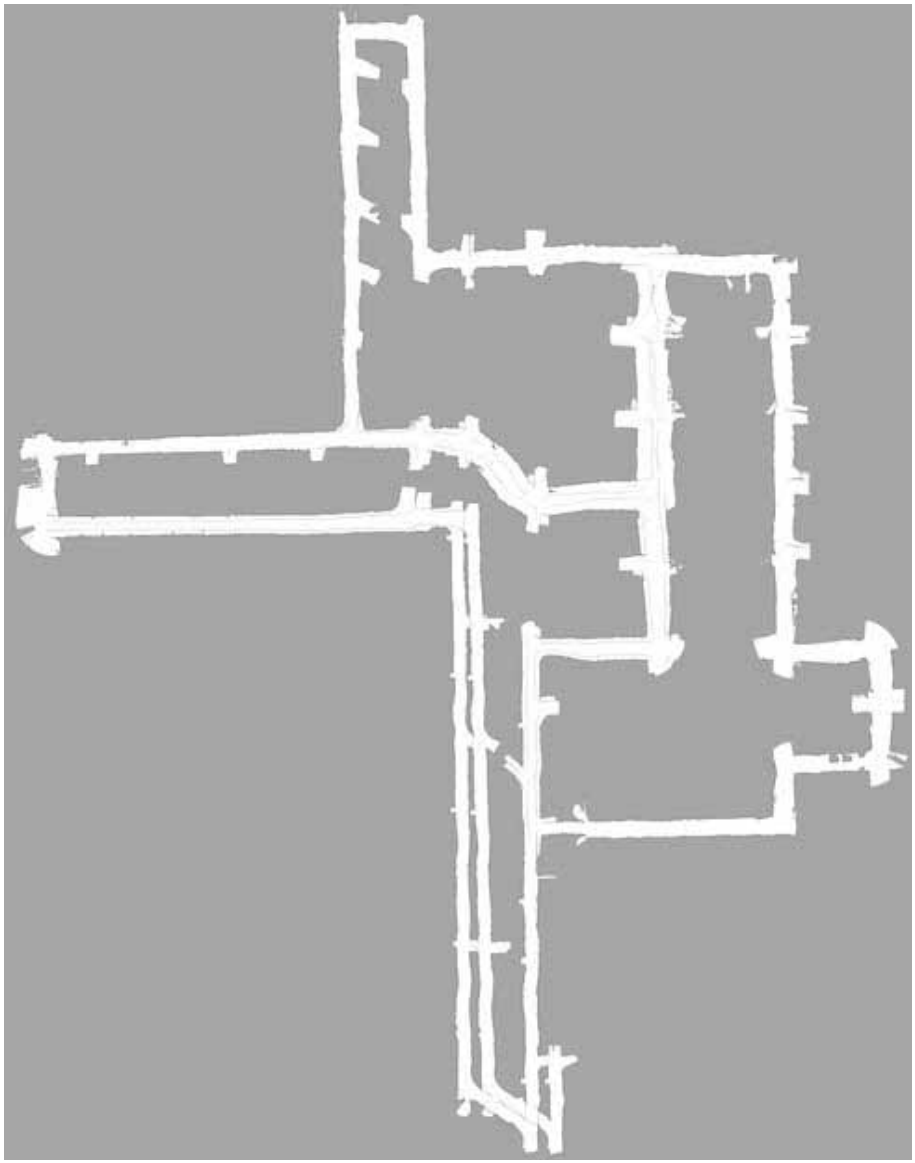


Figure 11.5 Map of a mine, acquired by pairwise scan matching. The diameter of this environment is approximately 250 meters. The map is obviously inconsistent, in that several hallways show up more than once. Image courtesy of Dirk Hähnel, University of Freiburg.

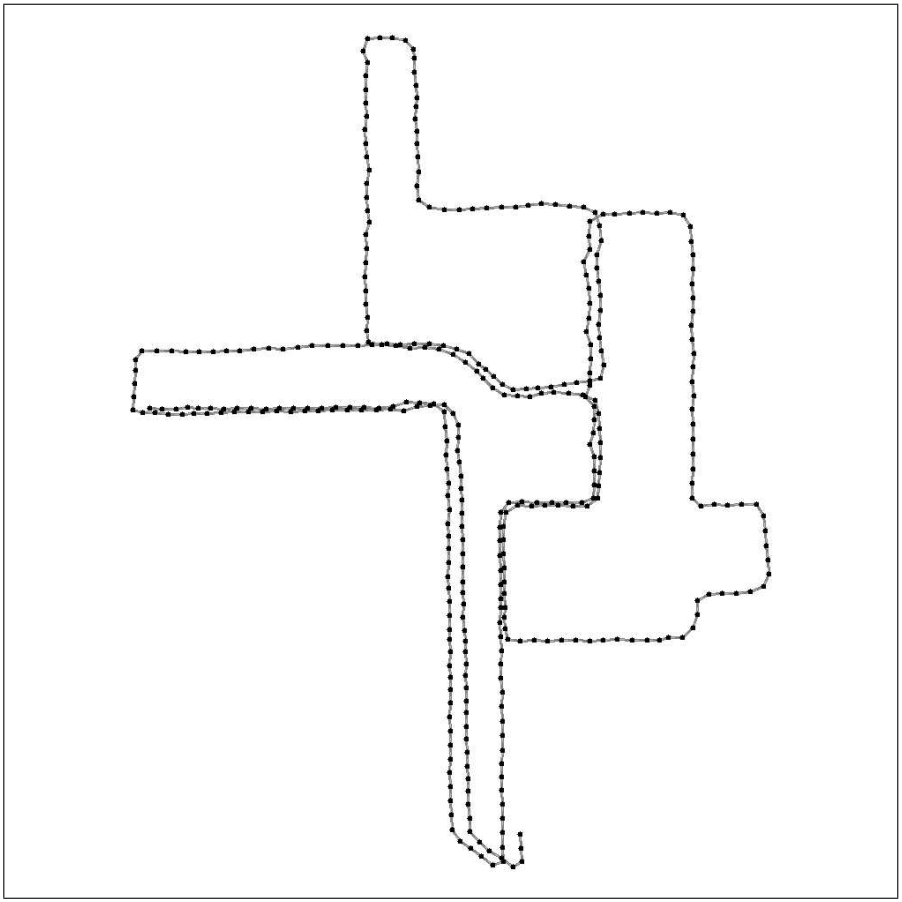


Figure 11.6 Mine map skeleton, visualizing the local maps.

GraphSLAM, but correspondence is only established between immediately consecutive scans. The result of this approach leads to an obvious deficiency of the map shown in Figure 11.5.

To apply the GraphSLAM algorithm, our software decomposes the map into small local submaps, one for each five meters of robot travel. Within these five meters, the maps are sufficiently accurate, as general drift is small and hence scan matching performs essentially flawlessly. Each submap's coordinates become a pose node in the GraphSLAM. Adjacent submaps are linked through the relative motion constraints between them. The resulting

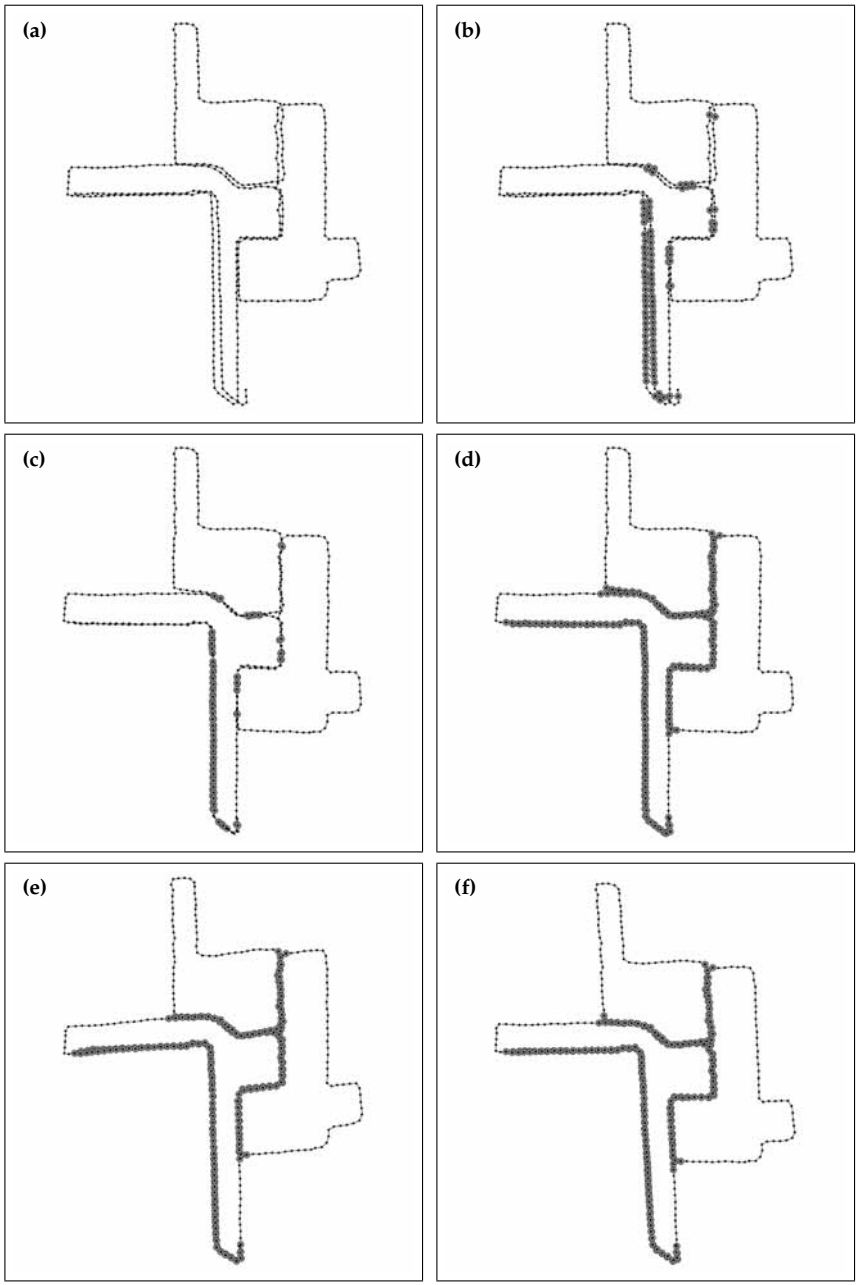


Figure 11.7 Data association search. See text.

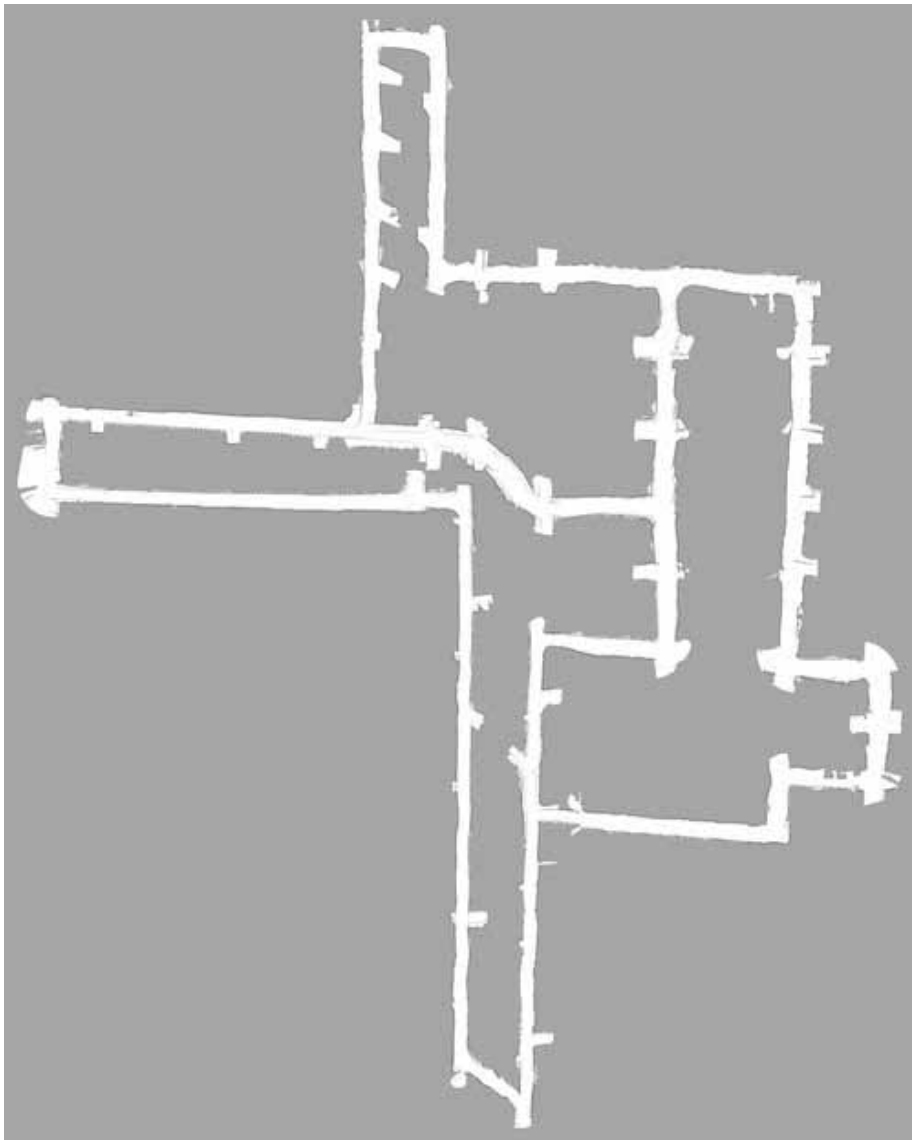


Figure 11.8 Final map, after optimizing for data associations. Image courtesy of Dirk Hähnel, University of Freiburg.

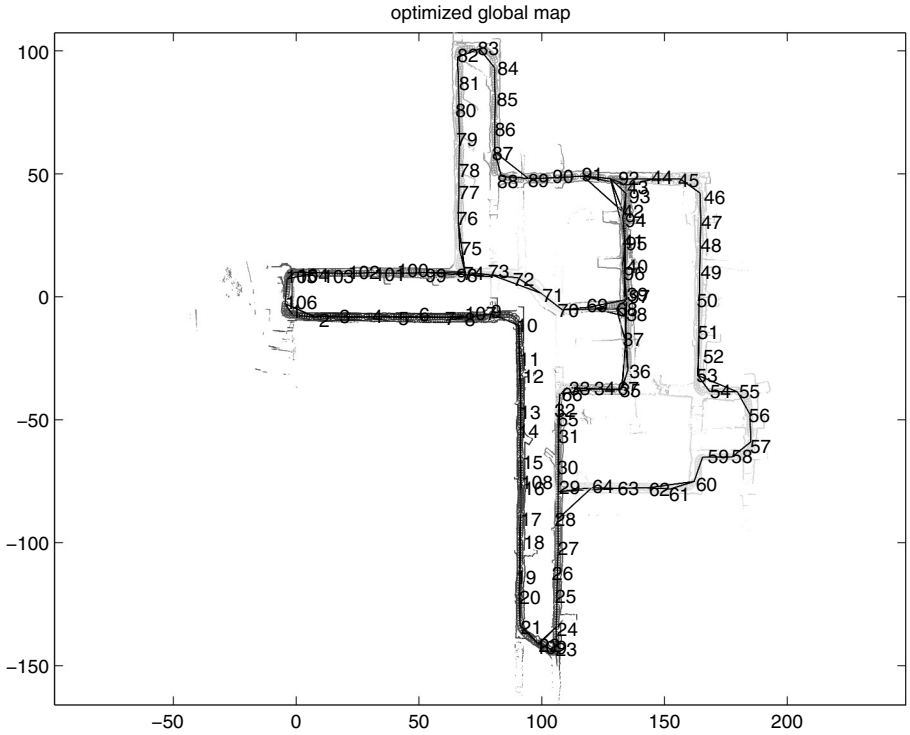


Figure 11.9 Mine map generated by the *Atlas* SLAM algorithm by Bosse et al. (2004). Image courtesy of Michael Bosse, Paul Newman, John Leonard, and Seth Teller, MIT.

structure is shown in Figure 11.6.

Next, we apply the recursive data association search. The correspondence test is now implemented using a correlation analysis for two overlaying maps, and the Gaussian matching constraints are recovered by approximating this match function through a Gaussian. Figure 11.7 illustrates the process of data association: The circles each correspond to a new constraint that is imposed when constructing the information form with GraphSLAM. This figure illustrates the iterative nature of the search: Certain correspondences are only discovered when others have been propagated, and others are dissolved in the process of the search. The final model is stable, in that additional search for new data association induces no further changes. Displayed as a grid map, it yields the 2-D map shown in Figure 11.8. While this map

is far from being perfect—largely due to a crude implementation of the local map matching constraints—it nevertheless is superior to the one found through incremental scan matching.

The reader should notice that other information-theoretic techniques for SLAM have produced similar results. Figure 11.9 shows a map of the same data set generated by Bosse et al. (2004), using an algorithm called *Atlas*. This algorithm decomposes maps into submaps whose relation is maintained through information-theoretic relative links. See the bibliographical remarks for more detail.

ATLAS

11.8 Alternative Optimization Techniques

The reader may recall that the central target function $J_{\text{GraphSLAM}}$ in GraphSLAM is the nonlinear quadratic function in Equation (11.3). GraphSLAM minimizes this function through a sequence of linearizations, variable eliminations, and optimizations. The inference technique in **GraphSLAM_solve** in Table 11.4 is generally not very efficient. If all one is interested in is a map and a path without covariances, the calculation of the inverse in line 2 in Table 11.4 can be avoided. The resulting implementation will be computationally much more efficient.

The key to efficient inference lies in the form of the function $J_{\text{GraphSLAM}}$. This function is of a general least squares form, and hence can be minimized by a number of different algorithms in the literature. Examples include *gradient descent techniques*, *Levenberg Marquardt*, and *conjugate gradient*.

CONJUGATE GRADIENT

Figure 11.10a shows results obtained using *conjugate gradient* for minimizing $J_{\text{GraphSLAM}}$. The data is a map of an outdoor environment of the approximate size 600 by 800 meters, collected on Stanford's campus with the robot shown in Figure 11.10b. Figure 11.11 illustrates the alignment process, from a data set that is based on pose data only, to a fully aligned map and robot path. This dataset contains approximately 10^8 features and 10^5 poses. Running an EKF would be infeasible on such a large data set. As would be inverting the matrix Ω in Table 11.4. Conjugate gradient required only a few seconds to minimize $J_{\text{GraphSLAM}}$. For this reason, many contemporary implementations of this approach use modern optimization techniques, instead of the relatively slow algorithm discussed here. The interested reader shall be referred to the bibliographical remarks for more pointers to alternative optimization techniques.

Figure 11.10 (a) A 3-D map of Stanford's campus. (b) The robot used for acquiring this data is based on a Segway RMP platform, whose development was funded by the DARPA MARS program. Image courtesy of Michael Montemerlo, Stanford University. *(Turn this page 90 degrees to the right to view this figure.)*

TESLA EXHIBIT 1011, Page 377

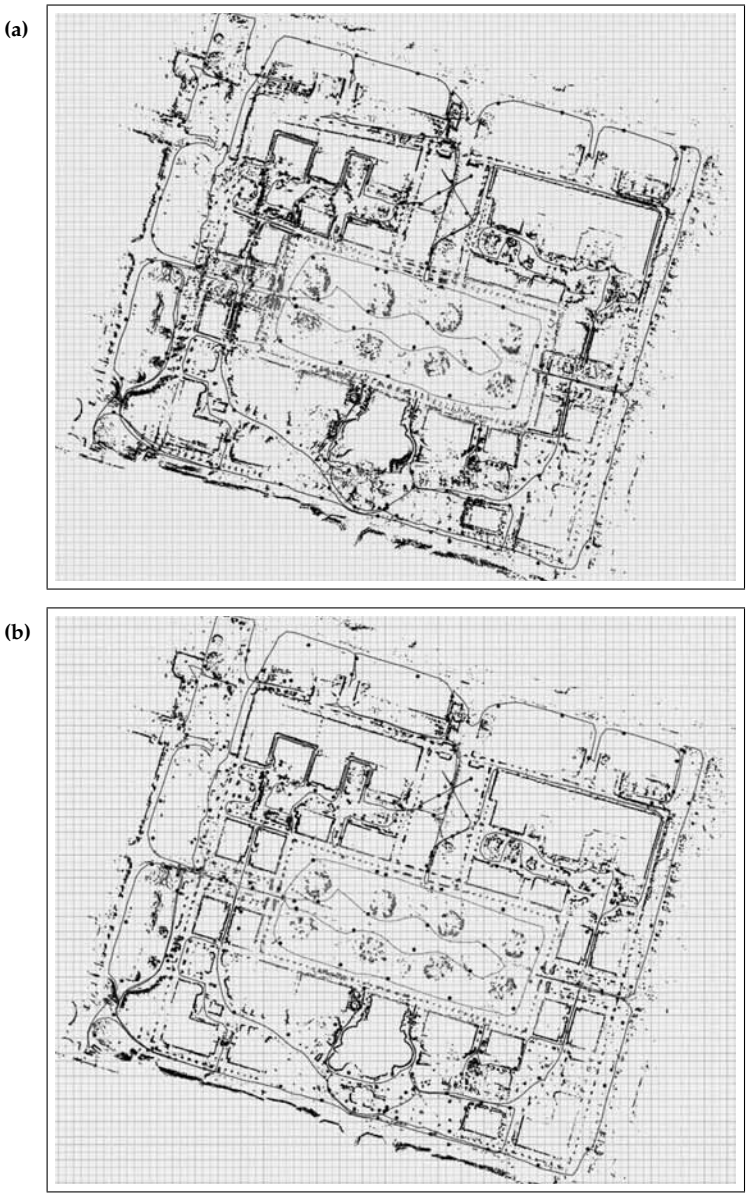


Figure 11.11 2-D slice through the Stanford campus map (a) before and (b) after alignment using conjugate gradient. Such an optimization takes only a few seconds with the conjugate gradient method applied to the least square formulation of Graph-SLAM. Images courtesy of Michael Montemerlo, Stanford University.

11.9 Summary

This chapter introduced the GraphSLAM algorithm to the full SLAM problem.

- GraphSLAM algorithm addresses the full SLAM problem. It calculates posteriors over the full robot path along with the map. Therefore, GraphSLAM is a batch algorithm, not an online algorithm like EKF SLAM.

- GraphSLAM constructs a graph of soft constraints from the data set. In particular, measurements are mapped into edges that represent nonlinear constraints between poses and sensed features, and motion commands map into soft constraints between consecutive poses. The graph is naturally sparse. The number of edges is a linear function of the number of nodes, and each node is only connected to finitely many other nodes, regardless of the size of the graph.

  GraphSLAM simply records all this information in the graph, through links that are defined between poses and features, and pairs of subsequent poses. However, this information representation does not provide estimates of the map or of the robot path.

- The sum of all such constraints is given by a function $J_{\text{GraphSLAM}}$. The maximum likelihood estimates for the robot path and the map can be obtained by minimizing this function $J_{\text{GraphSLAM}}$.

- GraphSLAM performs inference by mapping the graph into an isomorphic information matrix and information vector, defined over all pose variables and the entire map. The key insight of the GraphSLAM algorithm is that the structure of information is sparse. Measurements provide information of a feature relative to the robot's pose at the time of measurement. In information space, they form constraints between these pairs of variables. Similarly, motion provides information between two subsequent poses. In information space, each motion command forms a constraint between subsequent pose variables. This sparseness is inherited from the sparse graph.

- The vanilla GraphSLAM algorithm recovers maps through an iterative procedure that involves three steps: Construction of a linear information form through Taylor expansion, reduction of this form to remove the map, and solving the resulting optimization problem over robot poses. These three steps effectively resolve the information, and produce a consistent

probabilistic posterior over the path and the map. Since GraphSLAM is run batch, we can repeat the linearization step to improve the result.

- Alternative implementations perform inference through nonlinear least squares optimization of the function $J_{\text{GraphSLAM}}$. However, such techniques only find the mode of the posterior, not its covariance.

- Data association in GraphSLAM is performed by calculating the probability that two features have identical world coordinates. Since GraphSLAM is a batch algorithm, this can be done for any pair of features, at any time. This led to an iterative greedy search algorithm over all data association variables, which recursively identifies pairs of features in the map that likely correspond.

- Practical implementations of the GraphSLAM often use additional tricks to keep the computation low and to avoid false data associations. Specifically, practical implementations tend to reduce the data complexity by extracting local maps and using each map as the basic entity; they tend to match multiple features at-a-time, and they tend to consider negative information in the data association search.

- We briefly provided results for a variant of GraphSLAM that follows the decomposition idea, but uses occupancy grid maps for representing sets of range scans. Despite these approximations, we find that data association and inference techniques yield favorable results in a large-scale mapping problem.

- Results were also provided for a conjugate gradient implementation to the underlying least square problem. We noted that the general target function of GraphSLAM can be optimized by any least squares technique. Certain techniques, such as conjugate gradient, are significantly faster than the basic optimization technique in GraphSLAM.

As noted in the introduction, EKF SLAM and GraphSLAM are extreme ends of a spectrum of SLAM algorithms. Algorithms that fall in between these extremes will be discussed in the next two chapters. References to further techniques can be found in the bibliographical remarks of the next few chapters.

11.10 Bibliographical Remarks

Graphical inference techniques are well known in computer vision and photogrammetry, and are related to *structure from motion* and *bundle adjustment* (Hartley and Zisserman 2000; B et al. 2000; Mikhail et al. 2001). The first mention of relative, graph-like constraints in the SLAM literature goes back to Cheeseman and Smith (1986) and Durrant-Whyte (1988), but these approaches did not perform any global relaxation, or optimization. The algorithm presented in this chapter is loosely based on a seminal paper by Lu and Milios (1997). They were historically the first to represent the SLAM prior as a set of links between robot poses, and to formulate a global optimization algorithm for generating a map from such constraints. Their original algorithm for globally consistent range scan alignment used the robot pose variables as the frame of reference, which differed from the standard EKF view in which poses were integrated out. Through analyzing odometry and laser range scans, their approach generated relative constraints between poses that can be viewed as the edges in GraphSLAM; however, they did not phrase their method using information representations. Lu and Milios's (1997) algorithm was first successfully implemented by Gutmann and Nebel (1997), who reported numerical instabilities, possibly due to the extensive use of matrix inversion. Golfarelli et al. (1998) were the first to establish the relation of SLAM problems and spring-mass models, and Duckett et al. (2000, 2002) provided a first efficient technique for solving such problems. The relation between covariances and the information matrix is discussed in Frese and Hirzinger (2001). Araneda (2003) developed a more detailed elaborate graphical model.

The Lu and Milios algorithm initiated a development of offline SLAM algorithms that up to the present date runs largely parallel to the EKF work. Gutmann and Konolige combined their implementation with a Markov localization step for establishing correspondence when closing a loop in a cyclic environment. Bosse et al. (2003, 2004) developed Atlas, which is a hierarchical mapping framework based on the decoupled stochastic mapping paradigm, which retains relative information between submaps. It uses an optimization technique similar to the one in Duckett et al. (2000) and GraphSLAM when aligning multiple submaps. Folkesson and Christensen (2004a,b) exploited the optimization perspective of SLAM by applying gradient descent to the log-likelihood version of the SLAM posterior. Their *Graphical SLAM* algorithm reduced the number of variables to the path variables—just like GraphSLAM—when closing the loop. This reduction (which is mathematically an approximation since the map is simply omitted) significantly accelerated gradient descent. Konolige (2004) and Montemerlo and Thrun (2004) introduced *conjugate gradient* into the field of SLAM, which is known to be more efficient than gradient descent. Both also reduced the number of variables when closing large cycles, and report that maps with 10^8 features can be aligned in just a few seconds. The Levenberg Marquardt technique mentioned in the text is due to Levenberg (1944) and Marquardt (1963), who devised it in the context of least squares optimization. Frese et al. (2005) analyzed the efficiency of SLAM in the information form, and developed highly efficient optimization techniques using multi-grid optimization techniques. He reports speed-ups of several orders of magnitude; the resulting optimization techniques are presently the state-of-the-art. Dellaert (2005) developed efficient factorization techniques for the GraphSLAM constraint graph, specifically aimed at transforming the constraint graph into more compact versions while retaining sparseness.

It should be mentioned that the intuition to maintain relative links between local entities is at the core of many of the submapping techniques discussed in the previous section—although it is rarely made explicit. Authors such as Guivant and Nebot (2001); Williams (2001); Tardós et al. (2002); Bailey (2002) report of data structures for minuting the relative displacement between submaps, which are easily mapped to information theoretic concepts. While many of these

algorithms are filters, they nevertheless share a good amount of insight with the information form discussed in this chapter.

To our knowledge, the GraphSLAM algorithm presented here has never been published in the present form (an earlier draft of this book referred to this algorithm as *extended information form*). However, GraphSLAM is closely tied to the literature reviewed above, building on Lu and Milios's (1997) seminal algorithm. The name *GraphSLAM* bears resemblance to the name *Graphical SLAM* by Folkesson and Christensen (2004a); we have chosen it for this chapter because graphs of constraints are the essence of this entire line of SLAM research. A number of authors have developed *filters* in information form, which address the online SLAM problem instead of the full SLAM problem. These algorithms will be discussed in the coming chapter, which explicitly addresses the problem of filtering.

The GraphSLAM formulation of the SLAM problem relates to a decades-old discussion in the representation of spatial maps, which was already mentioned in the bibliographical remarks to Chapter 9. Information representations bring together two different paradigms of map representation: topological and metric. Behind this distinction is a decades-old debate about the representation of space, in people and for robots (Chown et al. 1995). Topological approaches were already discussed in the bibliographical remarks to Chapter 9. A key feature of topological representation pertains to the fact that they only specify *relative* information between entities in the map. Hence, they are free of the problem of finding a consistent metric embedding of this relative information. State-of-the-art topological tend to augment links with metric information, such as the distance between two locations.

Just like topological representations, information-theoretic methods accumulate relative information between adjacent objects (landmarks and robots). However, the relative map information is "translated" into metric embeddings by inference. In the linear Gaussian case, this inference step is loss-free and invertible. Computing the full posterior, including the covariance, requires matrix inversion. A second matrix inversion operation leads back to the original form of relative constraints. Thus, both the topological view and the metric view appear to be duals of each other, just like information-theoretic and probabilistic representations (or EKF and GraphSLAM). So maybe there shall be a unifying mathematical framework that embraces both, topological and metric maps? The reader shall be forewarned that this view has not yet been adopted by the mainstream research community.

11.11 Exercises

1. We already encountered *bearing only SLAM* in the exercises to the previous chapter, as a form of SLAM in which sensors can only measure the bearing of landmarks but not the range. We conjecture that GraphSLAM is better suited for this problem than the EKF. Why?

LINEAR GAUSSIAN SLAM

2. In this question, you are asked to prove convergence results for a special class of SLAM problems: *linear Gaussian SLAM*. In linear Gaussian SLAM, the motion equation is of the simple additive type

$$x_t \sim \mathcal{N}(x_{t-1} + u_t, R)$$

and the measurement equation is of the type

$$z_t = \mathcal{N}(m_j - x_t, Q)$$

where R and Q are diagonal covariances matrices, and m_j is the feature observed at time t. You may assume that the number of landmarks is finite, that all landmarks are seen infinitely often and in no specific order, and that the correspondence is known.

(a) Prove that for GraphSLAM the distance between any two landmarks converges to the correct distance with probability 1.

(b) What does this proof entail for EKF SLAM?

(c) Does GraphSLAM converge for the general SLAM problem with known correspondence? If so, argue why; if not, argue why not (without proof).

3. The algorithm **GraphSLAM_reduce** reduces the set of constraints by integrating out the map variables, leaving a constraint system over robot poses only. Is it possible to instead integrate out the pose variables, so that the resulting network of constraints is defined only over map variables? If so, what would the resulting inference problem be sparse? How would cycles in the robot's path affect this new set of constraints?

4. The GraphSLAM algorithm in this chapter ignored the landmark signatures. Extend the basic GraphSLAM algorithm to utilize such signatures in its measurements and in its map.

13 *The FastSLAM Algorithm*

We will now turn our attention to the *particle filter* approach to SLAM. We already encountered particle filters in several chapters of this book. We noted that particle filters are at the core of some of the most effective robotics algorithms. This raises the question as to whether particle filters are applicable to the SLAM problem. Unfortunately, particle filters are subject to the curse of dimensionality: whereas Gaussians scale between linearly and quadratically with the number of dimensions of the estimation problem, particle filters scale exponentially! A straightforward implementation of particle filters for the SLAM problem would be doomed to fail, due to the large number of variables involved in describing a map.

The algorithm in this chapter is based on an important characteristic of the SLAM problem, which has not yet been explicitly discussed in this book. Specifically, the full SLAM problem with known correspondences possesses a conditional independence between any two disjoint sets of features in the map, given the robot pose. Put differently, if an oracle told us the true robot path, we could estimate the location of all features independently of each other. Dependencies in these estimates arise *only* through robot pose uncertainty. This structural observation will make it possible to apply a version of particle filters to SLAM known as *Rao-Blackwellized particle filters*. Rao-Blackwellized particle filters use particles to represent the posterior over some variables, along with Gaussians (or some other parametric PDF) to represent all other variables.

RAO-BLACKWELLIZED PARTICLE FILTER

FastSLAM uses particle filters for estimating the robot path. As we shall see, for each of these particles the individual map errors are *conditionally independent*. Hence the mapping problem can be factored into many separate problems, one for each feature in the map. FastSLAM estimates these map feature locations by EKFs, but using a separate low-dimensional EKF

CONDITIONAL INDEPENDENCE

for each individual feature. This is fundamentally different from SLAM algorithms discussed in previous chapters, which all use a single Gaussian to estimate the location of all features jointly.

The basic algorithm can be implemented in time logarithmic in the number of features. Hence, FastSLAM offers computational advantages over plain EKF implementations and many of its descendants. The key advantage of FastSLAM, however, stems from the fact that data association decisions can be made on a per-particle basis. As a result, the filter maintains posteriors over multiple data associations, not just the most likely one. This is in stark contrast to all SLAM algorithms discussed so far, which track only a single data association at any point in time. In fact, by sampling over data associations, FastSLAM approximates the full posterior, not just the maximum likelihood data association. As shown empirically, the ability to pursue multiple data associations simultaneously makes FastSLAM significantly more robust to data association problems than algorithms based on incremental maximum likelihood data association.

Another advantage of FastSLAM over other SLAM algorithms arises from the fact that particle filters can cope with non-linear robot motion models, whereas previous techniques approximate such models via linear functions. This is important when the kinematics are highly non-linear, or when the pose uncertainty is relatively high.

The use of particle filters creates the unusual situation that FastSLAM solves both the *full SLAM problem* and the *online SLAM problem*. As we shall see, FastSLAM is formulated to calculate the full path posterior—only the full path renders feature locations conditionally independent. However, because particle filters estimate one pose at-a-time, FastSLAM is indeed an online algorithm. Hence it also solves the online SLAM problem. Among all SLAM algorithms discussed thus far, FastSLAM is the only algorithm that fits both categories.

This chapter describes several instantiations of the FastSLAM algorithm. FastSLAM 1.0 is the original FastSLAM algorithm, which is conceptually simple and easy to implement. In certain situations, however, the particle filter component of FastSLAM 1.0 generates samples inefficiently. The algorithm FastSLAM 2.0 overcomes this problem through an improved proposal distribution, but at the expense of an implementation that is significantly more involved (as is the mathematical derivation). Both of these FastSLAM algorithms assume the feature-based sensor model discussed earlier. The application of FastSLAM to range sensors results in an algorithm that solves the SLAM problem in the context of occupancy grid maps. For all algorithms,

this chapter provides techniques for estimating data association variables.

13.1 The Basic Algorithm

Particles in the basic FastSLAM algorithm are of the form shown in Table 13.1. Each particle contains an estimated robot pose, denoted $x_t^{[k]}$, and a set of Kalman filters with mean $\mu_{j,t}^{[k]}$ and covariance $\Sigma_{j,t}^{[k]}$, one for each feature m_j in the map. Here $[k]$ is the index of the particle. As usual, the total number of particles is denoted M.

The basic FastSLAM update step is depicted in Table 13.2. Barring the many details of the update step, the main loop is in large parts identical to the particle filter, as discussed in Chapter 4 of this book. The initial step involves the retrieval of a particle representing the posterior at time $t-1$, and the sampling of a robot pose for time t using the probabilistic motion model. The step that follows updates the EKFs for the observed features, using the standard EKF update equation. This update is not part of the vanilla particle filter, but it is necessary in FastSLAM to learn a map. The final steps are concerned with the calculation of an importance weight, which are then used to resample the particles.

We will now investigate each of these steps in more detail and derive them from the basic mathematical properties of the SLAM problem. We note that our derivation presupposes that FastSLAM solves the full SLAM problem, not the online problem. However, as shall become apparent further below in this chapter, FastSLAM is a solution to both of these problems: Each particle can be thought of as a sample in path space as required for the full SLAM problem, but the update only requires the most recent pose. For this reason, FastSLAM can be run just like a filter.

13.2 Factoring the SLAM Posterior

FastSLAM's key mathematical insight pertains to the fact that the full SLAM posterior, $p(y_{1:t} \mid z_{1:t}, u_{1:t})$ in Equation (10.2) can be written in the factored form

$$p(y_{1:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) \quad = \quad p(x_{1:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) \prod_{n=1}^{N} p(m_n \mid x_{1:t}, z_{1:t}, c_{1:t}) \tag{13.1}$$

This *factorization* states that the calculation of the posterior over paths and maps can be decomposed into $N+1$ probabilities.

	robot path	feature 1	feature 2	…	feature N
Particle $k=1$	$x_{1:t}^{[1]} = \{(x\ \ y\ \ \theta)^T\}_{1:t}^{[1]}$	$\mu_1^{[1]}, \Sigma_1^{[1]}$	$\mu_2^{[1]}, \Sigma_2^{[1]}$	…	$\mu_N^{[1]}, \Sigma_N^{[1]}$
Particle $k=2$	$x_{1:t}^{[2]} = \{(x\ \ y\ \ \theta)^T\}_{1:t}^{[2]}$	$\mu_1^{[2]}, \Sigma_1^{[2]}$	$\mu_2^{[2]}, \Sigma_2^{[2]}$	…	$\mu_N^{[2]}, \Sigma_N^{[2]}$
		$\vdots$			
Particle $k=M$	$x_{1:t}^{[M]} = \{(x\ \ y\ \ \theta)^T\}_{1:t}^{[M]}$	$\mu_1^{[M]}, \Sigma_1^{[M]}$	$\mu_2^{[M]}, \Sigma_2^{[M]}$	…	$\mu_N^{[M]}, \Sigma_N^{[M]}$

Figure 13.1 Particles in FastSLAM are composed of a path estimate and a set of estimators of individual feature locations with associated covariances.

- Do the following M times:
  - **Retrieval.** Retrieve a pose $x_{t-1}^{[k]}$ from the particle set Y_{t-1}.
  - **Prediction.** Sample a new pose $x_t^{[k]} \sim p(x_t \mid x_{t-1}^{[k]}, u_t)$.
  - **Measurement update.** For each observed feature z_t^i identify the correspondence j for the measurement z_t^i, and incorporate the measurement z_t^i into the corresponding EKF, by updating the mean $\mu_{j,t}^{[k]}$ and covariance $\Sigma_{j,t}^{[k]}$.
  - **Importance weight.** Calculate the importance weight $w^{[k]}$ for the new particle.
- **Resampling.** Sample, with replacement, M particles, where each particle is sampled with a probability proportional to $w^{[k]}$.

Figure 13.2 The basic steps of the FastSLAM algorithm.

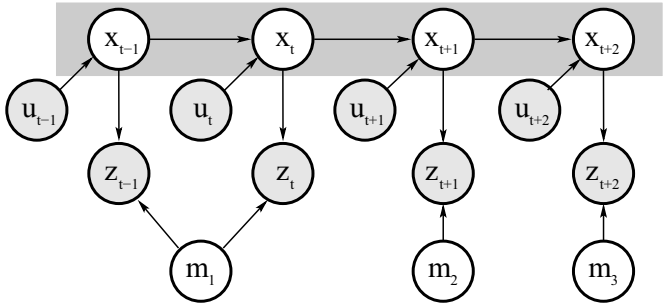


Figure 13.3 The SLAM problem depicted as Bayes network graph. The robot moves from pose x_{t-1} to pose x_{t+2}, driven by a sequence of controls. At each pose x_t it observes a nearby feature in the map $m = \{m_1, m_2, m_3\}$. This graphical network illustrates that the pose variables "separate" the individual features in the map from each other. If the poses are known, there remains no other path involving variables whose value is not known, between any two features in the map. This lack of a path renders the posterior of any two features in the map conditionally independent (given the poses).

FastSLAM uses a particle filter to compute the posterior over robot paths, denoted $p(x_{1:t} \mid z_{1:t}, u_{1:t}, c_{1:t})$. For each feature in the map, FastSLAM now uses a separate estimator over its location $p(m_n \mid x_{1:t}, c_{1:t}, z_{1:t})$ one for each $n = 1, \ldots, N$. Thus, in total there are $N + 1$ posteriors in FastSLAM. The feature estimators are conditioned on the robot path, which means we will have a separate copy of each feature estimator, one for each particle. With M particles, the number of filters will actually be $1 + MN$. The product of these probabilities represents the desired posterior in a factored way. As we shall show below, this factored representation is exact, not just an approximation. It is a generic characteristic of the SLAM problem.

To illustrate the correctness of this factorization, Figure 13.3 depicts the data acquisition process graphically, in the form of a dynamic Bayesian network. As this graph suggests, each measurement $z_1, \ldots, z_t$ is a functions of the position of the corresponding feature, along with the robot pose at the time the measurement was taken. Knowledge of the robot path separates the individual feature estimation problems and renders them independent of one another, in the sense that no direct path exists in this graphical depiction from one feature to another that would *not* involve variables on the robot's path. Knowledge of the exact location of one feature will therefore tell us nothing about the locations of other features, if the robot path is known.

This implies that features are *conditionally independent* given the robot path, as stated in Equation (13.1).

Before we discuss the implications of this property to the SLAM problem, let us briefly derive it mathematically.

13.2.1 Mathematical Derivation of the Factored SLAM Posterior

We will now derive Equation (13.1) from first principles. Clearly, we have

$$p(y_{1:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) = p(x_{1:t} \mid z_{1:t}, u_{1:t}, c_{1:t})\, p(m \mid x_{1:t}, z_{1:t}, c_{1:t}) \tag{13.2}$$

It therefore suffices to show that the second term on the right-hand side factors as follows:

$$p(m \mid x_{1:t}, c_{1:t}, z_{1:t}) = \prod_{n=1}^{N} p(m_n \mid x_{1:t}, c_{1:t}, z_{1:t}) \tag{13.3}$$

We will prove this by mathematical induction. Our derivation requires the distinction of two possible cases, depending on whether or not the feature m_n was observed in the most recent measurement. In particular, if $c_t \neq n$, the most recent measurement z_t has no effect on the posterior, and neither has the robot pose x_t or the correspondence c_t. Thus, we obtain:

$$p(m_n \mid x_{1:t}, c_{1:t}, z_{1:t}) = p(m_n \mid x_{1:t-1}, c_{1:t-1}, z_{1:t-1}) \tag{13.4}$$

If $c_t = n$ and hence $m_n = m_{c_t}$ was observed by the most recent measurement z_t, the situation calls for applying Bayes rule, followed by some standard simplifications:

$$\begin{aligned} p(m_{c_t} \mid x_{1:t}, c_{1:t}, z_{1:t}) &= \frac{p(z_t \mid m_{c_t}, x_{1:t}, c_{1:t}, z_{1:t-1})\, p(m_{c_t} \mid x_{1:t}, c_{1:t}, z_{1:t-1})}{p(z_t \mid x_{1:t}, c_{1:t}, z_{1:t-1})} \\ &= \frac{p(z_t \mid x_t, m_{c_t}, c_t)\, p(m_{c_t} \mid x_{1:t-1}, c_{1:t-1}, z_{1:t-1})}{p(z_t \mid x_{1:t}, c_{1:t}, z_{1:t-1})} \end{aligned} \tag{13.5}$$

This gives us the following expression for the probability of the observed feature m_{c_t}:

$$p(m_{c_t} \mid x_{1:t-1}, c_{1:t-1}, z_{1:t-1}) = \frac{p(m_{c_t} \mid x_{1:t}, c_{1:t}, z_{1:t})\, p(z_t \mid x_{1:t}, c_{1:t}, z_{1:t-1})}{p(z_t \mid x_t, m_{c_t}, c_t)} \tag{13.6}$$

The proof of the correctness of (13.3) is now carried out by induction. Let us assume that the posterior at time $t-1$ is already factored:

$$p(m \mid x_{1:t-1}, c_{1:t-1}, z_{1:t-1}) = \prod_{n=1}^{N} p(m_n \mid x_{1:t-1}, c_{1:t-1}, z_{1:t-1}) \tag{13.7}$$

This statement is trivially true for $t = 1$, since in the beginning of time the robot has no knowledge about any feature; hence all estimates must be independent. At time t, the posterior is of the following form:

$$
\begin{aligned}
p(m \mid x_{1:t}, c_{1:t}, z_{1:t}) &= \frac{p(z_t \mid m, x_{1:t}, c_{1:t}, z_{1:t-1})\, p(m \mid x_{1:t}, c_{1:t}, z_{1:t-1})}{p(z_t \mid x_{1:t}, c_{1:t}, z_{1:t-1})} \\
&= \frac{p(z_t \mid x_t, m_{c_t}, c_t)\, p(m \mid x_{1:t-1}, c_{1:t-1}, z_{1:t-1})}{p(z_t \mid x_{1:t}, c_{1:t}, z_{1:t-1})}
\end{aligned} \tag{13.8}
$$

Plugging in our inductive hypothesis (13.7) gives us:

$$
\begin{aligned}
& p(m \mid x_{1:t}, c_{1:t}, z_{1:t}) \\
&= \frac{p(z_t \mid x_t, m_{c_t}, c_t)}{p(z_t \mid x_{1:t}, c_{1:t}, z_{1:t-1})} \prod_{n=1}^{N} p(m_n \mid x_{1:t-1}, c_{1:t-1}, z_{1:t-1}) \\
&= \frac{p(z_t \mid x_t, m_{c_t}, c_t)}{p(z_t \mid x_{1:t}, c_{1:t}, z_{1:t-1})} \underbrace{p(m_{c_t} \mid x_{1:t-1}, c_{1:t-1}, z_{1:t-1})}_{Eq.\ (13.6)} \\
&\qquad \prod_{n \neq c_t} \underbrace{p(m_n \mid x_{1:t-1}, c_{1:t-1}, z_{1:t-1})}_{Eq.\ (13.4)} \\
&= p(m_{c_t} \mid x_{1:t}, c_{1:t}, z_{1:t}) \prod_{n \neq c_t} p(m_n \mid x_{1:t}, c_{1:t}, z_{1:t}) \\
&= \prod_{n=1}^{N} p(m_n \mid x_{1:t}, c_{1:t}, z_{1:t})
\end{aligned} \tag{13.9}
$$

Notice that we have substituted Equations (13.4) and (13.6) as indicated. This shows the correctness of Equation (13.3). The correctness of the main form (13.1) follows now directly from this result and the following generic transformation:

$$
\begin{aligned}
p(y_{1:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) &= p(x_{1:t} \mid z_{1:t}, u_{1:t}, c_{1:t})\, p(m \mid x_{1:t}, z_{1:t}, u_{1:t}, c_{1:t}) \\
&= p(x_{1:t} \mid z_{1:t}, u_{1:t}, c_{1:t})\, p(m \mid x_{1:t}, c_{1:t}, z_{1:t}) \\
&= p(x_{1:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) \prod_{n=1}^{N} p(m_n \mid x_{1:t}, c_{1:t}, z_{1:t})
\end{aligned} \tag{13.10}
$$

We note that conditioning on the entire path $x_{1:t}$ is indeed essential for this result. Conditioning on the most recent pose x_t would be insufficient as conditioning variable, as dependencies may arise through previous poses.

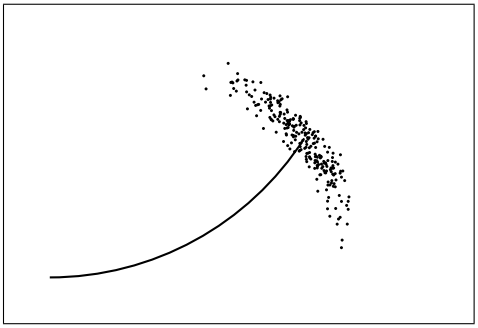


Figure 13.4 Samples drawn from the probabilistic motion model.

13.3 FastSLAM with Known Data Association

The factorial nature of the posterior provides us with significant computational advantages over SLAM algorithms that estimate an unstructured posterior distribution. FastSLAM exploits the factored representation by maintaining $MN + 1$ filters, M for each factor in (13.1). By doing so, all $MN + 1$ filters are low-dimensional.

As noted, FastSLAM estimates the path posterior using a particle filter. The map feature locations are estimated using EKFs. Because of our factorization, FastSLAM can maintain a separate EKF for each feature—which makes the update more efficient than in EKF SLAM. Each individual EKF is conditioned on a robot path. Hence, each particle possesses its own set of EKFs. In total there are NM EKFs, one for each feature in the map and one for each particle in the particle filter.

Let us begin with the FastSLAM algorithm in the case of known data association. *Particles* in FastSLAM will be denoted

$$Y_t^{[k]} = \left\langle x_t^{[k]},\ \mu_{1,t}^{[k]}, \Sigma_{1,t}^{[k]}, \ldots, \mu_{N,t}^{[k]}, \Sigma_{N,t}^{[k]} \right\rangle \tag{13.11}$$

As usual, the bracketed notation $[k]$ indicates the index of the particle; $x_t^{[k]}$ is the path estimate of the robot, and $\mu_{n,t}^{[k]}$ and $\Sigma_{n,t}^{[k]}$ are the mean and variance of the Gaussian representing the n-th feature location, relative to the k-th particle. Together, all these quantities form the k-th particle $Y_t^{[k]}$, of which there are a total of M in the FastSLAM posterior.

Filtering, or calculating the posterior at time t from the one at time $t-1$, involves generating a new particle set Y_t from Y_{t-1}, the particle set one time step earlier. This new particle set incorporates a new control u_t and a mea-

surement z_t with associated correspondence c_t. This update is performed in the following steps:

1. **Extending the path posterior by sampling new poses.** FastSLAM 1.0 uses the control u_t to sample new robot pose x_t for each particle in Y_{t-1}. More specifically, consider the k-the particle $Y_t^{[k]}$. FastSLAM 1.0 samples the pose x_t in accordance with this k-th particle, by drawing a sample according to the motion posterior

$$x_t^{[k]} \sim p(x_t \mid x_{t-1}^{[k]}, u_t) \tag{13.12}$$

Here $x_{t-1}^{[k]}$ is the posterior estimate for the robot location at time $t-1$, residing in the k-th particle. The resulting sample $x_t^{[k]}$ is then added to a temporary set of particles, along with the path of previous poses, $x_{1:t-1}^{[k]}$. The sampling step is graphically depicted in Figure 13.4, which illustrates a set of pose particles drawn from a single initial pose.

2. **Updating the observed feature estimate.** Next, FastSLAM 1.0 updates the posterior over the feature estimates, represented by the mean $\mu_{n,t-1}^{[k]}$ and the covariance $\Sigma_{n,t-1}^{[k]}$. The updated values are then added to the temporary particle set, along with the new pose.

The exact update equation depends on whether or not a feature m_n was observed at time t. For $n \neq c_t$ we did *not* observe feature n, we already established in Equation (13.4) that the posterior over the feature remains unchanged. This implies the simple update:

$$\left\langle \mu_{n,t}^{[k]}, \Sigma_{n,t}^{[k]} \right\rangle = \left\langle \mu_{n,t-1}^{[k]}, \Sigma_{n,t-1}^{[k]} \right\rangle \tag{13.13}$$

For the observed feature $n = c_t$, the update is specified through Equation (13.5), restated here with the normalizer denoted by η:

$$\begin{aligned} & p(m_{c_t} \mid x_{1:t}, z_{1:t}, c_{1:t}) \\ & \quad = \eta \, p(z_t \mid x_t, m_{c_t}, c_t) \, p(m_{c_t} \mid x_{1:t-1}, z_{1:t-1}, c_{1:t-1}) \end{aligned} \tag{13.14}$$

The probability $p(m_{c_t} \mid x_{1:t-1}, c_{1:t-1}, z_{1:t-1})$ at time $t-1$ is represented by a Gaussian with mean $\mu_{n,t-1}^{[k]}$ and covariance $\Sigma_{n,t-1}^{[k]}$. For the new estimate at time t to be Gaussian as well, FastSLAM linearizes the perceptual

model $p(z_t \mid x_t, m_{c_t}, c_t)$ in the same way as EKF SLAM. As usual, we approximate the measurement function h by Taylor expansion:

$$
\begin{aligned}
h(m_{c_t}, x_t^{[k]}) &\approx \underbrace{h(\mu_{c_t,t-1}^{[k]}, x_t^{[k]})}_{=:\, \hat{z}_t^{[k]}} + \underbrace{h'(x_t^{[k]}, \mu_{c_t,t-1}^{[k]})}_{=:\, H_t^{[k]}} (m_{c_t} - \mu_{c_t,t-1}^{[k]}) \\
&= \hat{z}_t^{[k]} + H_t^{[k]} \, (m_{c_t} - \mu_{c_t,t-1}^{[k]})
\end{aligned} \tag{13.15}
$$

Here the derivative h' is taken with respect to the feature coordinates m_{c_t}. This linear approximation is tangent to h at $x_t^{[k]}$ and $\mu_{c_t,t-1}^{[k]}$. Under this approximation, the posterior for the location of feature c_t is indeed Gaussian. The new mean and covariance are obtained using the standard EKF measurement update:

$$
K_t^{[k]} = \Sigma_{c_t,t-1}^{[k]} H_t^{[k]T} (H_t^{[k]} \Sigma_{c_t,t-1}^{[k]} H_t^{[k]T} + Q_t)^{-1} \tag{13.16}
$$

$$
\mu_{c_t,t}^{[k]} = \mu_{c_t,t-1}^{[k]} + K_t^{[k]} (z_t - \hat{z}_t^{[k]}) \tag{13.17}
$$

$$
\Sigma_{c_t,t}^{[k]} = (I - K_t^{[k]} H_t^{[k]}) \Sigma_{c_t,t-1}^{[k]} \tag{13.18}
$$

Steps 1 and 2 are repeated M times, resulting in a temporary set of M particles.

3. **Resampling.** In a final step, FastSLAM resamples this set of particles. We already encountered resampling in a number of algorithms. FastSLAM draws from its temporary set M particles (with replacement) according to a yet-to-be-defined importance weight. The resulting set of M particles then forms the new and final particle set, Y_t. The necessity to resample arises from the fact that the particles in the temporary set are not distributed according to the desired posterior: Step 1 generates poses x_t only in accordance with the most recent control u_t, paying no attention to the measurement z_t. As the reader should know well by now, resampling is the common technique in particle filtering to correct for such mismatches.

   This situation is once again illustrated in Figure 13.5, for a 1-D example. Here the dashed line symbolizes the *proposal distribution*, which is the distribution at which particles are generated, and the solid line is the target distribution. In FastSLAM, the proposal distribution does not depend on z_t, but the target distribution does. By weighing particles as shown in the bottom of this figure, and resampling according to those weights, the resulting particle set indeed approximates the target distribution.

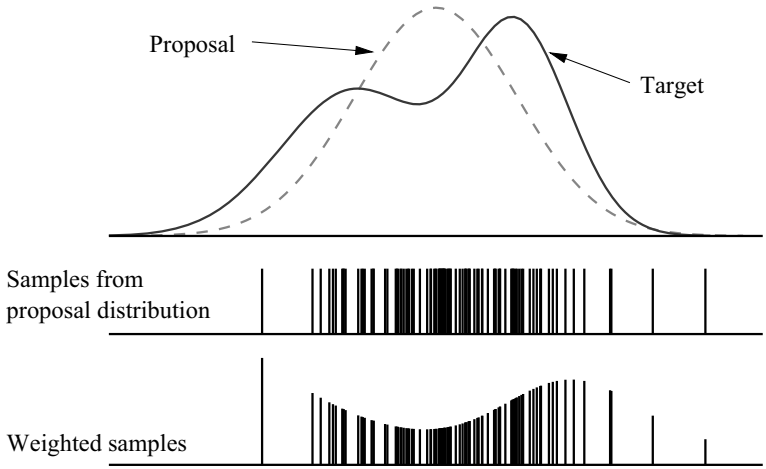


Figure 13.5 Samples cannot be drawn conveniently from the target distribution (shown as a solid line). Instead, the importance sampler draws samples from the proposal distribution (dashed line), which has a simpler form. Below, samples drawn from the proposal distribution are drawn with lengths proportional to their importance weights.

To determine the importance factor, it will prove useful to calculate the actual proposal distribution of the path particles in the temporary set. Under the assumption that the set of path particles in Y_{t-1} is distributed according to $p(x_{1:t-1} \mid z_{1:t-1}, u_{1:t-1}, c_{1:t-1})$ (which is an asymptotically correct approximation), path particles in the temporary set are distributed according to:

$$p(x_{1:t}^{[k]} \mid z_{1:t-1}, u_{1:t}, c_{1:t-1}) = p(x_t^{[k]} \mid x_{t-1}^{[k]}, u_t)\, p(x_{1:t-1}^{[k]} \mid z_{1:t-1}, u_{1:t-1}, c_{1:t-1}) \tag{13.19}$$

The factor $p(x_t^{[k]} \mid x_{t-1}^{[k]}, u_t)$ is the sampling distribution used in Equation (13.12).

The *target distribution* takes into account the measurement at time z_t, along with the correspondence c_t:

$$p(x_{1:t}^{[k]} \mid z_{1:t}, u_{1:t}, c_{1:t}) \tag{13.20}$$

The resampling process accounts for the difference of the target and the proposal distribution. As usual, the *importance factor* for resampling is

given by the quotient of the target and the proposal distribution:

$$
\begin{aligned}
w_t^{[k]} &= \frac{\text{target distribution}}{\text{proposal distribution}} \\
&= \frac{p(x_{1:t}^{[k]} \mid z_{1:t}, u_{1:t}, c_{1:t})}{p(x_{1:t}^{[k]} \mid z_{1:t-1}, u_{1:t}, c_{1:t-1})} \\
&= \eta\, p(z_t \mid x_t^{[k]}, c_t)
\end{aligned} \tag{13.21}
$$

The last transformation is a direct consequence of the following transformation of the enumerator in (13.21):

$$
\begin{aligned}
& p(x_{1:t}^{[k]} \mid z_{1:t}, u_{1:t}, c_{1:t}) \\
&= \eta\, p(z_t \mid x_{1:t}^{[k]}, z_{1:t-1}, u_{1:t}, c_{1:t})\, p(x_{1:t}^{[k]} \mid z_{1:t-1}, u_{1:t}, c_{1:t}) \\
&= \eta\, p(z_t \mid x_t^{[k]}, c_t)\, p(x_{1:t}^{[k]} \mid z_{1:t-1}, u_{1:t}, c_{1:t-1})
\end{aligned} \tag{13.22}
$$

To calculate the probability $p(z_t \mid x_t^{[k]}, c_t)$ in (13.21), it will be necessary to transform it further. In particular, this probability is equivalent to the following integration, where we once again omit variables irrelevant to the prediction of sensor measurements:

$$
\begin{aligned}
w_t^{[k]} &= \eta \int p(z_t \mid m_{c_t}, x_t^{[k]}, c_t)\, p(m_{c_t} \mid x_t^{[k]}, c_t)\, dm_{c_t} \\
&= \eta \int p(z_t \mid m_{c_t}, x_t^{[k]}, c_t)\, \underbrace{p(m_{c_t} \mid x_{1:t-1}^{[k]}, z_{1:t-1}, c_{1:t-1})}_{\sim\, \mathcal{N}(\mu_{c_t,t-1}^{[k]}, \Sigma_{c_t,t-1}^{[k]})}\, dm_{c_t}
\end{aligned} \tag{13.23}
$$

Here $\mathcal{N}(x; \mu, \Sigma)$ denotes a Gaussian distribution over the variable x with mean μ and covariance Σ.

The integration in (13.23) involves the estimate of the observed feature location at time t and the measurement model. To calculate (13.23) in closed form, FastSLAM employs the very same linear approximation used in the measurement update in Step 2. In particular, the importance factor is given by

$$
w_t^{[k]} \approx \eta\, |2\pi Q_t^{[k]}|^{-\frac{1}{2}}\, \exp\left\{-\tfrac{1}{2}(z_t - \hat{z}_t^{[k]}) Q_t^{[k]-1} (z_t - \hat{z}_t^{[k]})\right\} \tag{13.24}
$$

with the covariance

$$
Q_t^{[k]} = H_t^{[k]T} \Sigma_{n,t-1}^{[k]} H_t^{[k]} + Q_t \tag{13.25}
$$

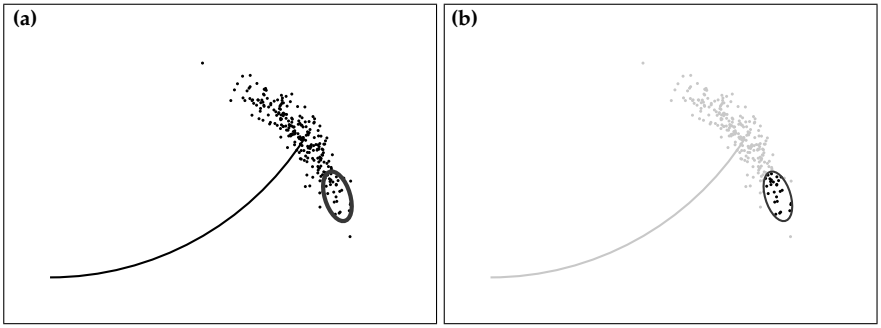


Figure 13.6 Mismatch between proposal and posterior distributions: (a) illustrates the forward samples generated by FastSLAM 1.0, and the posterior induced by the measurement (ellipse). Diagram (b) shows the sample set after the resampling step.

This expression is the probability of the actual measurement z_t under the Gaussian. It results from the convolution of the distributions in (13.23), exploiting our linear approximation of h. The resulting importance weights are used to draw with replacement M new samples from the temporary sample set. Through this resampling process, particles survive in proportion of their measurement probability.

These three steps together constitute the update rule of the FastSLAM 1.0 algorithm for SLAM problems with known data association. We note that the execution time of the update does not depend on the total path length t. In fact, only the most recent pose $x_{t-1}^{[k]}$ is used in the process of generating a new particle at time t. Consequently, past poses can safely be discarded. This has the pleasing consequence that neither the time requirements nor the memory requirements of FastSLAM depend on the total number of time steps spent on data acquisition.

A summary of the FastSLAM 1.0 algorithm with known data association is provided in Table 13.1. For simplicity, this implementation assumes that only a single feature is measured at each point in time. This algorithm implements the various update steps in a straightforward manner. Its implementation is relatively straightforward; in fact, FastSLAM 1.0 happens to be one of the easiest SLAM algorithms to implement!

Algorithm FastSLAM 1.0_known_correspondence(z_t, c_t, u_t, Y_{t-1}):

for $k = 1$ *to* M *do* // *loop over all particles*

  retrieve $\left\langle x_{t-1}^{[k]}, \left\langle \mu_{1,t-1}^{[k]}, \Sigma_{1,t-1}^{[k]} \right\rangle, \ldots, \left\langle \mu_{N,t-1}^{[k]}, \Sigma_{N,t-1}^{[k]} \right\rangle \right\rangle$ *from* Y_{t-1}

  $x_t^{[k]} \sim p(x_t \mid x_{t-1}^{[k]}, u_t)$ // *sample pose*

  $j = c_t$ // *observed feature*

  if feature j *never seen before*

    $\mu_{j,t}^{[k]} = h^{-1}(z_t, x_t^{[k]})$ // *initialize mean*

    $H = h'(x_t^{[k]}, \mu_{j,t}^{[k]})$ // *calculate Jacobian*

    $\Sigma_{j,t}^{[k]} = H^{-1}\ Q_t\ (H^{-1})^T$ // *initialize covariance*

    $w^{[k]} = p_0$ // *default importance weight*

  else

    $\hat{z} = h(\mu_{j,t-1}^{[k]}, x_t^{[k]})$ // *measurement prediction*

    $H = h'(x_t^{[k]}, \mu_{j,t-1}^{[k]})$ // *calculate Jacobian*

    $Q = H\ \Sigma_{j,t-1}^{[k]}\ H^T + Q_t$ // *measurement covariance*

    $K = \Sigma_{j,t-1}^{[k]}\ H^T\ Q^{-1}$ // *calculate Kalman gain*

    $\mu_{j,t}^{[k]} = \mu_{j,t-1}^{[k]} + K(z_t - \hat{z})$ // *update mean*

    $\Sigma_{j,t}^{[k]} = (I - K\ H)\Sigma_{j,t-1}^{[k]}$ // *update covariance*

    $w^{[k]} = |2\pi Q|^{-\frac{1}{2}}\ \exp\left\{-\frac{1}{2}(z_t - \hat{z}_n)^T\ Q^{-1}\ (z_t - \hat{z}_n)\right\}$ // *importance factor*

  endif

  for all other features $j' \neq j$ *do* // *unobserved features*

    $\mu_{j',t}^{[k]} = \mu_{j',t-1}^{[k]}$ // *leave unchanged*

    $\Sigma_{j',t}^{[k]} = \Sigma_{j',t-1}^{[k]}$

  endfor

endfor

$Y_t = \emptyset$ // *initialize new particle set*

do M *times* // *resample* M *particles*

  draw random k *with probability* $\propto w^{[k]}$ // *resample*

  add $\left\langle x_t^{[k]}, \left\langle \mu_{1,t}^{[k]}, \Sigma_{1,t}^{[k]} \right\rangle, \ldots, \left\langle \mu_N^{[k]}, \Sigma_N^{[k]} \right\rangle \right\rangle$ *to* Y_t

endfor

return Y_t

Table 13.1 FastSLAM 1.0 with known correspondence.

13.4 Improving the Proposal Distribution

FASTSLAM 2.0

FastSLAM 2.0 is largely equivalent to FastSLAM 1.0, with one important exception: Its proposal distribution takes the measurement z_t into account, when sampling the pose x_t. By doing so it can overcome a key limitation of FastSLAM 1.0.

On the surface, the difference looks rather marginal: The reader may recall that FastSLAM 1.0 samples poses based on the control u_t only, and then uses the measurement z_t to calculate importance weights. This is problematic when the accuracy of control is low relative to the accuracy of the robot's sensors. Such a situation is illustrated in Figure 13.6. Here the proposal generates a large spectrum of samples shown in Figure 13.6a, but only a small subset of these samples have high likelihood, as indicated by the ellipsoid. After resampling, only particles within the ellipsoid "survive" with reasonably high likelihood. FastSLAM 2.0 avoids this problem by sampling poses based on the measurement z_t in addition to the control u_t. Thus, as a result, FastSLAM 2.0 is more efficient than FastSLAM 1.0. On the downside, FastSLAM 2.0 is more difficult to implement than FastSLAM 1.0, and its mathematical derivation is more involved.

13.4.1 Extending the Path Posterior by Sampling a New Pose

In FastSLAM 2.0, the pose $x_t^{[k]}$ is drawn from the posterior

$$x_t^{[k]} \sim p(x_t \mid x_{1:t-1}^{[k]}, u_{1:t}, z_{1:t}, c_{1:t}) \tag{13.26}$$

This distribution differs from the proposal distribution provided in (13.12) in that (13.26) takes the measurement z_t into consideration, along with the correspondence c_t. Specifically, the expression in (13.26) conditions on $z_{1:t}$, whereas the pose sampler in FastSLAM 1.0 conditions on $z_{1:t-1}$.

Unfortunately, it also comes with more complex math. In particular, the mechanism for sampling from (13.26) requires further analysis. First, we rewrite (13.26) in terms of the "known" distributions, such as the measurement and motion models, and the Gaussian feature estimates in the k-th particle.

$$\begin{aligned} & p(x_t \mid x_{1:t-1}^{[k]}, u_{1:t}, z_{1:t}, c_{1:t}) \\ \overset{\text{Bayes}}{=} \quad & \frac{p(z_t \mid x_t, x_{1:t-1}^{[k]}, u_{1:t}, z_{1:t-1}, c_{1:t})\, p(x_t \mid x_{1:t-1}^{[k]}, u_{1:t}, z_{1:t-1}, c_{1:t})}{p(z_t \mid x_{1:t-1}^{[k]}, u_{1:t}, z_{1:t-1}, c_{1:t})} \\ = \quad & \eta^{[k]}\, p(z_t \mid x_t, x_{1:t-1}^{[k]}, u_{1:t}, z_{1:t-1}, c_{1:t})\, p(x_t \mid x_{1:t-1}^{[k]}, u_{1:t}, z_{1:t-1}, c_{1:t}) \end{aligned} \tag{13.27}$$

$$
\begin{aligned}
&\overset{\text{Markov}}{=} \quad \eta^{[k]}\; p(z_t \mid x_t, x^{[k]}_{1:t-1}, u_{1:t}, z_{1:t-1}, c_{1:t}); p(x_t \mid x^{[k]}_{t-1}, u_t) \\
&= \quad \eta^{[k]} \int p(z_t \mid m_{c_t}, x_t, x^{[k]}_{1:t-1}, u_{1:t}, z_{1:t-1}, c_{1:t}) \\
&\qquad p(m_{c_t} \mid x_t, x^{[k]}_{1:t-1}, u_{1:t}, z_{1:t-1}, c_{1:t})\; dm_{c_t}\; p(x_t \mid x^{[k]}_{t-1}, u_t) \\
&\overset{\text{Markov}}{=} \quad \eta^{[k]} \int \underbrace{p(z_t \mid m_{c_t}, x_t, c_t)}_{\sim \mathcal{N}(z_t; h(m_{c_t}, x_t), Q_t)}\; \underbrace{p(m_{c_t} \mid x^{[k]}_{1:t-1}, z_{1:t-1}, c_{1:t-1})}_{\sim \mathcal{N}(m_{c_t}; \mu^{[k]}_{c_t,t-1}, \Sigma^{[k]}_{c_t,t-1})}\; dm_{c_t} \\
&\qquad\qquad \underbrace{p(x_t \mid x^{[k]}_{t-1}, u_t)}_{\sim \mathcal{N}(x_t; g(x^{[k]}_{t-1}, u_t), R_t)}
\end{aligned}
$$

This expression makes apparent that our sampling distribution is truly the convolution of two Gaussians multiplied by a third. In the general SLAM case, the sampling distribution possesses no closed form from which we could easily sample. The culprit is the function h: If it were linear, this probability would be Gaussian, a fact that shall become more obvious below. In fact, not even the integral in (13.27) possesses a closed form solution. For this reason, sampling from the probability (13.27) is difficult.

This observation motivates the replacement of h by a linear approximation. As common in this book, this approximation is obtained through a first order Taylor expansion, given by the following linear function:

$$
h(m_{c_t}, x_t) \quad \approx \quad \hat{z}^{[k]}_t + H_m(m_{c_t} - \mu^{[k]}_{c_t,t-1}) + H_x\,(x_t - \hat{x}^{[k]}_t) \tag{13.28}
$$

Here we use the following abbreviations:

$$
\hat{z}^{[k]}_t \quad = \quad h(\mu^{[k]}_{c_t,t-1}, \hat{x}^{[k]}_t) \tag{13.29}
$$

$$
\hat{x}^{[k]}_t \quad = \quad g(x^{[k]}_{t-1}, u_t) \tag{13.30}
$$

The matrices H_m and H_x are the Jacobians of h. They are the derivatives of h with respect to m_{c_t} and x_t, respectively, evaluated at the expected values of their arguments:

$$
H_m \quad = \quad \nabla_{m_{c_t}} h(m_{c_t}, x_t)\big|_{x_t = \hat{x}^{[k]}_t; m_{c_t} = \mu^{[k]}_{c_t,t-1}} \tag{13.31}
$$

$$
H_x \quad = \quad \nabla_{x_t} h(m_{c_t}, x_t)\big|_{x_t = \hat{x}^{[k]}_t; m_{c_t} = \mu^{[k]}_{c_t,t-1}} \tag{13.32}
$$

Under this approximation, the desired sampling distribution (13.27) is a Gaussian with the following parameters:

$$
\Sigma^{[k]}_{x_t} \quad = \quad \left[H_x^T Q_t^{[k]-1} H_x + R_t^{-1} \right]^{-1} \tag{13.33}
$$

$$
\mu^{[k]}_{x_t} \quad = \quad \Sigma^{[k]}_{x_t} H_x^T Q_t^{[k]-1} (z_t - \hat{z}^{[k]}_t) + \hat{x}^{[k]}_t \tag{13.34}
$$

where the matrix $Q_t^{[k]}$ is defined as follows:

$$Q_t^{[k]} = Q_t + H_m \Sigma_{c_t,t-1}^{[k]} H_m^T \tag{13.35}$$

To see, we note that under out linear approximation the convolution theorem provides us with a closed form for the integral term in (13.27):

$$\mathcal{N}(z_t; \hat{z}_t^{[k]} + H_x x_t - H_x \hat{x}_t^{[k]}, Q_t^{[k]}) \tag{13.36}$$

The sampling distribution (13.27) is now given by the product of this normal distribution and the rightmost term in (13.27), the normal $\mathcal{N}(x_t; \hat{x}_t^{[k]}, R_t)$. Written in Gaussian form, we have

$$p(x_t \mid x_{1:t-1}^{[k]}, u_{1:t}, z_{1:t}, c_{1:t}) = \eta \exp\left\{-P_t^{[k]}\right\} \tag{13.37}$$

with

$$\begin{aligned} P_t^{[k]} &= \tfrac{1}{2}\Big[(z_t - \hat{z}_t^{[k]} - H_x x_t + H_x \hat{x}_t^{[k]})^T \; Q_t^{[k]-1} \; (z_t - \hat{z}_t^{[k]} - H_x x_t + H_x \hat{x}_t^{[k]}) \\ &\quad + (x_t - \hat{x}_t^{[k]})^T \; R_t^{-1} \; (x_t - \hat{x}_t^{[k]})\Big] \end{aligned} \tag{13.38}$$

This expression is obviously quadratic in our target variable x_t, hence $p(x_t \mid x_{1:t-1}^{[k]}, u_{1:t}, z_{1:t}, c_{1:t})$ is Gaussian. The mean and covariance of this Gaussian are equivalent to the minimum of $P_t^{[k]}$ and its curvature. Those are identified by calculating the first and second derivatives of $P_t^{[k]}$ with respect to x_t:

$$\begin{aligned} \frac{\partial P_t^{[k]}}{\partial x_t} &= -H_x^T Q_t^{[k]-1}(z_t - \hat{z}_t^{[k]} - H_x x_t + H_x \hat{x}_t^{[k]}) + R_t^{-1}(x_t - \hat{x}_t^{[k]}) \\ &= (H_x^T Q_t^{[k]-1} H_x + R_t^{-1}) x_t - H_x^T Q_t^{[k]-1}(z_t - \hat{z}_t^{[k]} + H_x \hat{x}_t^{[k]}) - R_t^{-1} \hat{x}_t^{[k]} \end{aligned} \tag{13.39}$$

$$\frac{\partial^2 P_t^{[k]}}{\partial x_t^2} = H_x^T Q_t^{[k]-1} H_x + R_t^{-1} \tag{13.40}$$

The covariance $\Sigma_{x_t}^{[k]}$ of the sampling distribution is now obtained by the inverse of the second derivative

$$\Sigma_{x_t}^{[k]} = \left[H_x^T Q_t^{[k]-1} H_x + R_t^{-1}\right]^{-1} \tag{13.41}$$

The mean $\mu_{x_t}^{[k]}$ of the sample distribution is obtained by setting the first derivative (13.39) to zero. This gives us:

$$\begin{aligned} \mu_{x_t}^{[k]} &= \Sigma_{x_t}^{[k]} \left[H_x^T Q_t^{[k]-1}(z_t - \hat{z}_t^{[k]} + H_x \hat{x}_t^{[k]}) + R_t^{-1} \hat{x}_t^{[k]}\right] \\ &= \Sigma_{x_t}^{[k]} H_x^T Q_t^{[k]-1}(z_t - \hat{z}_t^{[k]}) + \Sigma_{x_t}^{[k]} \left[H_x^T Q_t^{[k]-1} H_x + R_t^{-1}\right] \hat{x}_t^{[k]} \\ &= \Sigma_{x_t}^{[k]} H_x^T Q_t^{[k]-1}(z_t - \hat{z}_t^{[k]}) + \hat{x}_t^{[k]} \end{aligned} \tag{13.42}$$

This Gaussian is the approximation of the desired sampling distribution (13.26) in FastSLAM 2.0. Obviously, this proposal distribution is quite a bit more involved than the much simpler one for FastSLAM 1.0 in Equation (13.12).

13.4.2 Updating the Observed Feature Estimate

Just like our first version of the FastSLAM algorithm, FastSLAM 2.0 updates the posterior over the feature estimates based on the measurement z_t and the sampled pose $x_t^{[k]}$. The estimates at time $t-1$ are once again represented by the mean $\mu_{j,t-1}^{[k]}$ and the covariance $\Sigma_{j,t-1}^{[k]}$. The updated estimates are $\mu_{j,t}^{[k]}$ and $\Sigma_{j,t}^{[k]}$. The nature of the update depends on whether or not a feature j was observed at time t. For $j \neq c_t$, we already established in Equation (13.4) that the posterior over the feature remains unchanged. This implies that instead of updating the estimated, we merely have to copy it.

For the observed feature $j = c_t$, the situation is more intricate. Equation (13.5) already stated the posterior over observed features. Here we repeat it with the particle index k:

$$
\begin{aligned}
(13.43)\quad & p(m_{c_t} \mid x_t^{[k]}, c_{1:t}, z_{1:t}) \\
&= \eta \; \underbrace{p(z_t \mid m_{c_t}, x_t^{[k]}, c_t)}_{\sim \mathcal{N}(z_t; h(m_{c_t}, x_t^{[k]}), Q_t)} \; \underbrace{p(m_{c_t} \mid x_{1:t-1}^{[k]}, z_{1:t-1}, c_{1:t-1})}_{\sim \mathcal{N}(m_{c_t}; \mu_{c_t,t-1}^{[k]}, \Sigma_{c_t,t-1}^{[k]})}
\end{aligned}
$$

As in (13.27), the nonlinearity of h causes this posterior to be non-Gaussian, which is at odds with FastSLAM 2.0's Gaussian representation for feature estimates. Luckily, the exact same linearization as above provides the solution:

$$
(13.44)\quad h(m_{c_t}, x_t) \quad \approx \quad \hat{z}_t^{[k]} + H_m(m_{c_t} - \mu_{c_t,t-1}^{[k]})
$$

Notice that x_t is not a free variable here, hence we can omit the third term in (13.28). This approximation renders the probability (13.43) Gaussian in the target variable m_{c_t}:

$$
\begin{aligned}
(13.45)\quad & p(m_{c_t} \mid x_t^{[k]}, c_{1:t}, z_{1:t}) \\
&= \eta \; \exp\Big\{ -\tfrac{1}{2}(z_t - \hat{z}_t^{[k]} - H_m(m_{c_t} - \mu_{c_t,t-1}^{[k]}))\; Q_t^{-1} \\
&\qquad (z_t - \hat{z}_t^{[k]} - H_m(m_{c_t} - \mu_{c_t,t-1}^{[k]})) \\
&\qquad -\tfrac{1}{2}(m_{c_t} - \mu_{c_t,t-1}^{[k]})\; \Sigma_{c_t,t-1}^{[k]-1}(m_{c_t} - \mu_{c_t,t-1}^{[k]}) \Big\}
\end{aligned}
$$

The new mean and covariance are obtained using the standard EKF measurement update equations:

$$K_t^{[k]} = \Sigma_{c_t,t-1}^{[k]} H_m^T Q_t^{[k]-1} \tag{13.46}$$

$$\mu_{c_t,t}^{[k]} = \mu_{c_t,t-1}^{[k]} + K_t^{[k]}(z_t - \hat{z}_t^{[k]}) \tag{13.47}$$

$$\Sigma_{c_t,t}^{[k]} = (I - K_t^{[k]} H_m)\Sigma_{c_t,t-1}^{[k]} \tag{13.48}$$

We notice this is quite a bit more complicated than the update in FastSLAM 1.0, but the additional effort in implementing this often pays out, in terms of improved accuracy.

13.4.3 Calculating Importance Factors

The particles generated thus far do not yet match the desired posterior. In FastSLAM 2.0, the culprit is the normalizer $\eta^{[k]}$ in (13.27), which is usually different for each particle k. These differences are not yet accounted for in the resampling process. As in FastSLAM 1.0, the importance factor is given by the following quotient.

$$w_t^{[k]} = \frac{\text{target distribution}}{\text{proposal distribution}} \tag{13.49}$$

Once again, the target distribution that we would like our particles to assume is given by the path posterior, $p(x_t^{[k]} \mid z_{1:t}, u_{1:t}, c_{1:t})$. Under the asymptotically correct assumptions that paths in $x_{1:t-1}^{[k]}$ have been generated according to the target distribution one time step earlier, $p(x_{1:t-1}^{[k]} \mid z_{1:t-1}, u_{1:t-1}, c_{1:t-1})$, we note that the proposal distribution is now given by the product

$$p(x_{1:t-1}^{[k]} \mid z_{1:t-1}, u_{1:t-1}, c_{1:t-1})\; p(x_t^{[k]} \mid x_{1:t-1}^{[k]}, u_{1:t}, z_{1:t}, c_{1:t}) \tag{13.50}$$

The second term in this product is the pose sampling distribution (13.27). The importance weight is obtained as follows:

$$\begin{aligned} w_t^{[k]} &= \frac{p(x_t^{[k]} \mid u_{1:t}, z_{1:t}, c_{1:t})}{p(x_t^{[k]} \mid x_{1:t-1}^{[k]}, u_{1:t}, z_{1:t}, c_{1:t})\; p(x_{1:t-1}^{[k]} \mid u_{1:t-1}, z_{1:t-1}, c_{1:t-1})} \\ &= \frac{p(x_t^{[k]} \mid x_{1:t-1}^{[k]}, u_{1:t}, z_{1:t}, c_{1:t})\; p(x_{1:t-1}^{[k]} \mid u_{1:t}, z_{1:t}, c_{1:t})}{p(x_t^{[k]} \mid x_{1:t-1}^{[k]}, u_{1:t}, z_{1:t}, c_{1:t})\; p(x_{1:t-1}^{[k]} \mid u_{1:t-1}, z_{1:t-1}, c_{1:t-1})} \\ &= \frac{p(x_{1:t-1}^{[k]} \mid u_{1:t}, z_{1:t}, c_{1:t})}{p(x_{1:t-1}^{[k]} \mid u_{1:t-1}, z_{1:t-1}, c_{1:t-1})} \end{aligned} \tag{13.51}$$

$$
\begin{aligned}
&\overset{\text{Bayes}}{=} && \eta \, \frac{p(z_t \mid x_{1:t-1}^{[k]}, u_{1:t}, z_{1:t-1}, c_{1:t}) \; p(x_{1:t-1}^{[k]} \mid u_{1:t}, z_{1:t-1}, c_{1:t})}{p(x_{1:t-1}^{[k]} \mid u_{1:t-1}, z_{1:t-1}, c_{1:t-1})} \\
&\overset{\text{Markov}}{=} && \eta \, \frac{p(z_t \mid x_{1:t-1}^{[k]}, u_{1:t}, z_{1:t-1}, c_{1:t}) \; p(x_{1:t-1}^{[k]} \mid u_{1:t-1}, z_{1:t-1}, c_{1:t-1})}{p(x_{1:t-1}^{[k]} \mid u_{1:t-1}, z_{1:t-1}, c_{1:t-1})} \\
&= && \eta \, p(z_t \mid x_{1:t-1}^{[k]}, u_{1:t}, z_{1:t-1}, c_{1:t})
\end{aligned}
$$

The reader may notice that this expression is the inverse of our normalization constant $\eta^{[k]}$ in (13.27). Further transformations give us the following form:

$$
\begin{aligned}
(13.52) \quad w_t^{[k]} &= \eta \int p(z_t \mid x_t, x_{1:t-1}^{[k]}, u_{1:t}, z_{1:t-1}, c_{1:t}) \\
&\qquad p(x_t \mid x_{1:t-1}^{[k]}, u_{1:t}, z_{1:t-1}, c_{1:t}) \; dx_t \\
&\overset{\text{Markov}}{=} \eta \int p(z_t \mid x_t, x_{1:t-1}^{[k]}, u_{1:t}, z_{1:t-1}, c_{1:t}) \; p(x_t \mid x_{t-1}^{[k]}, u_t) \; dx_t \\
&= \eta \int \int p(z_t \mid m_{c_t}, x_t, x_{1:t-1}^{[k]}, u_{1:t}, z_{1:t-1}, c_{1:t}) \\
&\qquad p(m_{c_t} \mid x_t, x_{1:t-1}^{[k]}, u_{1:t}, z_{1:t-1}, c_{1:t}) \; dm_{c_t} \; p(x_t \mid x_{t-1}^{[k]}, u_t) \; dx_t \\
&\overset{\text{Markov}}{=} \eta \int \underbrace{p(x_t \mid x_{t-1}^{[k]}, u_t)}_{\sim \mathcal{N}(x_t; g(\hat{x}_{t-1}^{[k]}, u_t), R_t)} \int \underbrace{p(z_t \mid m_{c_t}, x_t, c_t)}_{\sim \mathcal{N}(z_t; h(m_{c_t}, x_t), Q_t)} \\
&\qquad \underbrace{p(m_{c_t} \mid x_{1:t-1}^{[k]}, u_{1:t-1}, z_{1:t-1}, c_{1:t-1})}_{\sim \mathcal{N}(m_{c_t}; \mu_{c_t,t-1}^{[k]}, \Sigma_{c_t,t-1}^{[k]})} \; dm_{c_t} \; dx_t
\end{aligned}
$$

We find that this expression can once again be approximated by a Gaussian over measurements z_t by linearizing h. As it is easily shown, the mean of the resulting Gaussian is $\hat{z}_t$, and its covariance is

$$
(13.53) \quad L_t^{[t]} = H_x^T \; Q_t \; H_x + H_m \Sigma_{c_t,t-1}^{[k]} H_m^T + R_t
$$

Put differently, the (non-normalized) importance factor of the k-th particle is given by the following expression:

$$
(13.54) \quad w_t^{[k]} = |2\pi L_t^{[t]}|^{-\frac{1}{2}} \exp\left\{ -\tfrac{1}{2} (z_t - \hat{z}_t) \; L_t^{[t]-1} \; (z_t - \hat{z}_t) \right\}
$$

As in FastSLAM 1.0, particles generated in Steps 1 and 2, along with their importance factor calculated in Step 3, are collected in a temporary particle set.

The final step of the FastSLAM 2.0 update is a resampling step. Just as in FastSLAM 1.0, FastSLAM 2.0 draws (with replacement) M particles from

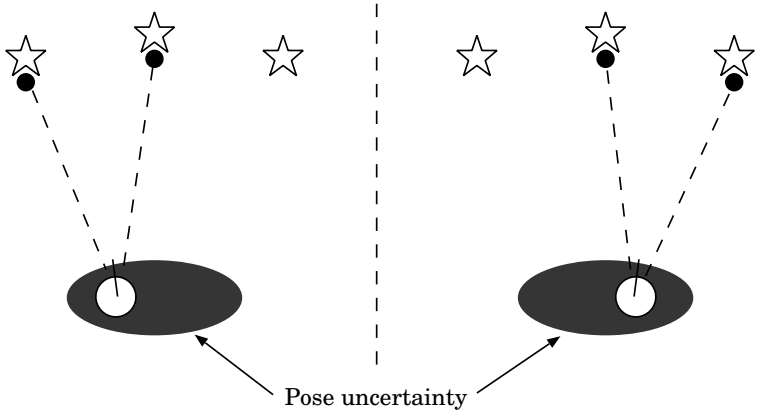


Figure 13.7 The data association problem in SLAM. This figure illustrates that the best data association may vary even within regions of high likelihood for the pose of the robot.

the temporary particle set. Each particle is drawn with a probability proportional to its importance factor $w_t^{[k]}$. The resulting particle set represent (asymptotically) the desired factored posterior at time t.

13.5 Unknown Data Association

This section extends both variants of the FastSLAM algorithm to cases where the correspondence variables $c_{1:t}$ are unknown. A key advantage of using particle filters for SLAM is that each particle can rely on its own, local data association decisions.

We remind the reader that the data association problem at time t is the problem of determining the variable c_t based on the available data. This problem is illustrated in Figure 13.7: Here a robot observes two features in the world. Depending on its actual pose relative to these features, these measurements correspond to different features in the map (depicted as stars in Figure 13.7).

So far, we discussed a number of data association technique using arguments such as maximum likelihood. Those techniques had in common that there is only a single data association per measurement, for the entire filter. FastSLAM, by virtue of using multiple particles, can determine the corre-

spondence on a per-particle basis. Thus, the filter not only samples over robot paths, but also over possible data association decisions along the way.

This is one of the key features of FastSLAM, which sets it aside from the rich body of Gaussian SLAM algorithms. As long as a small subset of the particles is based on the correct data association, data association errors are not as fatal as in EKF approaches. Particles subject to such errors tend to possess inconsistent maps, which increases the probability that they are simply sampled away in future resampling steps.

The mathematical definition of the per-particle data association is straightforward, in that it generalizes the per-filter data association to individual particles. Each particle maintains a local set of data association variables, denoted $\hat{c}_t^{[k]}$. In maximum likelihood data association, each $\hat{c}_t^{[k]}$ is determined by maximizing the likelihood of the measurement z_t:

$$\hat{c}_t^{[k]} = \operatorname*{argmax}_{c_t} \; p(z_t \mid c_t, \hat{c}_{1:t-1}^{[k]}, x_{1:t}^{[k]}, z_{1:t-1}, u_{1:t}) \tag{13.55}$$

An alternative is the data association sampler (DAS), which samples the data association variable according to their likelihood

$$\hat{c}_t \sim \eta \, p(z_t \mid c_t, \hat{c}_{1:t-1}, x_{1:t}^{[k]}, z_{1:t-1}, u_{1:t}) \tag{13.56}$$

Both techniques, ML and DAS, make it possible to estimate the number of features in the map. SLAM techniques using ML create new features in the map if the likelihood falls below a threshold p_0 for all known features in the map. DAS associates an observed measurement with a new, previously unobserved feature stochastically. They do so with probability proportional to ηp_0, where η is a normalizer defined in (13.56).

$$\hat{c}_t^{[k]} \sim \eta \, p(z_t \mid c_t, \hat{c}_{1:t-1}^{[k]}, x_{1:t}^{[k]}, z_{1:t-1}, u_{1:t}) \tag{13.57}$$

For both techniques, the likelihood is calculated as follows:

$$\begin{aligned} & p(z_t \mid c_t, \hat{c}_{1:t-1}^{[k]}, x_{1:t}^{[k]}, z_{1:t-1}, u_{1:t}) \\ &= \int p(z_t \mid m_{c_t}, c_t, \hat{c}_{1:t-1}^{[k]}, x_{1:t}^{[k]}, z_{1:t-1}, u_{1:t}) \\ &\qquad\qquad p(m_{c_t} \mid c_t, \hat{c}_{1:t-1}^{[k]}, x_{1:t}^{[k]}, z_{1:t-1}, u_{1:t}) \, dm_{c_t} \\ &= \int \underbrace{p(z_t \mid m_{c_t}, c_t, x_t^{[k]})}_{\sim \mathcal{N}(z_t; h(m_{c_t}, x_t^{[k]}), Q_t)} \; \underbrace{p(m_{c_t} \mid \hat{c}_{1:t-1}^{[k]}, x_{1:t-1}^{[k]}, z_{1:t-1})}_{\sim \mathcal{N}(\mu_{c_t,t-1}^{[k]}, \Sigma_{c_t,t-1}^{[k]})} \, dm_{c_t} \end{aligned} \tag{13.58}$$

Linearization of h enables us to obtain this in closed form:

$$p(z_t \mid c_t, \hat{c}_{1:t-1}^{[k]}, x_t^{[k]}, z_{1:t-1}, u_{1:t}) \tag{13.59}$$

$$= \quad |2\pi Q_t^{[k]}|^{-\frac{1}{2}} \exp\left\{-\tfrac{1}{2}(z_t - h(\mu_{c_t,t-1}^{[k]}, x_t^{[k]}))^T Q_t^{[k]-1}(z_t - h(\mu_{c_t,t-1}^{[k]}, x_t^{[k]}))\right\}$$

The variable $Q_t^{[k]}$ was defined in Equation (13.35), as a function of the data association variable c_t. New features are added to the map in exactly the same way as outlined above. In the ML approach, a new feature is added when the probability $p(z_t \mid c_t, \hat{c}_{1:t-1}^{[k]}, x_t^{[k]}, z_{1:t-1}, u_{1:t})$ falls beyond a threshold p_0. The DAS includes the hypothesis that an observation corresponds to a previously unobserved feature in its set of hypotheses, and samples it with probability ηp_0.

13.6 Map Management

Map management in FastSLAM is largely equivalent to EKF SLAM, with a few particulars arising from the fact that data association is handled on a per-particle level in FastSLAM.

As in the alternative SLAM algorithms, any newly added feature requires the initialization of a new Kalman filter. In many SLAM problems the measurement function h is *invertible*. This is the case, for example, for robots measuring range and bearing to features in the plane, in which a single measurement suffices to produce a (non-degenerate) estimate on the feature location. The initialization of the EKF is then straightforward:

$$x_t^{[k]} \sim p(x_t \mid x_{t-1}^{[k]}, u_t) \tag{13.60}$$

$$\mu_{n,t}^{[k]} = h^{-1}(z_t, x_t^{[k]}) \tag{13.61}$$

$$\Sigma_{n,t}^{[k]} = (H_{\hat{c}}^{[k]T} Q_t^{-1} H_{\hat{c}}^{[k]})^{-1} \quad \text{with} \quad H_{\hat{c}}^{[k]} = h'(\mu_{n,t}^{[k]}, x_t^{[k]}) \tag{13.62}$$

$$w_t^{[k]} = p_0 \tag{13.63}$$

Notice that for newly observed features, the pose $x_t^{[k]}$ is sampled according to the motion model $p(x_t \mid x_{t-1}^{[k]}, u_t)$. This distribution is equivalent to the FastSLAM sampling distribution (13.26) in situations where no previous location estimate for the observed feature is available.

Initialization techniques for situations in which h is not invertible are discussed in Deans and Hebert (2002). In general, such situations require the accumulation of multiple measurements, to obtain a good estimate for the linearization of h.

To accommodate features introduced erroneously into the map, FastSLAM features a mechanism for eliminating features that are not supported by suf-

ficient evidence. Just as in EKF SLAM, FastSLAM does so by keeping track of the log odds of the actual existence of individual features in the map.

Specifically, when a feature is observed, its log odds for existence is incremented by a fixed amount, which is calculated using the standard Bayes filter formula. Similarly, when a feature is not observed even though it should have, such negative information results in a decrement of the feature existence variable by a fixed amount. Features whose variable sinks below a threshold value are then simply removed from the list of particles. It is also possible to implement a provisional feature list in FastSLAM. Technically this is trivial, since each feature possesses its own particle.

13.7 The FastSLAM Algorithms

Tables 13.2 and 13.3 summarize both FastSLAM variants with unknown data association. In both algorithms, particles are of the form

$$Y_t^{[k]} = \left\langle x_t^{[k]}, N_t^{[k]}, \left\langle \mu_{1,t}^{[k]}, \Sigma_{1,t}^{[k]}, \tau_1^{[k]} \right\rangle, \ldots, \left\langle \mu_{N_t^{[k]},t}^{[k]}, \Sigma_{N_t^{[k]},t}^{[k]}, \tau_{N_t^{[k]}}^{[k]} \right\rangle \right\rangle \tag{13.64}$$

In addition to the pose $x_t^{[k]}$ and the feature estimates $\mu_{n,t}^{[k]}$ and $\Sigma_{n,t}^{[k]}$, each particle maintains the number of features $N_t^{[k]}$ in its local map, and each feature carries a probabilistic estimate of its existence $\tau_n^{[k]}$. Iterating the filter requires time linear in the maximum number of features $\max_k N_t^{[k]}$ in each map, and it is also linear in the number of particles M. Further below, we will discuss advanced data structure that yield more efficient implementations.

We note that both versions of FastSLAM, as described here, consider a single measurement at a time. As before, this choice is made for notational convenience, and many of the techniques discussed in previous SLAM chapters can be brought to bear.

13.8 Efficient Implementation

At first glance, it may appear that each update in FastSLAM requires $O(MN)$ time, where M is the number of particles M and N the number of features in the map. The linear complexity in M is unavoidable, given that we have to process M particles with each update. The linear complexity in N is the result of the resampling process. Whenever a particle is drawn more than once in the resampling process, a "naive" implementation might duplicate the entire map attached to the particle. Such a duplication process is linear in the

Algorithm FastSLAM 1.0(z_t, u_t, Y_{t-1}):

for $k = 1$ to M do // loop over all particles

retrieve $\left\langle x_{t-1}^{[k]}, N_{t-1}^{[k]}, \left\langle \mu_{1,t-1}^{[k]}, \Sigma_{1,t-1}^{[k]}, i_1^{[k]} \right\rangle, \ldots, \left\langle \mu_{N_{t-1}^{[k]},t-1}^{[k]}, \Sigma_{N_{t-1}^{[k]},t-1}^{[k]}, i_{N_{t-1}^{[k]},t-1}^{[k]} \right\rangle \right\rangle$ from Y_{t-1}

$x_t^{[k]} \sim p(x_t \mid x_{t-1}^{[k]}, u_t)$ // sample new pose

for $j = 1$ to $N_{t-1}^{[k]}$ do // measurement likelihoods

$\hat{z}_j = h(\mu_{j,t-1}^{[k]}, x_t^{[k]})$ // measurement prediction

$H_j = h'(\mu_{j,t-1}^{[k]}, x_t^{[k]})$ // calculate Jacobian

$Q_j = H_j \; \Sigma_{j,t-1}^{[k]} \; H_j^T + Q_t$ // measurement covariance

$w_j = |2\pi Q_j|^{-\frac{1}{2}} \; \exp\left\{-\frac{1}{2}(z_t - \hat{z}_j)^T Q_j^{-1}(z_t - \hat{z}_j)\right\}$ // likelihood of correspondence

endfor

$w_{1+N_{t-1}^{[k]}} = p_0$ // importance factor, new feature

$w^{[k]} = \max w_j$ // max likelihood correspondence

$\hat{c} = \operatorname{argmax} w_j$ // index of ML feature

$N_t^{[k]} = \max\{N_{t-1}^{[k]}, \hat{c}\}$ // new number of features in map

for $j = 1$ to $N_t^{[k]}$ do // update Kalman filters

if $j = \hat{c} = 1 + N_{t-1}^{[k]}$ then // is new feature?

$\mu_{j,t}^{[k]} = h^{-1}(z_t, x_t^{[k]})$ // initialize mean

$H_j = h'(\mu_{j,t}^{[k]}, x_t^{[k]}); \Sigma_{j,t}^{[k]} = (H_j^{-1})^T Q_t H_j^{-1}$ // initialize covar.

$i_{j,t}^{[k]} = 1$ // initialize counter

else if $j = \hat{c} \leq N_{t-1}^{[k]}$ then // is observed feature?

$K = \Sigma_{j,t-1}^{[k]} H_j^T Q_{\hat{c}}^{-1}$ // calculate Kalman gain

$\mu_{j,t}^{[k]} = \mu_{j,t-1}^{[k]} + K(z_t - \hat{z}_{\hat{c}})$ // update mean

$\Sigma_{j,t}^{[k]} = (I - K \; H_j)\Sigma_{j,t-1}^{[k]}$ // update covariance

$i_{j,t}^{[k]} = i_{j,t-1}^{[k]} + 1$ // increment counter

see next page for continuation

continued from the previous page

```
        else                                        // all other features
          μ_{j,t}^[k] = μ_{j,t-1}^[k]               // copy old mean
          Σ_{j,t}^[k] = Σ_{j,t-1}^[k]               // copy old covariance
          if μ_{j,t-1}^[k] outside perceptual
                 range of x_t^[k] then              // should feature have been seen?
            i_{j,t}^[k] = i_{j,t-1}^[k]             // no, do not change
          else
            i_{j,t}^[k] = i_{j,t-1}^[k] - 1         // yes, decrement counter
            if i_{j,t-1}^[k] < 0 then
              discard feature j                     // discard dubious features
            endif
          endif
        endif
      endfor
      add ⟨x_t^[k], N_t^[k], ⟨μ_{1,t}^[k], Σ_{1,t}^[k], i_1^[k]⟩, ..., ⟨μ_{N_t^[k],t}^[k], Σ_{N_t^[k],t}^[k], i_{N_t^[k]}^[k]⟩⟩ to Y_aux
    endfor
    Y_t = ∅                                         // construct new particle set
    do M times                                      // resample M particles
      draw random index k
            with probability ∝ w^[k]                // resample
      add ⟨x_t^[k], N_t^[k], ⟨μ_{1,t}^[k], Σ_{1,t}^[k], i_1^[k]⟩, ..., ⟨μ_{N_t^[k],t}^[k], Σ_{N_t^[k],t}^[k], i_{N_t^[k]}^[k]⟩⟩ to Y_t
    enddo
    return Y_t
```

Table 13.2 The algorithm FastSLAM 1.0 with unknown data association. This version does not implement any of the efficient tree representations discussed in the chapter.

Algorithm FastSLAM 2.0(z_t, u_t, Y_{t-1}):

for $k = 1$ *to* M *do* // *loop over all particles*

  retrieve $\left\langle x_{t-1}^{[k]}, N_{t-1}^{[k]}, \left\langle \mu_{1,t-1}^{[k]}, \Sigma_{1,t-1}^{[k]}, i_1^{[k]} \right\rangle, \ldots, \left\langle \mu_{N_{t-1}^{[k]},t-1}^{[k]}, \Sigma_{N_{t-1}^{[k]},t-1}^{[k]}, i_{N_{t-1}^{[k]}}^{[k]} \right\rangle \right\rangle$ *from* Y_{t-1}

  for $j = 1$ *to* $N_{t-1}^{[k]}$ *do* // *calculate sampling distribution*

    $\hat{x}_{j,t} = g(x_{t-1}^{[k]}, u_t)$ // *predict pose*

    $\bar{z}_j = h(\mu_{j,t-1}^{[k]}, \hat{x}_{j,t})$ // *predict measurement*

    $H_{x,j} = \nabla_{x_t}\ h(\mu_{j,t-1}^{[k]}, \hat{x}_{j,t})$ // *Jacobian wrt pose*

    $H_{m,j} = \nabla_{m_j}\ h(\mu_{j,t-1}^{[k]}, \hat{x}_{j,t})$ // *Jacobian wrt map feature*

    $Q_j = Q_t + H_{m,j}\ \Sigma_{j,t-1}^{[k]}\ H_{m,j}^T$ // *measurement information*

    $\Sigma_{x,j} = \left[H_{x,j}^T\ Q_j^{-1}\ H_{x,j} + R_t^{-1}\right]^{-1}$ // *Cov of proposal distribution*

    $\mu_{x_t,j} = \Sigma_{x,j}\ H_{x,j}^T\ Q_j^{-1}\ (z_t - \bar{z}_j) + \hat{x}_{j,t}$ // *mean of proposal distribution*

    $x_{t,j}^{[k]} \sim \mathcal{N}(\mu_{x_t,j}, \Sigma_{x,j})$ // *sample pose*

    $\hat{z}_j = h(\mu_{j,t-1}^{[k]}, x_t^{[k]})$ // *measurement prediction*

    $\pi_j = |2\pi Q_j|^{-\frac{1}{2}}\ \exp\left\{-\frac{1}{2}\ (z_t - \hat{z}_j)^T\ Q_j^{-1}\ (z_t - \hat{z}_j)\right\}$ // *correspondence likelihood*

  endfor

  $\pi_{1+N_{t-1}^{[k]}} = p_0$ // *likelihood of new feature*

  $\hat{c} = \operatorname{argmax} \pi_j$ // *ML correspondence*

  $N_t^{[k]} = \max\{N_{t-1}^{[k]}, \hat{c}\}$ // *new number of features*

  for $j = 1$ *to* $N_t^{[k]}$ *do* // *update Kalman filters*

    if $j = \hat{c} = 1 + N_{t-1}^{[k]}$ *then* // *is new feature?*

      $x_t^{[k]} \sim p(x_t \mid x_{t-1}^{[k]}, u_t)$ // *sample pose*

      $\mu_{j,t}^{[k]} = h^{-1}(z_t, x_t^{[k]})$ // *initialize mean*

      $H_{m,j} = \nabla_{m_j}\ h(\mu_{j,t}^{[k]}, x_t^{[k]})$ // *Jacobian wrt map feature*

      $\Sigma_{j,t}^{[k]} = (H_{m,j}^{-1})^T Q_t\ H_{m,j}^{-1}$ // *initialize covariance*

      $i_{j,t}^{[k]} = 1$ // *initialize counter*

      $w^{[k]} = p_0$ // *importance weight*

    else if $j = \hat{c} \leq N_{t-1}^{[k]}$ *then* // *is observed feature?*

      $x_t^{[k]} = x_{t,j}^{[k]}$

      $K = \Sigma_{j,t-1}^{[k]}\ H_{m,j}^T\ Q_j^{-1}$ // *calculate Kalman gain*

see next page for continuation

continued from the previous page

```
            μ_{j,t}^[k] = μ_{j,t-1}^[k] + K(z_t - ẑ_j)          // update mean
            Σ_{j,t}^[k] = (I - K H_{m,j}) Σ_{j,t-1}^[k]         // update covariance
            i_{j,t}^[k] = i_{j,t-1}^[k] + 1                     // increment counter
            L = H_{x,j} R_t H_{x,j}^T + H_{m,j} Σ_{j,t-1}^[k] H_{m,j}^T + Q_t
            w^[k] = |2πL|^(-1/2) exp{-1/2
                (z_t - ẑ_j)^T L^(-1) (z_t - ẑ_j)}               // importance weight
        else                                                    // all other features
            μ_{j,t}^[k] = μ_{j,t-1}^[k]                         // copy old mean
            Σ_{j,t}^[k] = Σ_{j,t-1}^[k]                         // copy old covariance
            if μ_{j,t-1}^[k] outside perceptual
                    range of x_t^[k] then                       // should feature have been seen?
                i_{j,t}^[k] = i_{j,t-1}^[k]                     // no, do not change
            else
                i_{j,t}^[k] = i_{j,t-1}^[k] - 1                 // yes, decrement counter
                if i_{j,t-1}^[k] < 0 then
                    discard feature j                           // discard dubious features
                endif
            endif
        endif
    endfor
    add ⟨x_t^[k], N_t^[k], ⟨μ_{1,t}^[k], Σ_{1,t}^[k], i_1^[k]⟩, ..., ⟨μ_{N_t^[k],t}^[k], Σ_{N_t^[k],t}^[k], i_{N_t^[k]}^[k]⟩⟩ to Y_aux
endfor
Y_t = ∅                                                         // construct new particle set
do M times                                                      // resample M particles
    draw random index k
            with probability ∝ w^[k]                            // resample
    add ⟨x_t^[k], N_t^[k], ⟨μ_{1,t}^[k], Σ_{1,t}^[k], i_1^[k]⟩, ..., ⟨μ_{N_t^[k],t}^[k], Σ_{N_t^[k],t}^[k], i_{N_t^[k]}^[k]⟩⟩ to Y_t
enddo
return Y_t
```

Table 13.3 The FastSLAM 2.0 Algorithm, stated here unknown data association.

size of the map N. Furthermore, a naive implementation of data association may result in evaluating the measurement likelihood for each of the N features in the map, resulting again in linear complexity in N. We note that a poor implementation of the sampling process might easily add another $\log N$ to the update complexity.

Efficient implementations of FastSLAM require only $O(M \log N)$ update time. This is logarithmic in the size of the map N. First, consider the situation with known data association. Linear copying costs can be avoided by introducing a data structure for representing particles that allow for more selective updates. The basic idea is to organize the map as a *balanced binary tree*. Figure 13.8a shows such a tree for a single particle, in the case of $M = 8$ features. Notice that all Gaussian parameters $\mu_j^{[k]}$ and $\Sigma_j^{[k]}$ for all j are located at the leaves of the tree. Assuming that the tree is approximately balanced, accessing a leaf requires time logarithmic in N.

Suppose FastSLAM incorporates a new control u_t and a new measurement z_t. Each new particle in Y_t will differ from the corresponding one in Y_{t-1} in two ways: First, it will possess a different pose estimate obtained via (13.26), and second, the observed feature's Gaussian will have been updated, as specified in Equations (13.47) and (13.48). All other Gaussian feature estimates, however, will be equivalent to the generating particle. When copying the particle, thus, only a single path has to be modified in the tree representing all Gaussians, leading to the logarithmic update time.

An illustration of this "trick" is shown in Figure 13.8b: Here we assume $c_t^i = 3$, hence only the Gaussian parameters $\mu_3^{[k]}$ and $\Sigma_3^{[k]}$ are updated. Instead of generating an entire new tree, only a single path is created, leading to the Gaussian $c_t^i = 3$. This path is an incomplete tree. The tree is completed by copying the missing pointers from the tree of the generating particle. Thus, branches that leave the path will point to the same (unmodified) subtree as that of the generating tree. Clearly, generating this tree takes only time logarithmic in N. Moreover, accessing a Gaussian also takes time logarithmic in N, since the number of steps required to navigate to a leaf of the tree is equivalent to the length of the path (which is by definition logarithmic). Thus, both generating and accessing a partial tree can be done in time $O(\log N)$. Since in each updating step M new particles are created, an entire update requires time in $O(M \log N)$.

Organizing particles in trees raises the question as to when to deallocate memory. Memory deallocation can equally be implemented in amortized logarithmic time. The idea is to assign a variable to each node—internal

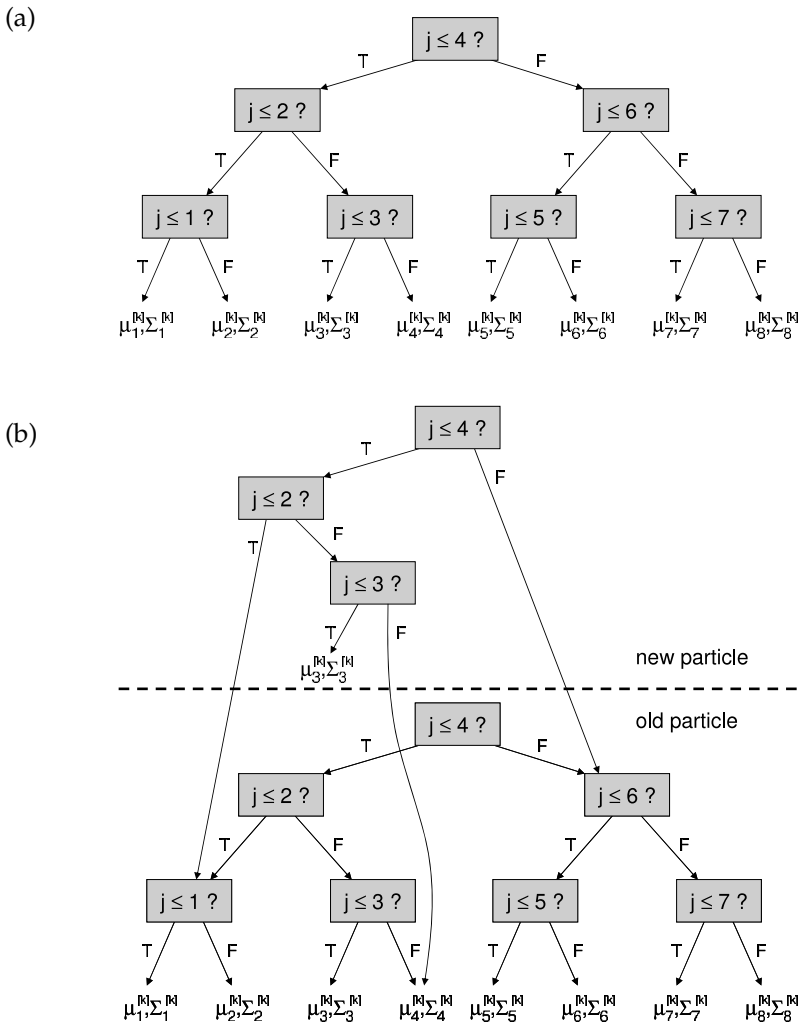


Figure 13.8 (a) A tree representing $N = 8$ feature estimates within a single particle. (b) Generating a new particle from an old one, while modifying only a single Gaussian. The new particle receives only a partial tree, consisting of a path to the modified Gaussian. All other pointers are copied from the generating tree. This can be done in time logarithmic in N.

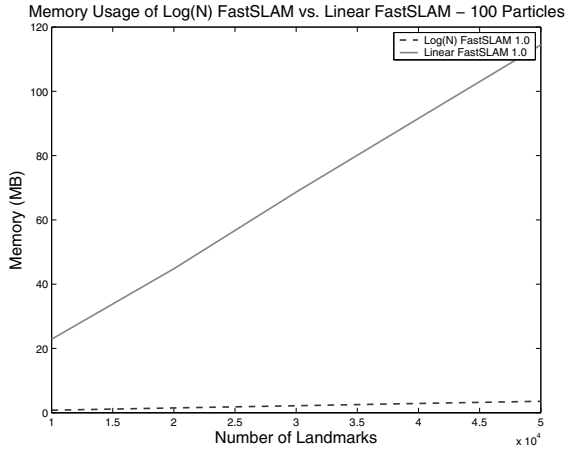


Figure 13.9 Memory requirements for linear and $\log(N)$ version of FastSLAM 1.0.

or leaf—that counts the number of pointers pointing to it. The counter of a newly created node will be initialized by 1. It will be incremented as pointers to a node are created in other particles. Decrements occur when pointers are removed (e.g., pointers of pose particles that fail to survive the resampling process). When a counter reaches zero, its children's counters are decremented and the memory of the corresponding node is deallocated. The process is then applied recursively to all children of the node whose counter may have reached zero. This recursive process will require $O(M \log N)$ time on average.

The tree also induces substantial memory savings. Figure 13.9 shows the effect of the efficient tree technique on the memory consumed by FastSLAM, measured empirically. This graph is the result of an actual implementation of FastSLAM 1.0 with $M = 100$ particles for acquiring feature-based maps. The graph shows nearly two orders of magnitude savings for a map with 50,000 features. The relative savings in update time are similar in value.

Obtaining logarithmic time complexity for FastSLAM with unknown data association is more difficult. Specifically, we need a technique that restricts data association search to the local neighborhood of a feature, to avoid calculating the data association probability for all N features in the map. Further, the tree has to remain approximately balanced.

There indeed exists variants of kernel density trees, or kd-trees, that can

meet these assumptions, assuming that the variance in the sensor measurements is small compared to the overall size of the map. For example, the *bkd-tree* proposed by Procopiuc et al. (2003) maintains a sequence of trees of growing complexity. By carefully shifting items across those trees, a logarithmic time recall can be guaranteed under amortized logarithmic time for inserting new features in the map. Another is the DP-SLAM algorithm proposed by Eliazar and Parr (2003), which uses *history trees* for efficient storage and retrieval, similar to the one described here.

13.9 FastSLAM for Feature-Based Maps

13.9.1 Empirical Insights

The FastSLAM algorithm has been applied to a number of map representation and sensor data. The most basic application concerns feature-based maps, assuming that the robot is equipped with a sensor for detecting range and bearing to landmarks. One such data set is the Victoria Park dataset, which we already discussed in Chapter 12. Figure 13.10a shows the path of the vehicle obtained by integrating the estimated controls. Controls are a poor predictor of location for this vehicle; after 30 minutes of driving, the estimated position of the vehicle is well over 100 meters away from its GPS position.

The remaining three panels of Figure 13.10 show the output of FastSLAM 1.0. In all these diagrams, the path estimated by GPS is shown as a dashed line, and the output of FastSLAM is shown as a solid line. The RMS error of the resulting path is just over 4 meters over the 4 km traverse. This experiment was run with $M = 100$ particles. This error is indistinguishable from the error of other state-of-the-art SLAM algorithms, such as the ones discussed in previous chapters. The robustness of FastSLAM becomes apparent in Figure 13.10d, which plots the result for an experiment in which we simply ignored the motion information. Instead, the odometry-based motion model was replaced by a Brownian motion model. The average error of FastSLAM is statistically indistinguishable from the error obtained before.

When implementing FastSLAM in feature-based maps, it is important to consider negative information. When negative information is used to estimate the existence of each feature, as described in Chapter 13.6, many spurious features can be removed from the map. Figure 13.11 shows the Victoria Park map built with and without considering negative evidence. Here the use of negative information results in 44% percent fewer features in the re-

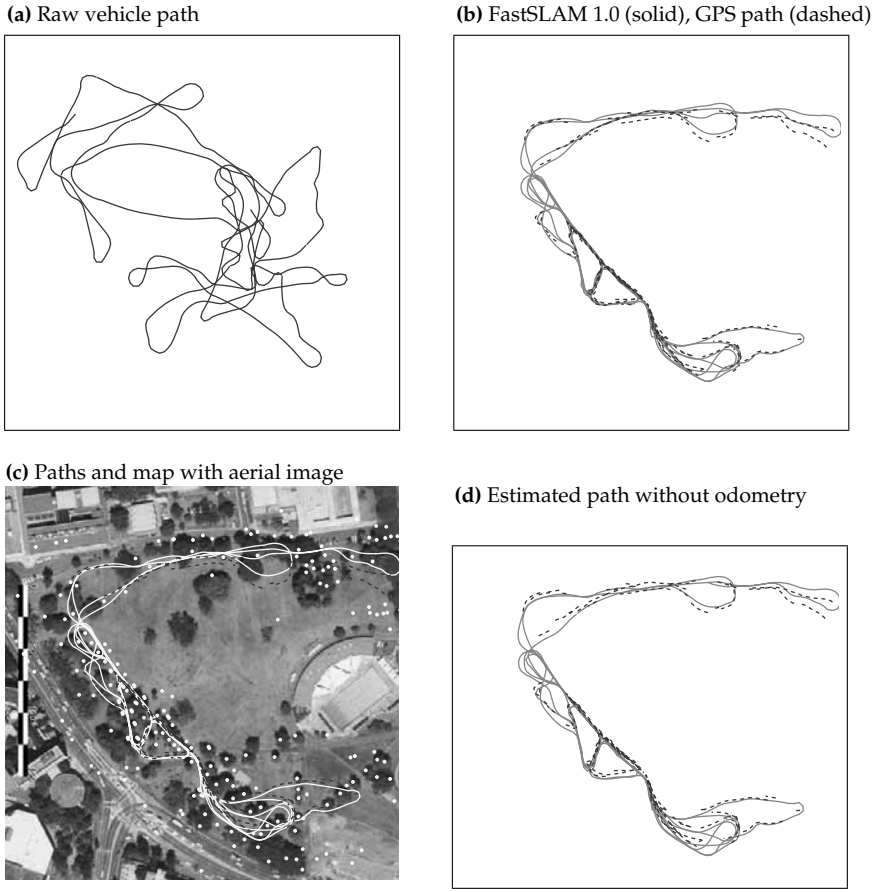


Figure 13.10 (a) Vehicle path predicted by the odometry; (b) True path (dashed line) and FastSLAM 1.0 path (solid line); (c) Victoria Park results overlayed on aerial imagery with the GPS path in blue (dashed), average FastSLAM 1.0 path in yellow (solid), and estimated features as yellow circles. (d) Victoria Park Map created without odometry information. Data and aerial image courtesy of José Guivant and Eduardo Nebot, Australian Centre for Field Robotics.

sulting map. While the *correct* number of features is not available, visual inspection of the maps suggests that many of the spurious features have been eliminated.

It makes sense to compare FastSLAM to EKF SLAM, which continues to be

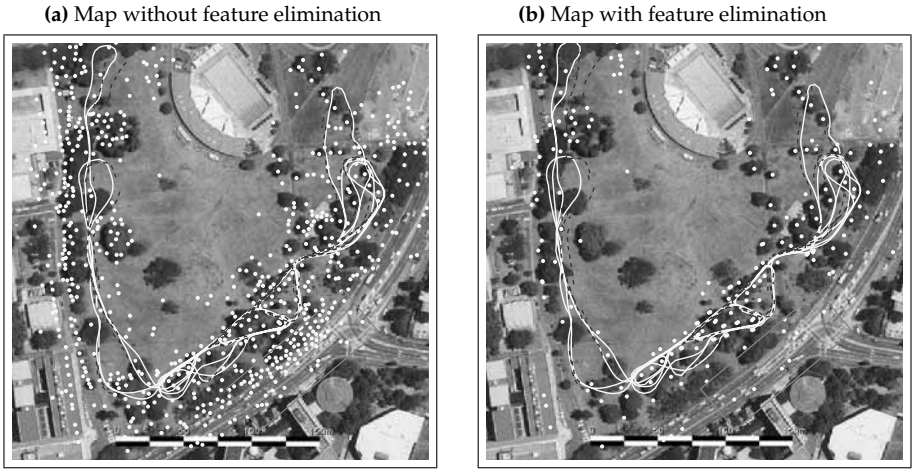


Figure 13.11 FastSLAM 1.0 (a) without and (b) with feature elimination based on negative information.

a popular benchmark algorithm. For example, Figure 13.12 compares the accuracy of FastSLAM 1.0 with that of the EKF, for various numbers of particles from 1 to 5,000. The error of the EKF SLAM is shown as a dashed horizontal line in Figure 13.12 for comparison. The accuracy of FastSLAM 1.0 approaches the accuracy of the EKF as the number of particles is increased, and it is statistically indistinguishable from that of the EKF past approximately 10 particles. This is interesting because FastSLAM 1.0 with 10 particles and 100 features requires an order of magnitude fewer parameters than EKF SLAM in order to achieve this level of accuracy.

In practice, FastSLAM 2.0 yields superior results to FastSLAM 1.0, though the improvement is of significance only under certain circumstances. As a rule of thumb, both algorithms produce comparable results when the number of particles M is large, and when the measurement noise is large compared to the motion uncertainty. This is illustrated in Figure 13.13, which graphs the accuracy of either FastSLAM variant as a function of the measurement noise, using $M = 100$ particles. The most important finding here is FastSLAM 1.0's relatively poor performance in low-noise simulations. One way to test whether a FastSLAM 1.0 implementation suffers from this pathology is to artificially inflate the measurement noise in the probabilistic model $p(z \mid x)$. If, as a result of this inflation, the overall map error goes down—not

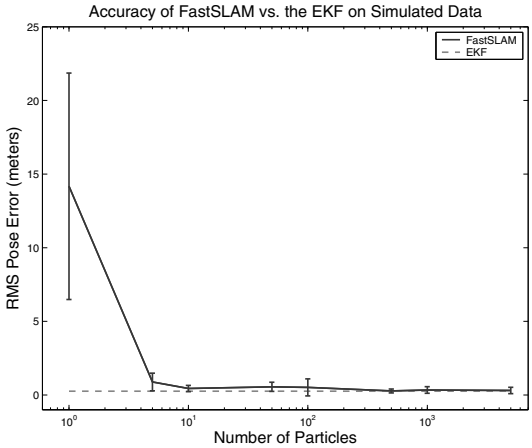


Figure 13.12 A comparison of the accuracy of FastSLAM 1.0 and the EKF on simulated data.

up—it is time to switch to FastSLAM 2.0.

13.9.2 Loop Closure

No algorithm is perfect. There exists problems in which FastSLAM is inferior to its Gaussian counterparts. One such problem involves *loop closure*. In loop closure, a robot moves through unknown terrain and at some point encounters features not seen for a long time. It is here that maintaining the correlations in a SLAM algorithm is particularly important, so that the information acquired when closing a loop can be propagated through the entire map. EKFs and GraphSLAM maintain such correlations directly, whereas FastSLAM maintains them through its diversity in the particle sets. Thus, the ability to close loops, depends on the number of particles M. Better diversity in the sample set results in better loop closing performance, because new observations can affect the pose of the vehicle further back in the past.

Unfortunately, by pruning away improbable trajectories of the vehicle, resampling eventually causes all of the FastSLAM particles to share a common history at some point in the past. New observations cannot affect the positions of features observed prior to this point. This common history point can be pushed back in time by increasing the number of particles M. This process of throwing away correlation data over time enables FastSLAM's effi-

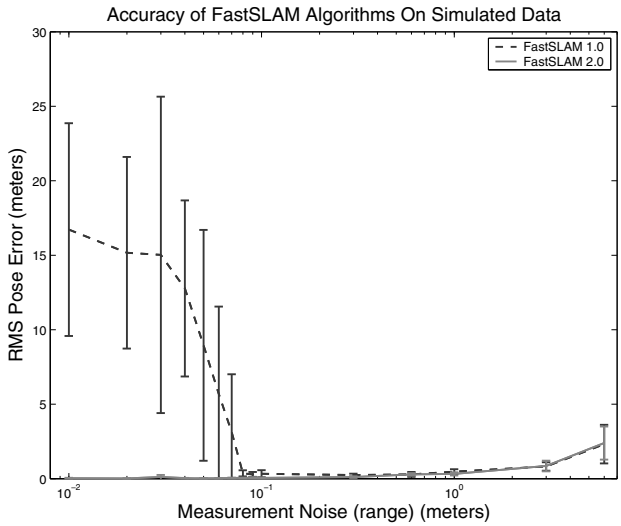


Figure 13.13 FastSLAM 1.0 and 2.0 with varying levels of measurement noise: As to be expected, FastSLAM 2.0 is uniformly superior to FastSLAM 1.0. The difference is particularly obvious for small particle sets, where the improved proposal distribution focuses the particles much better.

cient sensor updates. This efficiency comes at the cost of slower convergence speed. Throwing away correlation information means that more observations will be required to achieve a given level of accuracy. Clearly, FastSLAM 2.0's improved proposal distribution ensures that fewer particles are eliminated in resampling compared to FastSLAM 1.0, but it does not alleviate this problem.

In practice, diversity is important, and it is worthwhile to optimize the implementation so as to maintain maximum diversity. Examples of loop closure are shown in Figure 13.15. These figures show the histories of all M particles. In Figure 13.15a, the FastSLAM 1.0 particles share a common history part of the way around the loop. New observations can not affect the positions of features observed before this threshold. In this case of FastSLAM 2.0, the algorithm is able to maintain diversity that extends back to the beginning of the loop. This is crucial for reliable loop closing and fast convergence.

Figure 13.16a shows the result of an experiment comparing the loop closing performance of FastSLAM 1.0 and 2.0. As the size of the loop increases, the error of both algorithms increases. However, FastSLAM 2.0 consistently

Figure 13.14 Map of Victoria Park by FastSLAM 2.0 with $M = 1$ particle.

outperforms FastSLAM 1.0. Alternately, this result can be rephrased in terms of particles. FastSLAM 2.0 requires fewer particles to close a given loop than FastSLAM 1.0.

Figure 13.16b shows the results of an experiment comparing the convergence speed of FastSLAM 2.0 and the EKF. FastSLAM 2.0 (with 1, 10, and 100 particles) and the EKF were each run 10 times around a large simulated loop of features, similar to the ones shown in Figure 13.16a&b. Different random seeds were used for each run, causing different controls and observations to be generated for each loop. The RMS position error in the map at every time step was averaged over the 10 runs for each algorithm.

As the vehicle goes around the loop, error should gradually build up in the map. When the vehicle closes the loop at iteration 150, revisiting old features should affect the positions of features all around the loop, causing the overall error in the map to decrease. This clearly happens in the EKF. FastSLAM 2.0 with a single particle has no way to affect the positions of past features so there is no drop in the feature error. As more particles are added to FastSLAM 2.0, the filter is able to apply observations to feature positions further back in time, gradually approaching the convergence speed of the EKF. Clearly, the number of particles necessary to achieve convergence time close to the EKF will increase with the size of the loop. The lack of long-range correlations in the FastSLAM representation is arguably the most important

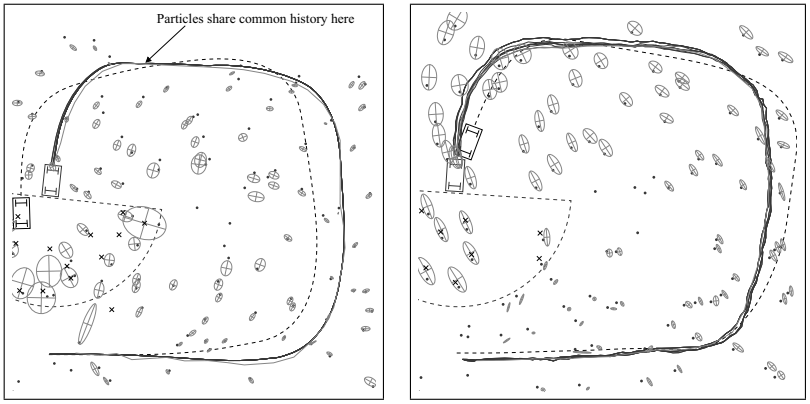


Figure 13.15 FastSLAM 2.0 can close larger loops than FastSLAM 1.0 given a constant number of particles.

weakness of FastSLAM algorithm over Gaussian SLAM techniques.

13.10 Grid-based FastSLAM

13.10.1 The Algorithm

In Chapter 9 we studied *occupancy grid maps* as a volumetric representation of robot environments. The advantage of such a representation is that it does not require any predefined definition of landmarks. Instead, in can model arbitrary types of environments. In the remainder of this chapter, we will therefore extend the FastSLAM algorithm to such representations.

To adapt the FastSLAM algorithm to occupancy grid maps, we need three functions that we already defined in previous sections. First, we have to sample from the motion posterior $p(x_t \mid x_{t-1}^{[k]}, u_t)$ as in Equation 13.12. Hence, we need such a sampling technique. Second, we need a technique for estimating the map of each particle. It turns out that we can rely on occupancy grid mapping, as described in Chapter 9. Finally, we need to compute the importance weights of the individual particles. That is, we require an approach to compute the likelihood $p(z_t \mid x_t^k, m^{[k]})$ of the observation z_t, conditioned on the pose x_t^k, the map $m^{[k]}$, and the most recent measurement z_t.

As it turns out, the extension of FastSLAM to occupancy grid maps is quite straightforward. Table 13.4 describes FastSLAM with occupancy grid maps. Not surprisingly, this algorithm borrows parts of Monte Carlo Localization

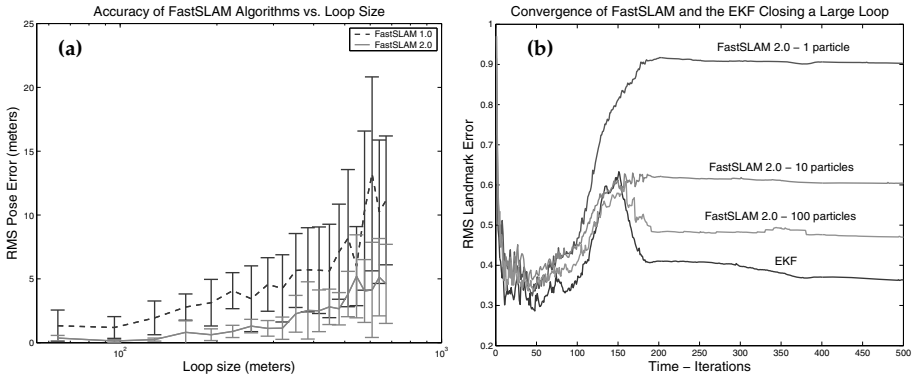


Figure 13.16 (a) Accuracy as a function of loop size: FastSLAM 2.0 can close larger loops than FastSLAM 1.0 given a fixed number of particles. (b) Comparison of the convergence speed of FastSLAM 2.0 and the EKF.

(see Table 8.2) and occupancy grid mapping (see Table 9.1). The individual functions used in this algorithm are variants of those used in the localization and mapping algorithms.

In particular, the function $\mathbf{measurement_model_map}(z_t, x_t^{[k]}, m_{t-1}^{[k]})$ computes the likelihood of the measurement z_t given the pose $x_t^{[k]}$ represented by the k-th particle and given the map $m_{t-1}^{[k]}$ computed based on the previous measurement and the trajectory represented by this particle. Furthermore, the function $\mathbf{updated_occupancy_grid}(z_t, x_t^{[k]}, m_{t-1}^{[k]})$ computes a new occupancy grid map, given the current pose $x_t^{[k]}$ of the k-th particle, the map $m_{t-1}^{[k]}$ associated to it, and the measurement z_t.

13.10.2 Empirical Insights

Figure 13.17 shows a typical situation of the application of the grid-based FastSLAM algorithm. Shown there are three particles together with their associated maps. Each particle represents a potential trajectory of the robot, which explains why each occupancy grid map looks different. The center map is the best in terms of global consistency.

A typical map acquired with the FastSLAM algorithm is depicted in Figure 13.19. The size of this environment is 28m $\times$ 28m. The length of the robot's trajectory is 491m and the average speed was 0.19m/s. The reso-

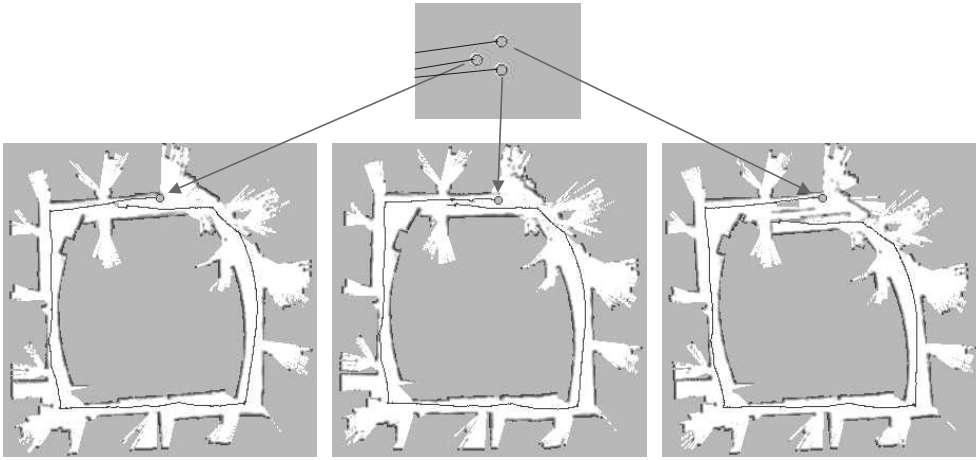


Figure 13.17 Application of the grid-based variant of the FastSLAM algorithm. Each particle carries its own map and the importance weights of the particles are computed based on the likelihood of the measurements given the particle's own map.

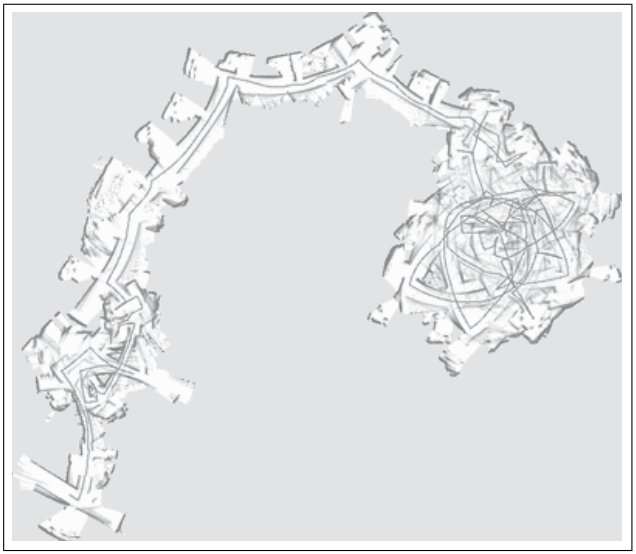


Figure 13.18 Occupancy grid map generated from laser range data and based on pure odometry. All images courtesy of Dirk Hähnel, University of Freiburg.

Figure 13.19 Occupancy grid map corresponding to the particle with the highest accumulated importance weight obtained by the algorithm listed in Table 13.4 from the data depicted in Figure 13.18. The number of particles to create this experiment was 500. Also depicted in this image is the path represented by the particle with the maximum accumulated importance weight.

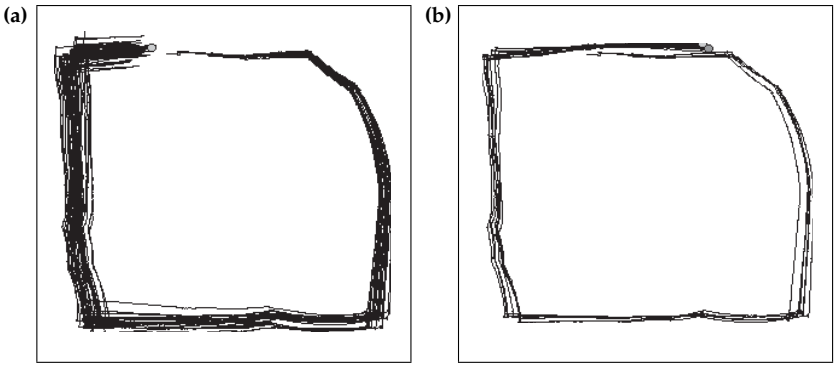


Figure 13.20 Trajectories of all samples shortly before (left) and after (right) closing the outer loop of the environment depicted in Figure 13.19. Images courtesy of Dirk Hähnel, University of Freiburg.

```
Algorithm FastSLAM_occupancy_grids(X_{t-1}, u_t, z_t):
    X̄_t = X_t = ∅
    for k = 1 to M do
        x_t^[k] = sample_motion_model(u_t, x_{t-1}^[k])
        w_t^[k] = measurement_model_map(z_t, x_t^[k], m_{t-1}^[k])
        m_t^[k] = updated_occupancy_grid(z_t, x_t^[k], m_{t-1}^[k])
        X̄_t = X̄_t + ⟨x_t^[k], m_t^[k], w_t^[k]⟩
    endfor
    for k = 1 to M do
        draw i with probability ∝ w_t^[i]
        add ⟨x_t^[i], m_t^[i]⟩ to X_t
    endfor
    return X_t
```

Table 13.4 The FastSLAM algorithm for learning occupancy grid maps.

lution of the map is 10cm. To learn this map, as few as 500 particles were used. During the overall process the robot encountered two loops. A map calculated from pure odometry data is shown in Figure 13.18, illustrating the amount of error in the robot's odometry.

The importance of using multiple particles becomes evident in Figure 13.20, which visualizes the trajectories of the samples shortly before and after closing a loop. As the left image illustrates, the robot is quite uncertain about its position relative to the starting position, hence the wide spread of particles at the time of loop closure. However, a few resampling steps after the robot re-enters known terrain suffice to reduce the uncertainty drastically (right image).

13.11 Summary

This chapter has presented the particle filter approach to the SLAM problem, known as the FastSLAM algorithm.

- The basic idea of FastSLAM is to maintain a set of particles. Each particle contains a sampled robot path. It also contains a map, but here each feature in the map is represented by its own, local Gaussian. The resulting representation requires space linear in the size of the map, and linear in the number of particles.

- The "trick" to represent a map as a set of separate Gaussians—instead of one big joint Gaussian as was the case in the EKF SLAM algorithm—is possible because of the factorial structure of the SLAM problem. Specifically, we noticed that the map features are conditionally independent given the path. By factoring out the path (one per particle), we can then simply treat each map feature as independent, thereby avoiding the costly step of maintaining the correlations between them that plagued the EKF approach.

- The update in FastSLAM follows directly that of the conventional particle filter: Sample a new pose, then update the observed features. This update can be performed online, and FastSLAM is a solution to the online SLAM problem.

- Consequently, we noted that FastSLAM solves both SLAM problems: The offline SLAM problem and the online SLAM problem. Our derivation treated FastSLAM as an offline technique, in which particles represented samples in path space, not just in the momentary pose space. However, it so turned out that none of the update steps require knowledge of any pose other than the most recent one, so one can safely discard past pose estimates. This makes it possible to run FastSLAM as a filter. It also avoids that the size of particles grow linearly with time.

- We encountered two different instantiations of FastSLAM, with version numbers 1.0 and 2.0. FastSLAM 2.0 is an advanced version of FastSLAM. It differs from the base version in one key idea: FastSLAM 2.0 incorporates the measurement when sampling a new pose. The math for doing so was somewhat more involved, but FastSLAM 2.0 is found to be superior in that it requires fewer particles than FastSLAM 1.0.

- The idea of using particle filters makes it possible to estimate data association variables on a per-particle basis. Each particle can be based on a different data association. This provides FastSLAM with a straightforward and powerful mechanism for solving data association problems in SLAM. Previous algorithms, specifically the EKF, GraphSLAM, and the SEIF, are forced to adopt a single data association decision for the entire filter at any point in time, and therefore require more care in selecting the value of the data association.

- For efficiently updating the particles over time, we discussed tree representations of the map. These representations make it possible to reduce the complexity of FastSLAM updates from linear to logarithmic, enabling particles to share parts of the maps that are identical. The ideas of such trees is important in practice, as it enables FastSLAM to scale to 10^9 or more features in the map.

- We also discussed techniques for utilizing negative information. One pertains to the removal of features from the map that are not supported by sufficient evidence. Here FastSLAM adopts an evidence integration approach familiar from the occupancy grid map chapter. Another pertains to the weighting of particles themselves: When failing to observe a feature in the map of a particle, such a particle can be devalued by multiplying its importance weight accordingly.

- We discussed a number of practical properties of the two FastSLAM algorithms. Experiments show that both algorithms perform well in practice, both in feature-based maps and in volumetric occupancy grid-style maps. From a practical perspective, FastSLAM is among the best probabilistic SLAM techniques presently available. Its scaling properties are only rivaled by some of the information filter algorithms described in the two previous chapters.

- We extended FastSLAM 2.0 to different map representations. In one representation, the map was composed of points detected by a laser range finder. In this case, we were able to abandon the idea of modeling the uncertainty in the features through Gaussians, and rely on scan matching techniques to implement the forward sampling process of FastSLAM 2.0. The use of a particle filter leads to a robust loop closure technique.

Possibly the biggest limitation of FastSLAM is the fact that maintains dependencies in the estimates of feature locations only implicitly, through the

diversity of its particle set. In certain environments this can negatively affect the convergence speed when compared to Gaussian SLAM techniques. When using FastSLAM care has to be taken to reduce the damaging effects of the particle deprivation problem in FastSLAM.

13.12 Bibliographical Remarks

The idea of computing distributions over sets of variables by combining samples with a parametric density function is due to Rao (1945) and Blackwell (1947). Today, this idea has become a common tool in the statistics literature (Gilks et al. 1996; Doucet et al. 2001). A first mapping algorithm that uses particle filters for closing loops can be found in Thrun et al. (2000b). The formal introduction of Rao-Blackwellized particle filters into the field of SLAM is due to Murphy (2000a); Murphy and Russell (2001), who developed this idea in the context of occupancy grid maps.

The FastSLAM algorithm was first developed by Montemerlo et al. (2002a), who also developed a tree representation for efficiently maintaining multiple maps. The extension of this algorithm to high-resolution maps is due to Eliazar and Parr, whose algorithm *DP-SLAM* has generated maps from laser range scans with unprecedented accuracy and detail. Their central data structure is called *ancestry trees*, which extends FastSLAM's trees to update problems in occupancy-style maps. A more efficient version is known as DP-SLAM 2.0 (Eliazar and Parr 2004). The FastSLAM 2.0 algorithm was developed by Montemerlo et al. (2003b). It is based on prior work by van der Merwe et al. (2001), who pioneered the idea of using the measurement as part of the proposal distribution in particle filter theory. The grid-based FastSLAM algorithm in this chapter is due to Hähnel et al. (2003b), who integrated the improved proposal distribution idea with Rao-Blackwellized filters applied to grid-based maps. A Rao-Blackwellized filter for tracking the status of doors in a dynamic office environment is described in Avots et al. (2002).

One of FastSLAM's most important contributions lies in the area of data association, so a word is on order on the literature of data association in SLAM. The original SLAM work (Smith et al. 1990; Moutarlier and Chatila 1989a) resorted to maximum likelihood data association, as derived in detail by Dissanayake et al. (2001). A key limitation of these data association techniques was the inability to enforce *mutual exclusivity*: Two different features seen within a single sensor measurement (or in short temporal succession) cannot possibly correspond to the same physical feature in the world. Realizing this, Neira and Tardós developed techniques for testing correspondence for sets of features, which greatly reduced the number of data association errors. To accommodate the huge number of potential associations (exponential in the number of features considered at each point in time), Neira et al. (2003) proposed random sampling techniques in the data association space. All of these techniques, however, maintained a single mode in the SLAM posterior. Feder et al. (1999) applied the greedy data association idea to sonar data, but implemented a delayed decision to resolve ambiguities.

The idea of maintaining multi-modal posterior in SLAM goes back to Durrant-Whyte et al. (2001), whose algorithm *sum of Gaussians* uses a Gaussian mixture for representing the posterior. Each mixture component corresponded to a different trace through the history of all data association decisions. FastSLAM follows this idea, but using particles instead of Gaussian mixture components. The idea of lazy data association can be traced back to other fields, such as the popular RANSAC algorithm (Fischler and Bolles 1981) in computer vision. The tree algorithm presented in the previous chapter is due to Hähnel et al. (2003a). As mentioned there, it parallels work by Kuipers et al. (2004). An entirely different approach for data association is

described in Shatkay and Kaelbling (1997); Thrun et al. (1998b), which use the *expectation maximization* algorithm for resolving correspondence problems (see (Dempster et al. 1977)). The *EM algorithm* iterates a phase of data association for all features with a phase of map building, thereby performing search simultaneously in the space of numerical map parameters, and in the space of discrete correspondences. Araneda (2003) successfully uses MCMC techniques for data association in offline SLAM.

The data association problem arises naturally in the context of multi-robot map integration. A number of papers have developed algorithms for localizing one robot relative to another under the assumption that they both operate in the same environment, and their maps overlap (Gutmann and Konolige 2000; Thrun et al. 2000b). Roy and Dudek (2001) developed a technique by which robots had to rendezvous for integrating their information. For the general case, however, a data association technique must consider the possibility that maps might not even overlap. Stewart et al. (2003) developed a particle filter algorithm that explicitly models the possibility that maps do not overlap. Their algorithm contains a Bayesian estimator for calculating this probability, which takes into consideration how "common" specific local maps are in the environment. The idea of matching sets of features for data association in multi-robot mapping is due to Dedeoglu and Sukhatme (2000); see also Thrun and Liu (2003). A new non-planar representation of maps was proposed by Howard (2004), for circumvented inconsistency in incomplete maps. His approach led to remarkably accurate multi-robot mapping results (Howard et al. 2004).

13.13 Exercises

1. Name three key, distinct advantages for each of the following SLAM algorithms: EKF, GraphSLAM, and FastSLAM.

2. Describe a set of circumstances under which FastSLAM 1.0 will fail to converge and FastSLAM 2.0 will converge to a correct map (with probability 1).

3. On page 443, we stated that *conditioning on the most recent pose x_t instead of the entire path $x_{1:t}$ is insufficient, as dependencies may arise through previous poses.* Prove this assertion. You might prove it with an example.

4. FastSLAM generates many different maps, one for each particle. This chapter left open how to combine these maps into a single posterior estimate. Suggest two such methods, one for FastSLAM with known correspondence, and one for FastSLAM with per-particle data association.

5. The improvement of FastSLAM 2.0 over FastSLAM 1.0 lies in the nature of the proposal distribution. Develop the same for Monte Carlo Localization: Devise a similar proposal distribution for MCL in feature-based maps, and state the resulting algorithm "MCL 2.0." For this exercise you might assume known correspondence.

6. Chapter 13.8 describes an efficient tree implementation, but it does not provide pseudo-code. In this exercise, you are asked to provide the corresponding data structures and update equations for the tree, assuming that the number of features is known in advance, and that we do not face a correspondence problem.

7. In this question, you are asked to verify empirically that FastSLAM indeed maintains the correlations between feature estimates and the robot pose estimate. Specifically, you are asked to implement a simple FastSLAM 1.0 algorithm for linear Gaussian SLAM. Recall from previous exercises that the motion and measurement equations in linear Gaussian SLAM are linear with additive Gaussian noise:

$$\begin{aligned} x_t &\sim \mathcal{N}(x_{t-1} + u_t, R) \\ z_t &= \mathcal{N}(m_j - x_t, Q) \end{aligned}$$

   Run FastSLAM 1.0 in simulation. After t steps, fit a Gaussian over the joint space of feature locations and the robot pose. Compute the correlation matrix from this Gaussian, and characterize the strength of the correlations as a function of t. What are your observations?

8. As mentioned in the text, FastSLAM is a Rao-Blackwellized particle filter. In this question you are asked to design a Rao-Blackwellized filter for a different problem: Localization with a robot that systematically drifts. Systematic drift is a common phenomena in odometry; just inspect Figures 9.1 and 10.7 for two particularly strong instances of drift. Suppose you are given a map of the environment. Can you design a Rao-Blackwellized filter that simultaneously estimates the drift parameters of the robot and the global location of the robot in the environment? Your filter should combine particle filters with Kalman filters.

PART IV

Planning and Control

14 *Markov Decision Processes*

14.1 Motivation

This is the first chapter on probabilistic *planning* and *control* in this book. Thus far, the book has focused exclusively on robot perception. We have discussed a range of probabilistic algorithms that estimate quantities of interest from sensor data. However, the ultimate goal of any robot software is to choose the right actions. This and the following chapters will discuss probabilistic algorithms for action selection.

To motivate the study of probabilistic planning algorithms, consider the following examples.

1. A robotic manipulator grasps and assembles parts arriving in random configuration on a conveyor belt. The configuration of a part is unknown at the time it arrives, yet the optimal manipulation strategy requires knowledge of the configuration. How can a robot manipulate such pieces? Will it be necessary to sense? If so, are all sensing strategies equally good? May there exist manipulation strategies that result in a well-defined configuration without sensing?

2. An underwater vehicle shall travel from Canada to the Caspian Sea. Shall it take the shortest route through the North Pole, running risk of loosing orientation under the Ice? Or should it take the longer route through open waters, where it can regularly relocalize using GPS, the satellite-based global positioning system? To what extent do such decisions depend on the accuracy of the submarine's inertial sensors?

3. A team of robots explores an unknown planet, seeking to acquire a joint map. Shall the robots seek each other to determine their relative location to each other? Or shall they instead avoid each other so that they can

cover more unknown terrain in shorter amounts of time? How does the optimal exploration strategy change when the relative starting locations of the robots are unknown?

These examples illustrate that action selection in many robotics tasks is closely tied to the notion of uncertainty. In some tasks, such as robot exploration, reducing uncertainty is the direct goal of action selection. Such problems are known as *information gathering tasks*. They will be studied in Chapter 17. In other cases, reducing uncertainty is merely a means to achieving some other goal, such as reliably arriving at a target location. These tasks will be studied in this and the next chapters.

INFORMATION GATHERING TASK

From an algorithm design perspective, it is convenient to distinguish two types of uncertainty: uncertainty in action effects, and uncertainty in perception.

First, we distinguish *deterministic* from stochastic action effects. Many theoretical results in robotics are based on the assumption that the effects of control actions are deterministic. In practice, however, actions cause uncertainty, as outcomes of actions are non-deterministic. The uncertainty arising from the stochastic nature of the robot and its environments mandates that the robot senses at execution time, and reacts to unanticipated situations—even if the environment state is fully observable. It is insufficient to plan a single sequence of actions and blindly execute it at run-time.

Second, we distinguish *fully observable* from *partially observable systems*. Classical robotics often assumes that sensors can measure the full state of the environment. If this was always the case, then we would not have written this book! In fact, the contrary appears to be the case. In nearly all interesting real-world robotics problems, sensor limitations are a key factor.

Obviously, robots should consider their *current uncertainty* when determining what to do. When selecting a control action, at a minimum a robot should consider the various outcomes (which might include catastrophic failures), and weigh them by the probability that such outcomes might actually occur. However, robot control must also cope with future, *anticipated uncertainty*. An example of the latter was given above, where we discussed a robot that has the choice between a shorter path through a GPS-denied environment, with a longer one that reduces the danger of getting lost. Minimizing anticipated uncertainty is essential for many robotic applications.

ANTICIPATED UNCERTAINTY

Throughout this chapter, we will take a very liberal view and make virtually no distinction between planning and control. Fundamentally, both planning and control address the same problem: to select actions. They dif-

TESLA EXHIBIT 1011, Page 488

fer in the time constraints under which actions have to be selected, and in the role of sensing during execution. The algorithms described in this chapter are all similar in that they require an off-line optimization, or planning, phase. The result of this planning phase is a control policy, which prescribes a control action for any reasonable situation. In other words, the control policy is effectively a controller, in the sense that it can be used to determine robot actions with minimum computing time. By no means is the choice of algorithms meant to suggest that this is the only way to cope with uncertainty in robotics. However, it reflects the style of algorithms that are currently in use in the field of probabilistic robotics.

The majority of algorithms discussed in this chapter assume finite state and action spaces. Continuous spaces are approximated using grid-style representations.

The following four chapters are organized as follows.

- This chapter discusses in depth the role of the two types of uncertainty and lays out their implications on algorithm design. As a first solution to a restricted class of problems, we introduce *value iteration*, a popular planning algorithm for probabilistic systems. Our discussion in this chapter addresses only the first type of uncertainty: the uncertainty in robot motion. It rests on the assumption that the state is fully observable. The underlying mathematical framework is known as *Markov decision processes* (MDP).

- Chapter 15 generalizes the value iteration technique to both types of uncertainty, in action effects and perception. This algorithm applies value iteration to a belief state representation. The framework underlying this algorithm is called partially observable Markov decision processes (POMDPs). POMDP algorithms anticipate uncertainty, actively gather information, and explore optimally in pursuit of an arbitrary performance goal. Chapter 15 also discusses a number of efficient approximations which can calculate control policies more efficiently.

- Chapter 16 introduces a number of approximate value iteration algorithms for POMDPs. These algorithms have in common that they approximate the probabilistic planning process, so as to gain computational efficiency. One of these algorithms will shortcut probabilistic planning by assuming that at some point in the future, the state becomes fully observable. Another compresses the belief state into a lower dimensional representation, and plans using this representation. A third algorithm uses

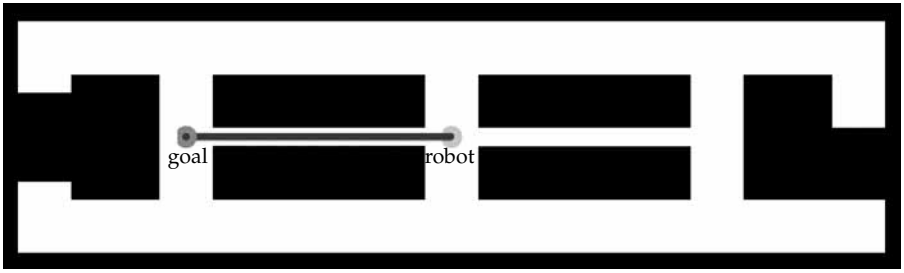


Figure 14.1 Near-symmetric environment with narrow and wide corridors. The robot starts at the center with unknown orientation. Its task is to move to the goal location on the left.

particle filters and a machine learning approach to condense the problem space. All three of these algorithms yield considerable computational improvements while still performing well in practical robotic applications.

- Chapter 17 addresses the specialized problem of robot exploration. Here the robot's goal is to accumulate information about its environment. While exploration techniques address the problem of sensor uncertainty, the problem is significantly easier than the full POMDP problem, and hence can be solved much more efficiently. Probabilistic exploration techniques are popular in robotics, as robots are frequently used for acquiring information about unknown spaces.

14.2 Uncertainty in Action Selection

Figure 14.1 shows a toy-like environment that we will use to illustrate the different types of uncertainty a robot may encounter. Shown in this figure is a mobile robot in a corridor-like environment. The environment is highly symmetric; the only distinguishing feature are its far ends, which are shaped differently. The robot starts at the location indicated, and it seeks to reach the location labeled goal. We notice there exist multiple paths to the goal, one that is short but narrow, and two others that are longer but wider.

In the classical robot planning paradigm, there is no uncertainty. The robot would simply know its initial pose and the location of the goal. Furthermore, actions when executed in the physical world have predictable effects, and such effects can be pre-planned. In such a situation, there is no need to sense. It suffices to plan off-line a single sequence of actions, which can

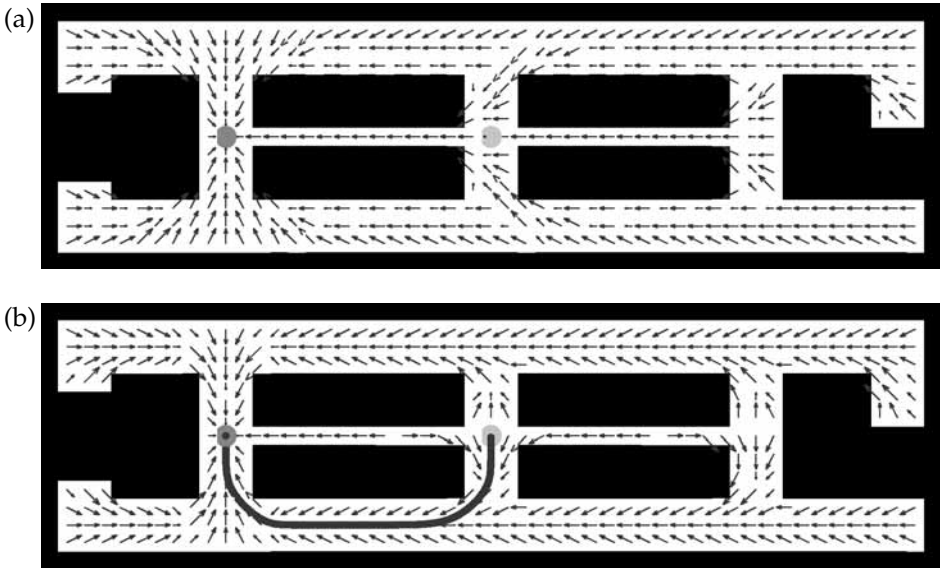


Figure 14.2 The value function and control policy for an MDP with (a) deterministic and (b) nondeterministic action effects. Under the deterministic model, the robot is perfectly fine to navigate through the narrow path; it prefers the longer path when action outcomes are uncertain, to reduce the risk of colliding with a wall. Panel (b) also shows a path.

then be executed at run-time. Figure 14.1 shows an example of such a plan. Obviously, in the absence of errors in the robot's motion, the narrow shorter path is superior to any of the longer, wider ones. Hence, a "classical" planner would choose the former path over the latter.

In practice, such plans tend to fail, for more than one reason. A robot blindly following the narrow hallway runs danger of colliding with the walls. Furthermore, a blindly executing robot might miss the goal location because of the error it accrued during plan execution. In practice, thus, planning algorithms of this type are often combined with a sensor-based, reactive control module that consults sensor readings to adjust the plan so as to avoid collisions. Such a module might prevent the robot from colliding in the narrow corridor. However, in order to do so it may have to slow down the robot, making the narrow path inferior to the wider, longer path.

A paradigm that encompasses uncertainty in robot motion is known as *Markov decision processes*, or *MDPs*. MDPs assume that the state of the en-

MARKOV DECISION PROCESS

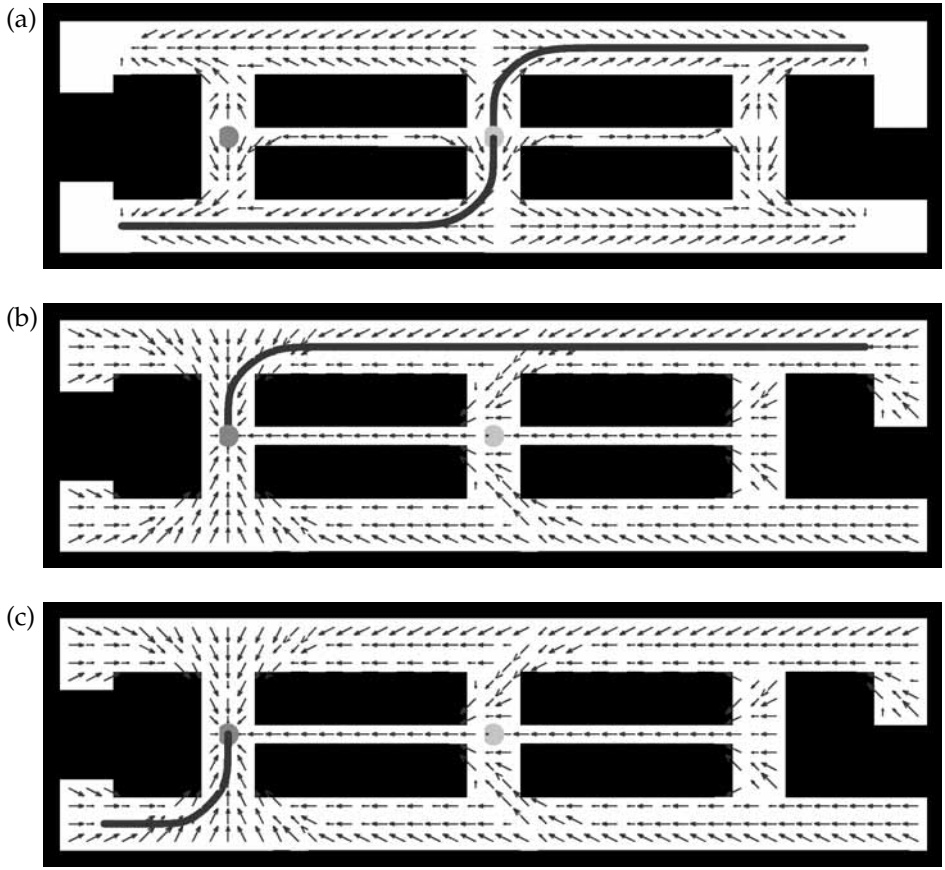


Figure 14.3 Knowledge gathering actions in POMDPs: To reach its goal with more than 50% chance, the belief space planner first navigates to a location where the global orientation can be determined. Panel (a) shows the corresponding policy, and a possible path the robot may take. Based on its location, the robot will then find itself in panel (b) or (c), from where it can safely navigate to the goal.

vironment can be fully sensed at all times. In other words, the perceptual model $p(z \mid x)$ is deterministic and bijective. However, the MDP framework allows for stochastic action effects. The action model $p(x' \mid u, x)$ may be non-deterministic. As a consequence, it is insufficient to plan a single sequence of actions. Instead, the planner has to generate actions for a whole range of situations that the robot might find itself in, because of its actions or

other unpredictable environment dynamics. Our way to cope with the resulting uncertainty shall be to generate a *policy for action selection* defined for all states that the robot might encounter. Such mappings from states to actions are known under many names, such as *control policy*, *universal plans*, and *navigation functions*. An example of a policy is shown in Figure 14.2. Instead of a single sequence of actions, the robot calculates a mapping from states to actions indicated by the arrows. Once such a mapping is computed, the robot can accommodate non-determinism by sensing the state of the world, and acting accordingly. The panel in Figure 14.2a shows a policy for a robot with nearly no motion uncertainty, in which cases the narrow path is indeed acceptable. Figure 14.2b depicts the same situation for an increased randomness in robot motion. Here the narrow path makes a collision more likely, and the detour becomes preferable. This example illustrates two things: the importance of incorporating uncertainty in the motion planning process; and the ability of finding a good control policy using the algorithm described in this chapter.

CONTROL POLICY

Let us now return to the most general, fully probabilistic case, by dropping the assumption that the state is fully observable. This case is known as *partially observable Markov decision processes*, or *POMDPs*. In most if not all robotics applications, measurements z are noisy projections of the state x. Hence, the state can only be estimated up to a certain degree. To illustrate this, consider once again our example, but under different assumptions. Assume that the robot knows its initial location but does not know whether it is oriented towards the left or to the right. Further, it has no sensor to sense whether it has arrived at the goal location.

PARTIALLY OBSERVABLE MARKOV DECISION PROCESS

Clearly, the symmetry of the environment makes it difficult to disambiguate the orientation. By moving directly towards the projected goal state the robot faces a 50% chance of missing the goal, and instead moving to the symmetric location on the right size of the environment. The optimal plan, thus, is to move to any of the corners of the environment, which is perceptually sufficiently rich to disambiguate its orientation. A policy for moving to these locations is shown in Figure 14.3a. Based on its initial orientation, the robot may execute any of the two paths shown there. As it reaches a corner, its sensors now reveal its orientation, and hence its actual location relative to the environment. The robot might now find itself in any of the two situations depicted in Figures 14.3b&c, from where it can safely navigate to the goal location as shown.

This example illustrates one of the key aspects of probabilistic robotics. The robot has to actively gather information, and in order to do so, might be

suffering a detour relative to a robot that knows its state with absolute certainty. This problem is paramount in robotics. For nearly any robotics task, the robot's sensors are characterized by intrinsic limitations as to what the robot can know, and where information can be acquired. Similar situations occur in tasks as diverse as locate-and-retrieve, planetary exploration, urban search-and-rescue, and so on.

The question now arises as to how can one devise an algorithm for action selection that can cope with this type of uncertainty. As we will learn, this is not a trivial question. One might be tempted to solve the problem of what to do by analyzing each possible situation that might be the case under the current state of knowledge. In our example, there are two such cases: the case where the goal is on the upper left relative to the initial robot heading, and the case where the goal is on the lower right. In both these cases, however, the optimal policy does not bring the agent to a location where it would be able to disambiguate its pose. That is, the planning problem in a partially observable environment cannot be solved by considering all possible environments and averaging the solution.

Instead, the key idea is to generate plans in the *belief space*, a synonym for *information space*. The belief space comprises the space of all posterior beliefs that the robot might hold about the world. The belief space for our simple example corresponds to the three panels in Figure 14.3. The top panel shows such a belief space policy. It displays the initial policy while the robot is unaware of its orientation. Under this policy, the robot navigates to one of the corners of the environment where it can localize. Once localized, it can safely navigate to the target location, as illustrated in the two lower panels of Figure 14.3. Since the a priori chance of each orientation is the same, the robot will experience a random transition with a 50% chance of ending up in either of the two bottom diagrams.

In our toy example, the number of different belief states happens to be finite: either the robot knows or it does not have a clue. In practical applications, this is usually not the case. In worlds with finitely many states the belief space is usually continuous, but of finite dimensionality. In fact, the number of dimensions of the belief space is of the same order as the number of states in the underlying state space. If the state space is continuous, the belief space possesses infinitely many dimensions.

This example illustrates a fundamental property that arises from the robot's inability to perfectly sense the state of the world—one whose importance for robotics has often been under-appreciated. In uncertain worlds a robot planning algorithm must consider the state of its knowledge when

TESLA EXHIBIT 1011, Page 494

making control decisions. In general it does not suffice to consider the most likely state only. By conditioning the action on the belief state—as opposed to the most likely actual state—the robot can actively pursue information gathering. In fact, the optimal plan in belief state gathers information "optimally," in that it only seeks new information to the extent that it is actually beneficial to the expected utility of the robot's action. The ability to devise optimal control policies is a key advantage of the probabilistic approach to robotics over the classical deterministic, omniscient approach. However, as we shall see soon, it comes at the price of an increased complexity of the planning problem.

14.3 Value Iteration

Our first algorithm for finding control policies is called *value iteration*. Value iteration recursively calculates the utility of each action relative to a payoff function. Our discussion in this chapter will be restricted to the first type of uncertainty: stochasticity of the robot and the physical world. We defer our treatment of uncertainty arising from sensor limitations to the subsequent chapter. Thus, we will assume that the state of the world is fully observable at any point in time.

14.3.1 Goals and Payoff

Before describing a concrete algorithm, let us first define the problem in more concise terms. In general, robotic action selection is driven by *goals*. Goals might correspond to specific configurations (e.g., a part has successfully been picked up and placed by a robot manipulator), or they might express conditions over longer periods of time (e.g., a robot balances a pole). In robotics, one is sometimes concerned with reaching a specific goal configuration, while simultaneously optimizing other variables, often thought of as *cost*. For example, one might be interested in moving the end-effector of a manipulator to a specific location, while simultaneously minimizing time, energy consumption, or the number of collisions with obstacles.

GOAL

COST

At first glance, one might be tempted to express these desires by two quantities, one that is being maximized (e.g., the binary flag that indicates whether or not a robot reached its goal location), and the other one being minimized (e.g., the total energy consumed by the robot). However, both can be expressed using a single function called the *payoff function*.

PAYOFF FUNCTION

The payoff, denoted r, is a function of the state and the robot control. For example, a simple payoff function may be the following:

$$r(x,u) = \begin{cases} +100 & \text{if } u \text{ leads to a goal configuration or state} \\ -1 & \text{otherwise} \end{cases} \tag{14.1}$$

This payoff function "rewards" the robot with $+100$ if a goal configuration is attained, while it "penalizes" the robot by -1 for each time step where it has not reached that configuration. Such a payoff function will yield maximum cumulative return if the robot reaches its goal configuration in the minimum possible time.

Why use a single payoff variable to express both goal achieval and costs? This is primarily because of two reasons: First, the notation avoids clutter in the formulae yet to come, as it shall unify our treatment of costs and goal achieval throughout this book. Second, and more importantly, it pays tribute to the fundamental trade-off between goal achievement and costs along the way. Since robots are inherently uncertain, they cannot know with certainty as to whether a goal configuration has been achieved; instead, all one can hope for is to maximize the chances of reaching a goal. This trade-off between goal achieval and cost is characterized by questions like *Is increasing the probability of reaching a goal worth the extra effort (e.g., in terms of energy, time)?* Treating both goal achieval and costs as a single numerical factor enables us to trade off one against the other, hence providing a consistent framework for selecting actions under uncertainty.

We are interested in devising programs that generate actions so as to optimize future payoff in expectation. Such a program is usually referred to as a *control policy*, or simply *policy*. It will be denoted as follows:

POLICY

$$\pi : z_{1:t-1}, u_{1:t-1} \longrightarrow u_t \tag{14.2}$$

In the case of full observability, we assume the much simpler case:

$$\pi : x_t \longrightarrow u_t \tag{14.3}$$

Thus, a policy π is a function that maps past data into controls, or states into controls when the state is observable.

So far, our definition of a control policy makes no statements about its computational properties. It might be a fast, reactive algorithm that bases its decision on the most recent data item only, or an elaborate planning algorithm. In practice, however, computational considerations are essential, in that any delay in calculating a control may negatively affect a robot's performance. Our definition of a policy π also makes no commitment as to whether it is deterministic or non-deterministic.

TESLA EXHIBIT 1011, Page 496

PLANNING HORIZON

An interesting concept in the context of creating control policies is the *planning horizon*. Sometimes, it suffices to choose a control action so as to maximize the immediate next payoff value. Most of the time, however, an action might not pay off immediately. For example, a robot moving to a goal location will receive the final payoff for reaching its goal only after the very last action. Thus, payoff might be delayed. An appropriate objective is then to choose actions so that the sum of all future payoff is maximal. We will call this sum the *cumulative payoff*. Since the world is non-deterministic, the best one can optimize is the *expected cumulative payoff*, which is conveniently written as

CUMULATIVE PAYOFF

$$R_T = E\left[\sum_{\tau=1}^{T} \gamma^\tau r_{t+\tau}\right] \tag{14.4}$$

Here the expectation $E[\,]$ is taken over future momentary payoff values $r_{t+\tau}$ which the robot might accrue between time t and time $t+T$. The individual payoffs $r_{t+\tau}$ are multiplied by a factor γ^τ, called the *discount factor*. The value of γ is a problem-specific parameter, and it is constrained to lie in the interval $[0;1]$. If $\gamma = 1$, we have $\gamma^\tau = 1$ for arbitrary values of τ, and hence the factor can be omitted in Equation (14.4). Smaller values of γ discount future payoff exponentially, making earlier payoffs exponentially more important than later ones. This discount factor, whose importance will be discussed later, bears resemblance to the value of money, which also loses value over time exponentially due to inflation.

DISCOUNT FACTOR

PLANNING HORIZON

We notice that R_T is a sum of T time steps. T is called the *planning horizon*, or simply: *horizon*. We distinguish three important cases:

GREEDY CASE

1. $T = 1$. This is called the *greedy case*, where the robot only seeks to minimize the immediate next payoff. While this approach is degenerate in that it does not capture the effect of actions beyond the immediate next time step, it nevertheless plays an important role in practice. The reason for its importance stems from the fact that greedy optimization is much simpler than multi-step optimization. In many robotics problems, greedy algorithms are currently the best known solutions that can be computed in polynomial time. Obviously, greedy optimization is invariant with respect to the discount factor γ, but it requires that $\gamma > 0$.

FINITE-HORIZON CASE

2. T larger than 1 but finite. This case is known as the *finite-horizon case*. Typically, the payoff is not discounted over time, thus $\gamma = 1$. One might argue that the finite-horizon case is the only one that matters, since for all practical purposes time is finite. However, finite-horizon optimality is often

harder to achieve than optimality in the discounted infinite-horizon case. Why is this so? A first insight stems from the observation that the optimal control action is a function of time horizon. Near the far end of the time horizon, for example, the optimal policy might differ substantially from the optimal choice earlier in time, even under otherwise identical conditions (e.g., same state, same belief). As a result, planning algorithms with finite horizon are forced to maintain different plans for different horizons, which can add undesired complexity.

INFINITE-HORIZON CASE

3. T is infinite. This case is known as the *infinite-horizon case*. This case does not suffer the same problem as the finite horizon case, as the number of remaining time steps is the same for any point in time (it is infinite!). However, here the discount factor γ is essential. To see why, let us consider the case where we have two robot control programs, one that earns us \$1 per hour, and another one that makes us \$100 per hour. In the finite horizon case, the latter is clearly preferable to the former. No matter what the value of the horizon is, the expected cumulative payoff of the second program exceeds that of the first by a factor of a hundred. Not so in the infinite horizon case. Without discounting, both programs will earn us an infinite money, rendering the expected cumulative payoff R_T insufficient to select the better program.

   Under the assumption that each individual payoff r is bounded in magnitude (that is, $|r| < r_{\max}$ for some value $r_{\max}$), discounting guarantees that R_∞ is finite—despite the fact that the sum has infinitely many terms. Specifically, we have

$$R_\infty \leq r_{\max} + \gamma r_{\max} + \gamma^2 r_{\max} + \gamma^3 r_{\max} + \ldots = \frac{r_{\max}}{1-\gamma} \tag{14.5}$$

   This shows that R_∞ is finite as long as γ is smaller than 1. As an aside, we note that an alternative to discounting involves maximizing the *average* payoff instead of the total payoff. Algorithms for maximizing average payoff will not be studied in this book.

Sometimes we will refer to the cumulative payoff R_T conditioned on a state x_t. This will be written as follows:

$$R_T(x_t) = E\left[\sum_{\tau=1}^{T} \gamma^\tau r_{t+\tau} \mid x_t\right] \tag{14.6}$$

The cumulative payoff R_T is a function of the robot's policy for action selection. Sometimes, it is beneficial to make this dependence explicit:

$$R_T^\pi(x_t) = E\left[\sum_{\tau=1}^{T} \gamma^\tau r_{t+\tau} \mid u_{t+\tau} = \pi(z_{1:t+\tau-1}, u_{1:t+\tau-1})\right] \tag{14.7}$$

This notation enables us to compare two control policies π and π', and determine which one is better. Simply compare R_T^π to $R_T^{\pi'}$ and pick the algorithm with higher expected discounted future payoff!

14.3.2 Finding Optimal Control Policies for the Fully Observable Case

This chapter shall be concluded with a concreted value iteration algorithm, for calculating control policies in fully observable domains. At first glance, such algorithms depart from the basic assumption of probabilistic robotics, namely that state is not observable. However, in certain applications one can safely assume that the posterior $p(x_t \mid z_{1:t}, u_{1:t})$ is well-represented by its mean $E[p(x_t \mid z_{1:t}, u_{1:t})]$.

The fully observable case has some merit. The algorithm discussed in turn also prepares us for the more general case of partial observability.

We already noted that the framework of stochastic environments with fully observable state is known as Markov decision processes. Policies, in MDPs, are mappings from state to control actions:

$$\pi : x \longrightarrow u \tag{14.8}$$

The fact that the state is sufficient for determining the optimal control is a direct consequence of our Markov assumption that was discussed in length in Chapter 2.4.4. The goal of planning in the MDP framework is to identify the policy π that maximizes the future cumulative payoff.

Let us begin with defining the optimal policy for a planning horizon of $T = 1$, hence we are only interested in a policy that maximizes the immediate next payoff. This policy shall be denoted $\pi_1(x)$ and is obtained by maximizing the expected 1-step payoff over all controls:

$$\pi_1(x) = \operatorname*{argmax}_u \; r(x, u) \tag{14.9}$$

Thus, an optimal action is one that maximizes the immediate next payoff in expectation. The policy that chooses such an action is optimal in expectation.

VALUE FUNCTION

Every policy has an associated *value function*, which measures the expected value (cumulative discounted future payoff) of this specific policy. For π_1,

the value function is simply the expected immediate payoff, discounted by the factor γ:

$$V_1(x) = \gamma \max_u r(x,u) \tag{14.10}$$

This value for longer planning horizons is now defined recursively. The optimal policy for horizon $T = 2$ selects the control that maximizes the sum of the one-step optimal value $V_1(x)$ and the immediate 1-step payoff:

$$\pi_2(x) = \operatorname*{argmax}_u \left[r(x,u) + \int V_1(x')\, p(x' \mid u, x)\, dx' \right] \tag{14.11}$$

It should be immediately obvious why this policy is optimal. The value of this policy conditioned on the state x is given by the following discounted expression:

$$V_2(x) = \gamma \max_u \left[r(x,u) + \int V_1(x')\, p(x' \mid u, x)\, dx' \right] \tag{14.12}$$

The optimal policy and its value function for $T = 2$ was constructed recursively, from the optimal value function for $T = 1$. This observation suggests that for any finite horizon T the optimal policy, and its associated value function, can be obtained recursively from the optimal policy and value function $T - 1$.

$$\pi_T(x) = \operatorname*{argmax}_u \left[r(x,u) + \int V_{T-1}(x')\, p(x' \mid u, x)\, dx' \right] \tag{14.13}$$

The resulting policy $\pi_T(x)$ is optimal for the planning horizon T. The associated value function is defined through the following recursion:

$$V_T(x) = \gamma \max_u \left[r(x,u) + \int V_{T-1}(x')\, p(x' \mid u, x)\, dx' \right] \tag{14.14}$$

In the infinite horizon case, the optimal value function tends to reach the equilibrium (with the exception of some rare deterministic systems, in which no such equilibrium exists):

$$V_\infty(x) = \gamma \max_u \left[r(x,u) + \int V_\infty(x')\, p(x' \mid u, x)\, dx' \right] \tag{14.15}$$

BELLMAN EQUATION

This invariance is known as *Bellman equation*. Without proof, we notice that every value function V that satisfies the condition (14.15) is both necessary and sufficient for the induced policy to be *optimal*.

14.3.3 Computing the Value Function

This consideration leads to the definition of a practical algorithm for calculating the optimal policy in stochastic systems with full state observability. Value iteration does this by successively approximating the optimal value functions, as defined in (14.15).

In detail, let us denote out value function approximation by $\hat{V}$. Initially, $\hat{V}$ is set to $r_{\min}$, the minimum possible immediate payoff:

$$\hat{V} \longleftarrow r_{\min} \tag{14.16}$$

Value iteration then successively updates the approximation via the following recursive rule, which computes the value function for increasing horizons:

$$\hat{V}(x) \longleftarrow \gamma \max_u \left[r(x,u) + \int \hat{V}(x')\, p(x' \mid u, x)\, dx' \right] \tag{14.17}$$

Since each update propagates information in reverse temporal order through the value function, it is usually referred to as the *backup step*.

BACKUP STEP

The value iteration rule bears close resemblance to the calculation of the horizon-T optimal policy above. Value iteration converges if $\gamma < 1$ and, in some special cases, even for $\gamma = 1$. The order in which states are updated in value iteration is irrelevant, as long as each state is updated infinitely often. In practice, convergence is usually observed after a much smaller number of iterations.

At any point in time, the value function $\hat{V}(x)$ defines a policy:

$$\pi(x) = \operatorname*{argmax}_u \left[r(x,u) + \int \hat{V}(x')\, p(x' \mid u, x)\, dx' \right] \tag{14.18}$$

After convergence of value iteration, the policy that is greedy with respect to the final value function is optimal.

We note that all of these equations have been formulated for general state spaces. For finite state spaces, the integral in each of these equations can be implemented as a finite sum over all states. This sum can often be calculated efficiently, since $p(x' \mid u, x)$ will usually be non-zero for a relatively few states x and x'. This leads to an efficient family of algorithms for calculating value functions.

Table 14.1 shows three algorithms: The general value iteration algorithm **MDP_value_iteration** for arbitrary state and action spaces; its discrete variant for finite state spaces **MDP_discrete_value_iteration**, and the algorithm for retrieving the optimal control action from the value function, **policy_MDP**.

Algorithm MDP_value_iteration( ):

for all x *do*
  $\hat{V}(x) = r_{\min}$
endfor
repeat until convergence
  for all x
    $$\hat{V}(x) = \gamma \max_u \left[r(x,u) + \int \hat{V}(x')\, p(x' \mid u, x)\, dx' \right]$$
  endfor
endrepeat
return $\hat{V}$

Algorithm MDP_discrete_value_iteration( ):

for $i = 1$ *to* N *do*
  $\hat{V}(x_i) = r_{\min}$
endfor
repeat until convergence
  for $i = 1$ *to* N *do*
    $$\hat{V}(x_i) = \gamma \max_u \left[r(x_i,u) + \sum_{j=1}^{N} \hat{V}(x_j)\, p(x_j \mid u, x_i) \right]$$
  endfor
endrepeat
return $\hat{V}$

Algorithm policy_MDP($x, \hat{V}$**):**

return $$\operatorname*{argmax}_u \left[r(x,u) + \sum_{j=1}^{N} \hat{V}(x_j)\, p(x_j \mid u, x_i) \right]$$

Table 14.1 The value iteration algorithm for MDPs, stated here in its most general form and for MDPs with finite state and control spaces. The bottom algorithm computes the best control action.

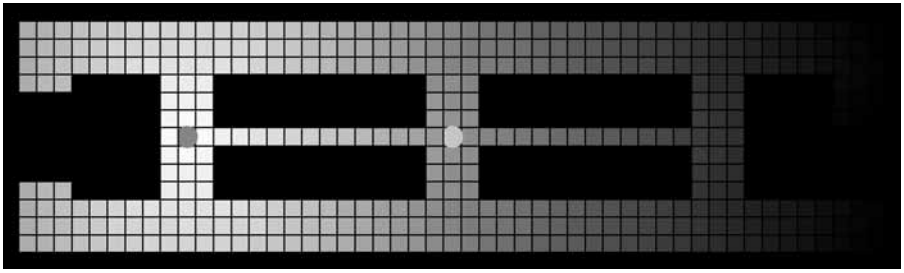


Figure 14.4 An example of an infinite-horizon value function T_∞, assuming that the goal state is an "absorbing state." This value function induces the policy shown in Figure 14.2a.

The first algorithm **MDP_value_iteration** initializes the value function in line 3. Line 5 through 9 implement the recursive calculation of the value function. Once value iteration converges, the resulting value function $\hat{V}$ induces the optimal policy. If the state space is finite, the integral is replaced by a finite sum, as shown in **MDP_discrete_value_iteration**. The factor γ is the discount factor. The algorithm **policy_MDP** processes the optimal value function along with a state x, and returns the control u that maximizes the expected value.

Figure 14.4 depicts an example value function, for our example discussed above. Here the shading of each grid cell corresponds to its value, with white being $V = 100$ and black being $V = 0$. Hill climbing in this value function using Equation 14.18 leads to the policy shown in Figure 14.2a.

14.4 Application to Robot Control

The simple value iteration algorithm is applicable to low-dimensional robot motion planning and control problems. To do so, we have to introduce two approximations.

First, the algorithm in Table 14.5 defines a value function over a continuous space, and requires maximization and integration over a continuous space. In practice, it is common to approximate the state space by a discrete decomposition, similar to our histogram representations in Chapter 4.1. Similarly, it is common to discretize the control space. The function $\hat{V}$ is then easily implemented as a look-up table. However, such a decomposition works only

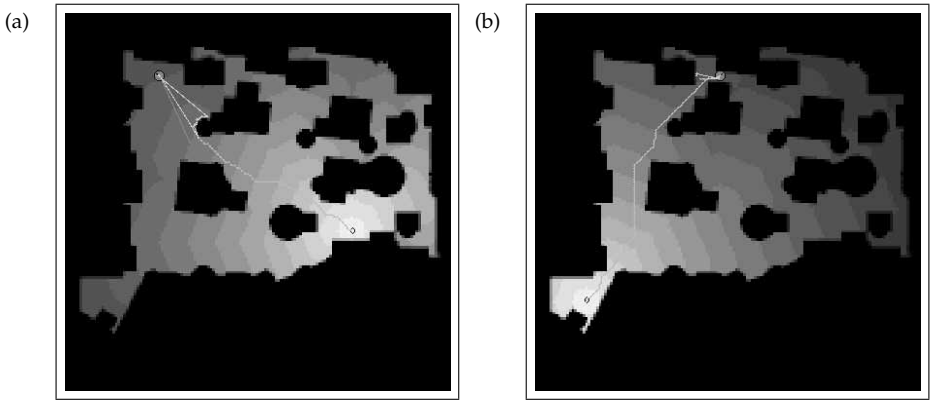


Figure 14.5 Example of value iteration over state spaces in robot motion. Obstacles are shown in black. The value function is indicated by the gray shaded area. Greedy action selection with respect to the value function lead to optimal control, assuming that the robot's pose is observable. Also shown in the diagrams are example paths obtained by following the greedy policy.

for low-dimensional state and control spaces, due to the curse of dimensionality. In higher dimensional situations, it is common to introduce learning algorithms to represent the value function.

Second, we need a state! As noted above, it might be viable to replace the posterior by its mode

$$\hat{x}_t = E[p(x_t \mid z_{1:t}, u_{1:t})] \tag{14.19}$$

In the context of robot localization, for example, such an approximation works well if we can guarantee that the robot is always approximately localized, and the residual uncertainty in the posterior is local. It ceases to work when the robot performs global localization, or if it is being kidnapped.

Figure 14.5 illustrates value iteration in the context of a robotic path planning problem. Shown there is a two-dimensional projection of a configuration space of a circular robot. The configuration space is the space of all $\langle x, y, \theta \rangle$ coordinates that the robot can physically attain. For circular robots, the configuration space is obtained by "growing" the obstacles in the map by the radius of the robot. These increased obstacles shown in black in Figure 14.5.

The value function is shown in gray, where the brighter a location, the higher its value. The path obtained by following the optimal policy leads to

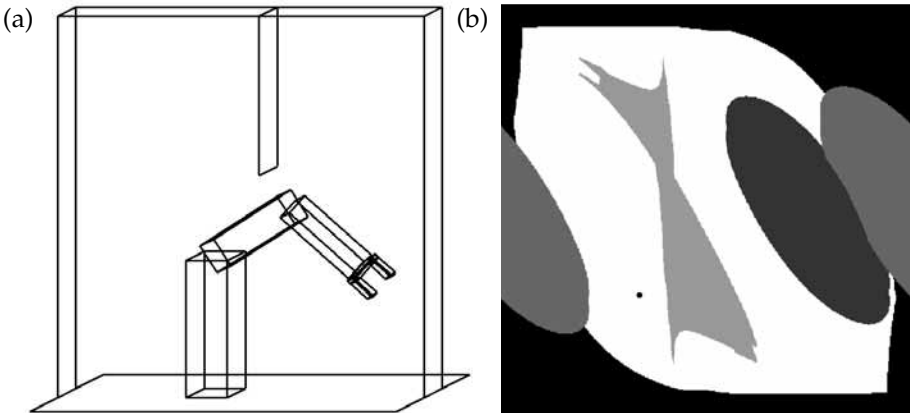


Figure 14.6 (a) 2-DOF robot arm in an environment with obstacles. (b) The *configuration space* of this arm: the horizontal axis corresponds to the shoulder joint, and the vertical axis to its elbow joint configuration. Obstacles are shown in gray. The small dot in this diagram corresponds to the configuration on the left.

the respective goal location, as indicated in Figure 14.5. The key observation is that the value function is defined over the entire state space, enabling the robot to select an action no matter where it is. This is important in non-deterministic worlds, where actions have stochastic effects on the robot's state.

The path planner that generated Figure 14.5 makes specific assumptions in order to keep the computational load manageable. For circular robots that can turn on the spot, it is common to compute the value function in the two-dimensional Cartesian coordinates only, ignoring the cost of rotation. We also ignore state variables such as the robot's velocity, despite the fact that velocity clearly constrains where a robot can move at any given point in time. To turn such a control policy into actual robot controls, it is therefore common practice to combine such path planners with fast, reactive collision avoidance modules that generate motor velocities while obeying dynamic constraints. A path planner that considers the full robot state would have to plan in at least five dimensions, comprising the full pose (three dimensions), the translational and the rotational velocity of the robot. In two dimensions, calculating the value function for environment like the one above takes only a fraction of a second on a low-end PC.

TESLA EXHIBIT 1011, Page 505

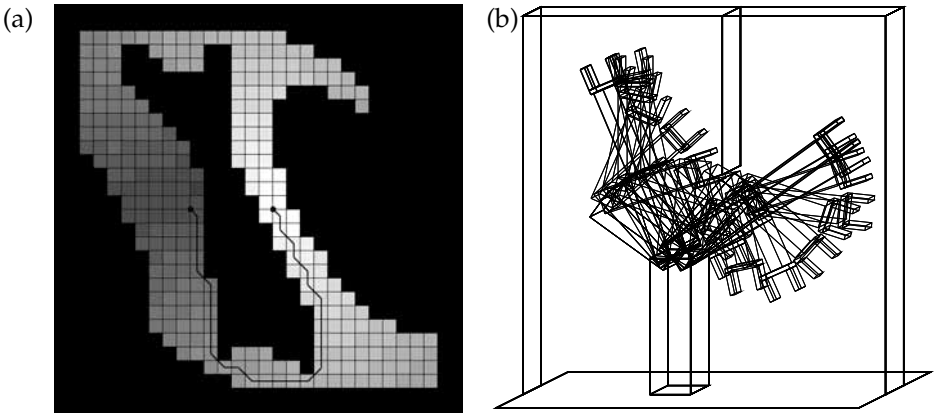


Figure 14.7 (a) Value iteration applied to a coarse discretization of the configuration space. (b) Path in workspace coordinates. The robot indeed avoids the vertical obstacle.

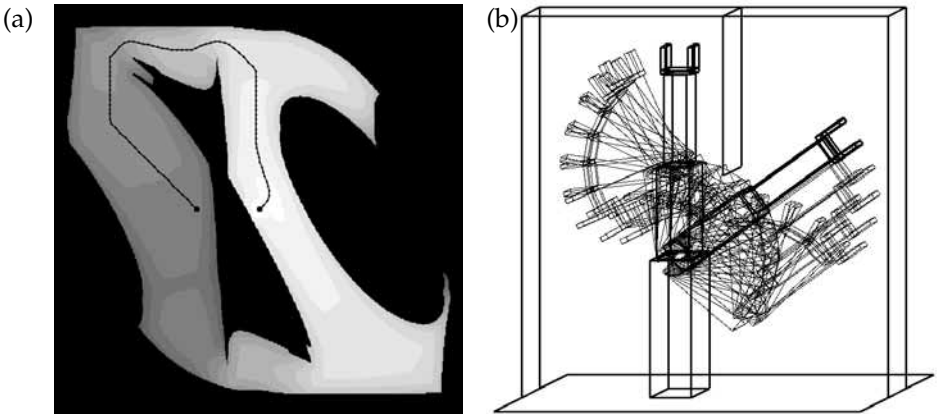


Figure 14.8 (a) Probabilistic value iteration, here over a fine-grained grid. (b) The corresponding path.

A second example is illustrated in Figure 14.6a. Shown there is a robot arm model with two rotational degrees of freedom (DOFs), a shoulder and an elbow joint. Determining the exact configuration of these joints is usually possible, through shaft encoders that are attached to the joints. Hence, the approximation in (14.19) is a valid one. However, robot arm motion is usually subject to noise, and as such the control noise should be taken into account during planning. This makes arm control a primary application for probabilistic MDP-style algorithms.

CONFIGURATION SPACE

Robot arm motion is usually tackled in the *configuration space*. The configuration space for the specific arm is shown in Figure 14.6b. Here the horizontal axis graphs the orientation of the shoulder, and the vertical orientation that of the elbow. Each point in this diagram, thus, corresponds to a specific configuration. In fact, the small dot in Figure 14.6b corresponds to the configuration shown in Figure 14.6a.

It is common to decompose the configuration space into areas in which the robot can move, and areas where it would collide. This is shown in Figure 14.6b. The white area in this figure corresponds to the collision-free configuration space, commonly called freespace. The black boundary in the configuration space is the constraint imposed by the table and the enclosing case. The vertical obstacle, protruding into the robot's workspace from above in Figure 14.6a, corresponds to the light gray obstacle in the center of Figure 14.6b. This figure is not at all obvious, and the reader may take a minute to visualize configurations in which the robot collides with this obstacle.

Figure 14.7a shows the result of value iteration using a coarse discretization of the configurations space. Here the value is propagated using a deterministic motion model and the resulting path is also shown. When executed, this policy leads to a motion shown in Figure 14.7b. Figure 14.8 shows a result for a probabilistic motion model, with the resulting motion of the arm. Again, this is the result of applying value iteration under the assumption that the configuration of the robot arm is fully observable—a rare instance in which this assumption is valid!

14.5 Summary

This chapter introduced the basic framework of probabilistic control.

- We identified the two basic types of uncertainty a robot may face: uncertainty with regards to controls, and uncertainty in perception. The former makes it difficult to determine what lies ahead, whereas the latter makes

it hard to determine what is. Uncertainty from unpredictable events in the environment was silently subsumed into this taxonomy.

- We defined the control objective through a *payoff function*, which maps states and controls to a goodness value. The payoff enables us to express performance goals as well as costs of robot operation. The overall control objective is the maximization of all payoff, immediate and at future points in time. To avoid possibly infinite sums, we introduced a so-called discount factor that exponentially discounts future payoffs.

- We discussed an approach to solving probabilistic control problems by devising a control policy. A control policy defines the control action that is to be chosen, as a function of the robot's information about the world. A policy is optimal if it maximizes the sum of all future cumulative payoffs. The policy is computed in a planning phase that precedes robot operation. Once computed, it specifies the optimal control action for any possible situation a robot may encounter.

- We devised a concrete algorithm for finding the optimal control policy for the restricted case of fully observable domains, in which the state is fully observable. Those domains are known as Markov decision processes. The algorithm involved the calculation of a value function, which measures the expected cumulative payoff. A value function defines a policy of greedily selecting the control that maximizes value. If the value function is optimal, so is the policy. The value iteration algorithm successively improved the value function by updating it recursively.

- We discussed applications of MDP value iteration to probabilistic robotics problems. For that we extracted the mode of the belief as the state, and approximated the value function by a low-dimensional grid. The result was a motion planning algorithm for stochastic environments, which enables a robot to navigate even if its action effects are nondeterministic.

The material in this chapter lays the groundwork for the next chapter, in which we will tackle the more general problem of control under measurement uncertainty, also known as the partially observable MDP problem. We already intuitively discussed why this problem is much harder than the MDP problem. Nevertheless, some of the intuitions and basic algorithms carry over to this case.

We close this chapter by remarking that there exist a number of alternative techniques for probabilistic planning and control under uncertainty. Our

choice of value iteration as the basic method is due to its popularity; further, value iteration techniques are among the best-understood techniques for the more general POMDP case.

Value iteration is by no means the most effective algorithm for generating control. Common planning algorithms include the A* algorithm, which uses a heuristic in the computation of the value function, or direct policy search techniques that identify a locally optimal policy through gradient descent. However, value iteration plays a pivotal role in the next chapter, when we address the much harder case of optimal control under sensor uncertainty.

14.6 Bibliographical Remarks

The idea of dynamic programming goes back to Bellman (1957) and Howard (1960). Bellman (1957) identified the equilibrium equation for value iteration, which has henceforth been called the Bellman equation. Markov decision processes (MDPs) with incomplete state estimation were first discussed by Astrom (1965); see also Mine and Osaki (1970) for early work on MDPs. Since then, dynamic programming for control has been a vast field, as a recent book on this topic attests (Bertsekas and Tsitsiklis 1996). Recent improvements to the basic paradigm include techniques for real-time value iteration (Korf 1988), value iteration guided through the interaction with an environment (Barto et al. 1991), model free value iteration (Watkins 1989), and value iteration with parametric representation of the value function (Roy and Tsitsiklis 1996; Gordon 1995), or using trees (Moore 1991) (see also (Mahadevan and Kaelbling 1996)). Hierarchical value iteration techniques have been developed by Parr and Russell (1998) and Dietterich (2000), and Boutilier et al. (1998) improved the efficiency of MDP value iteration by reachability analysis. There exists also a rich literature on applications of value iteration, for example work by Barniv (1990) on moving target detection. In the light of this rich literature, the material in this chapter is a basic exposition of the most simple value iteration techniques, in preparation of the techniques described in the chapters that follow.

Within robotics, the issue of robot motion planning has typically been investigated in a nonprobabilistic framework. As noted, the assumption is usually that the robot and its environment is perfectly known, and controls have deterministic effects. Complications arise from the fact that the state space is continuous and high-dimensional. The standard text in this field is Latombe (1991). It is predated by seminal work on a number of basic motion planning techniques, visibility graphs (Wesley and Lozano-Perez 1979), potential field control (Khatib 1986), and Canny's (1987) famous silhouette algorithm. Rowat (1979) introduced the idea of Voronoi graphs into the field of robot control, and Guibas et al. (1992) showed how to compute them efficiently. Choset (1996) developed this paradigm into a family of efficient online exploration and mapping techniques. Another set of techniques used randomized (but not probabilistic!) techniques for searching the space of possible robot paths (Kavraki and Latombe 1994). A contemporary book on this topic is due to Choset et al. (2004).

APPROXIMATE CELL DECOMPOSITION

In the language of the robotic motion planning field, the methods discussed in this chapter are *approximate cell decompositions* which, in the deterministic case, provide no guarantee of completeness. Decompositions of continuous spaces into finite graphs have been studied in robotics for decades. Reif (1979) developed a number of techniques for decomposing continuous spaces into finitely many cells that retained completeness in motion planning. The idea of configuration

spaces, necessary to check collision with the techniques described in this chapter, was originally proposed by Lozano-Perez (1983). Recursive cell decomposition techniques for configuration space planning can be found in Brooks and Lozano-Perez (1985), all under the non-probabilistic assumptions of perfect world models and perfect robots. Acting under partial knowledge has been addressed by Goldberg (1993), who in his Ph.D. thesis developed algorithms for orienting parts in the absence of sensors. Sharma (1992) developed robot path planning techniques with stochastic obstacles.

Policy functions that assign to every possible robot state a control action are known as *controllers*. An algorithm for devising a control policy is often called *optimal controller* (Bryson and Yu-Chi 1975). Control policies are also known as *navigation function* (Koditschek 1987; Rimon and Koditschek 1992). Within AI, they are known as *universal plans* (Schoppers 1987), and a number of symbolic planning algorithms have addressed the problem of finding such universal plans (Dean et al. 1995; Kushmerick et al. 1995; Hoey et al. 1999). Some robotic work has been devoted to bridging the gap between universal plans and open-loop action sequences, such as in Nourbakhsh's (1987) Ph.D. work.

We also remark that the notion of "control" in this book is somewhat narrowly defined. The chapter deliberately did not address standard techniques in the rich field of control, such as PID control and other popular techniques often found in introductory textbooks (Dorf and Bishop 2001). Clearly, such techniques are both necessary and applicable in many real-world robot systems. Our choice to omit them is based on space constraints, and on the fact that most of these techniques do not rely on explicit representations of uncertainty.

14.7 Exercises

1. The dynamic programming algorithm uses the most likely state to determine its action. Can you draw a robot environment in which conditioning actions on the most likely state is fundamentally the wrong choice? Can you give a concise reason why this might sometimes be a poor choice?

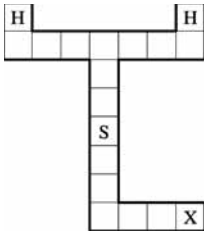
2. Suppose we run value iteration to completion, for a fixed cost function. Then the cost function changes. We would now like to adjust the value function by further iterations of the algorithm, using the previous value function as a starting point.

   (a) Is this a good or a bad idea? Does your answer depend on whether the cost increased or decreased?

   (b) Can you flesh out an algorithm that would be more efficient than simply continuing value iteration after the cost changes? If you can, argue why your algorithm is more efficient. If not, argue why no such algorithm may exist.

3. **Heaven or Hell?** In this exercise, you are asked to extend dynamic programming to an environment with a single hidden state variable. The

environment is a maze with a designated start marked "S," and two possible goal states, both marked "H."



What the agent does not know is which of the two goal states provides a positive reward. One will give $+100$, whereas the other will give -100. There is a .5 probability that either of those situations is true. The cost of moving is -1; the agent can only move into the four directions north, south, east, and west. Once a state labeled "H" has been reached, the play is over.

(a) Implement a value iteration algorithm for this scenario (and ignore the label "X" in the figure). Have your implementation compute the value of the starting state. What is the optimal policy?

(b) Modify your value algorithm to accommodate a probabilistic motion model: with .9 chance the agent moves as desired; with .1 chance it will select any of the three other directions at random. Run your value iteration algorithm again, and compute both the value of the starting state, and the optimal policy.

(c) Now suppose the location labeled "X" contains a sign that informs the agent of the correct assignment of rewards to the two states labeled "H." How does this affect the optimal policy?

(d) How can you modify your value iteration algorithm to find the optimal policy? Be concise. State any modifications to the space over which the value function is defined.

(e) Implement your modification, and compute both the value of the starting state and the optimal policy.