PROBABILISTIC ROBOTICS

Sebastian THRUN
Stanford University
Stanford, CA

■

Wolfram BURGARD
University of Freiburg
Freiburg, Germany

■

Dieter FOX
University of Washington
Seattle, WA

CONTENTS

1

INTRODUCTION

1.1 UNCERTAINTY IN ROBOTICS

Robotics is the science of perceiving and manipulating the physical world through computer-controlled mechanical devices. Examples of successful robotic systems include mobile platforms for planetary exploration [], robotics arms in assembly lines [], cars that travel autonomously on highways [], actuated arms that assist surgeons []. Robotics systems have in common that they are are situated in the physical world, perceive their environments through sensors, and manipulate their environment through things that move.

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 disasters sites like Chernobyl? Wouldn't it be great if our homes were populated by intelligent service robots that would carry out such tedious tasks as loading the dishwasher, and vacuuming the carpet, or walking our dogs? And lastly, a better understanding of robotics will ultimately lead to a better understanding of animals and people.

Tomorrows application domains differ from yesterdays, such as manipulators in assembly lines that carry out the identical task day-in day-out. The most striking characteristic of the new robot systems is that they operate in increasingly unstructured environments, environments that are inherently unpredictable. An assembly line is orders of magnitude more predictable and controllable than a private home. As a result, robotics is moving into areas where sensor input becomes increasingly important, and where robot software has to be robust enough to cope with a range of situations—often too many to anticipate them all. Robotics, thus, is increasingly becoming a software

science, where the goal is to develop robust software that enables robots to withstand the numerous challenges arising in unstructured and dynamic environments.

This book focuses on a key element of robotics: *Uncertainty*. Uncertainty arises if the robot lacks critical information for carrying out its task. It arises from five different factors:

1. **Environments.** Physical worlds are inherently unpredictable. While the degree of uncertainty in well-structured environments such assembly lines is small, environments such as highways and private homes are highly dynamic and unpredictable.

2. **Sensors.** Sensors are inherently limited in what they can perceive. Limitations arise from two primary factors. First, range and resolution of a sensor is subject to physical laws. For example, Cameras can't see through walls, and even within the perceptual range the spatial resolution of camera images is limited. Second, sensors are subject to noise, which perturbs sensor measurements in unpredictable ways and hence limits the information that can be extracted from sensor measurements.

3. **Robots.** Robot actuation involves motors that are, at least to some extent, unpredictable, due effects like control noise and wear-and-tear. Some actuators, such as heavy-duty industrial robot arms, are quite accurate. Others, like low-cost mobile robots, can be extremely inaccurate.

4. **Models.** Models are inherently inaccurate. 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 largely been ignored in robotics, despite the fact that most robotic models used in state-or-the-art robotics systems are rather crude.

5. **Computation.** Robots are real-time systems, which limits the amount of computation that can be carried out. Many state-of-the-art algorithms (such as most of the algorithms described in this book) are approximate, achieving timely response through sacrificing accuracy.

All of these factors give rise to uncertainty. Traditionally, such uncertainty has mostly been ignored in robotics. However, as robots are moving away from factory floors into increasingly unstructured environments, the ability to cope with uncertainty is critical for building successful robots.

1.2 PROBABILISTIC ROBOTICS

This book provides a comprehensive overview of probabilistic algorithms for robotics. Probabilistic robotics is a new approach to robotics that pays tribute to the uncertainty in robot perception and action. They key idea of 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 in the world, probabilistic algorithms represent information by probability distributions over a whole space of possible hypotheses. By doing so, they can represent ambiguity and degree of belief in a mathematically sound way, enabling them to accommodate all sources of uncertainty listed above. Moreover, by basing control decisions on probabilistic information, these algorithms degrade nicely in the face of the various sources of uncertainty described above, leading to new solutions to hard robotics problems.

Let us illustrate the probabilistic approach with a motivating example: mobile robot localization. Localization is the problem of estimating a robot's coordinates in an external reference frame from sensor data, using a map of the environment. Figure 1.1 illustrates the probabilistic approach to mobile robot localization. The specific localization problem studied here is known as *global localization*, where a robot is placed somewhere in the environment and has to localize itself from scratch. In the probabilistic paradigm, the robot's momentary estimate (also called *belief*) is represented by a probability density function over the space of all locations. This is illustrated in the first diagram in Figure 1.1, which shows a uniform distribution (the *prior*) that corresponds to maximum uncertainty. Suppose the robot takes a first sensor measurement and observes that it is next to a door. The resulting belief, shown in the second diagram in Figure 1.1, places high probability at places next to doors and low probability elsewhere. Notice that this distribution possesses three peaks, each corresponding to one of the (indistinguishable) doors in the environment. Furthermore, the resulting distribution assigns high probability to three distinct locations, illustrating that the probabilistic framework can handle multiple, conflicting hypotheses that naturally arise in ambiguous situations. Finally, even non-door locations possess non-zero probability. This is accounted by the uncertainty inherent in sensing: With a small, non-zero probability, the robot might err and actually not be next to a door. Now suppose the robot moves. The third diagram in Figure 1.1 shows the effect of robot motion on its belief, assuming that the robot moved as indicated. The belief is shifted in the direction of motion. It is also smoothed, to account for the inherent uncertainty in robot motion. Finally, the fourth and last diagram 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.
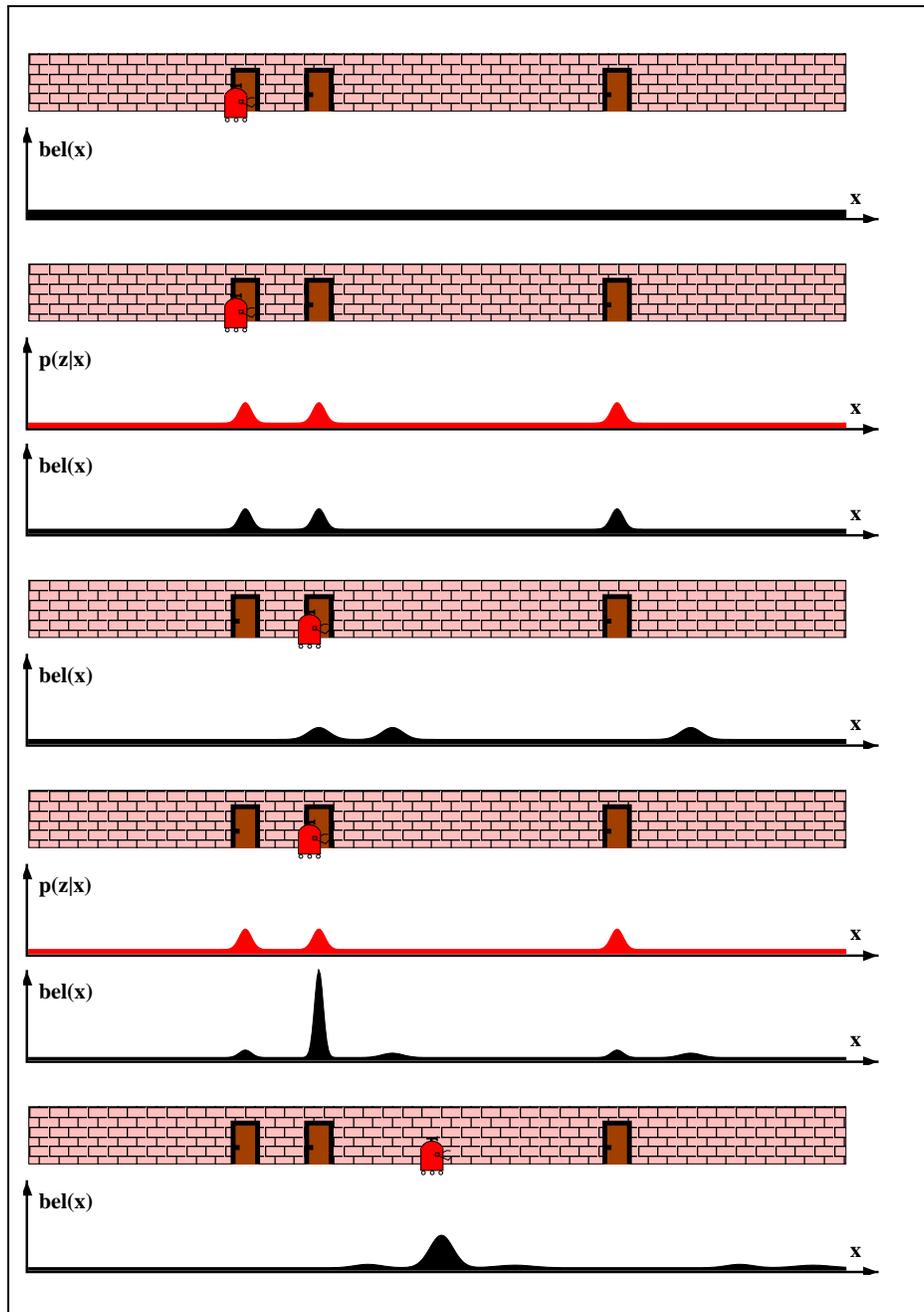
Figure 1.1 The basic idea of Markov localization: A mobile robot during global localization.

This example illustrates the probabilistic paradigm in the context of a specific perceptual problem. 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. Similarly, when selecting actions, probabilistic approaches considers the full uncertainty, not just the most likely guess. By doing so, the probabilistic approach trades off information gathering (exploration) and exploitation, and act optimally relative to the state of knowledge.

1.3   IMPLICATIONS

What are the advantages of programming robots probabilistically, when compared to other approaches that do not represent uncertainty explicitly? Our central conjecture is nothing less than the following:

> *A robot that carries a notion of its own uncertainty and that acts accordingly is superior to one that does not.*

In particular, probabilistic approaches are typically more robust in the face of sensor limitations, sensor noise, environment dynamics, and so on. They often scale much better to complex and unstructured environments, where the ability to handle 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 *kidnapped robot problem*, in which a mobile robot must recover from localization failure; or the problem of building accurate maps of very large environments, in the absence of a global positioning device such as GPS. Additionally, probabilistic algorithms make much weaker requirements on the accuracy of models than many classical planning algorithms do, thereby relieving the programmer from the unsurmountable burden to come up with accurate models. 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. And finally, probabilistic algorithms are broadly applicable to virtually every problem involving perception and action in the real world.

However, these advantages come at a price. Traditionally, the two most frequently cited limitations of probabilistic algorithms are *computational inefficiency*, and *a need to approximate*. Probabilistic algorithms are inherently less efficient than nonprobabilistic ones, due to the fact that they consider entire probability densities. The need to approximate arises from the fact that most robot worlds are continuous. Computing exact posterior distributions is typically infeasible, since distributions over the

continuum possess infinitely many dimensions. Sometimes, one is fortunate in that the uncertainty can approximated tightly with a compact parametric model (e.g., discrete distributions or Gaussians); in other cases, such approximations are too crude and more complicated representations most be employed. Recent research has successfully led to a range of are computationally efficient probabilistic algorithms, for a range of hard robotics problems—many of which are described in depth in this book.

1.4 ROAD MAP

This book attempts to provide a comprehensive and in-depth introduction into probabilistic robotics. The choice of material is somewhat biased towards research carried out at Carnegie Mellon University, the University of Bonn, and affiliated labs. However, we have attempted to include in-depth descriptions of other, important probabilistic algorithms. The algorithms described here have been developed for mobile robots; however, many of them are equally applicable to other types of robots. Thus, the coverage of the material is by no means complete; probabilistic ideas have recently become extremely popular in robotics, and a complete description of the field would simply not fit into a single book. However, we believe that the choice of material is representative for the existing body of literature.

The goal of the book is to provide a systematic introduction into the probabilistic paradigm, from the underlying mathematical framework to implementation. For each major algorithm, this book provides

- a complete mathematical derivation,

- pseudo-code in a C-like language,

- discussions of implementation details and potential pitfalls, and

- empirical results obtained in fielded systems.

We believe that all four items are essential for obtaining a deep understanding of the probabilistic paradigm. At the end of each chapter, the book also provides bibliographical notes and a list of questions and exercises.

The book has been written with researchers, graduate students or advanced undergraduate students in mind, specializing in robotics or applied statistics. We have attempted to present the material in a way that requires a minimum of background knowledge.

However, basic knowledge of probability theory will almost certainly help in understanding the material. The various mathematical derivations can easily be skipped at first reading. However, we strongly recommend to take the time and study the mathematical derivations, as a profound mathematical understanding is will almost certainly lead to deep and important insights into the working of the probabilistic approach to robotics.

If used in the classroom, each chapter should be covered in one or two lectures; however, we recommend that the study of the book be accompanied by practical, hands-on experimentation as directed by the questions and exercises at the end of each chapter.

This book is organized in four major parts.

- The first part, Chapters 2 through 5, discuss the basic mathematical framework that underlies all of the algorithms described in this book. Chapters 2 through 4 introduce the basic probabilistic notation and describes a collection of filters for probabilistic state estimation. Chapter 5 discusses specific probabilistic models that characterize mobile robot perception and motion.

- The second part, which comprised Chapters 7 to **??**, describes a range of perceptual algorithms, which map sensor measurements into internal robot beliefs. In particular, Chapter 7 describes algorihtms for mobile robot localization, followed by algorithms for map acquisition described in Chapters **??**. This part also contains a chapter on learning models.

- The third part, in Chapters **??** to **??**, introduces probabilistic planning and action selection algorihtms.

- Finally, Chapter **??** describes two robot systems that were controlled by probabilistic algorithms. These robots were deployed in museums as interactive tour-guide robots, where they managed to navigate reliably without the need to modify the museums in any way.

The book is best read in order, from the beginning to the end. However, we have attempted to make each individual chapter self-explanatory.

1.5   BIBLIOGRAPHICAL REMARKS

The term 'robot' was invented in 1921 by the Czech novelist Karel Čapek [42], to describe a willing, intelligent and human-like machines that make life pleasant by doing the type work we don't like to do. In the Fourties, Asimov coined the term 'robotics' and postulated the famous three laws of robotics [1, 2],

Robotics, as a scientific discipline has been an active field of research foer several decades. In the early years, most of the research focused on

Major trends in robotics:

- Seventies: classical (deliberate) approach, accurate models, no uncertainty, no sensing. Still: very hard problem. Reif: planning problem NP hard, but only doubly exponential algorithms known. Canny: first single exponential planning algorithm. Latombe: Many impressive randomized planning algorithms. Important difference: randomization used for search, not for representing uncertainty. Planning algorithm for special topologies: Schwartz, and Sharir. Some take sensor data into account. Koditschek navigation function for feedback control, generalizes Khatib's potential fields, which suffer from local minima (navigation functions don't).

- Mid-Eighties: reactive approach, rejection of models, pure reliance on sensors, uncertainty (if any) handled by feedback control. Relies on sensor data carrying sufficient information for action selection, therefore uncertainty not an issue. Typically confined to small environments, where everything of importance can be perceived by the sensors. Nevertheless, some impressive results for control of legged robots. Brooks, Mataric, Steels, Connell, and many others. Biologically inspired robotics. Smithers.

- Mid-Nineties: Hybrid approaches, reactive at low level (fast decision cycle), deliberate at higher levels. Uses sensing mostly at low level, and models at high level. Gat, Arkins, Balch, Firby, Simmons and many others. harvests best of both worlds: robust through reaction, but can do more powerful tasks due to deliberation.

- Late Ninties: Probabilistic robotics, different way to integrate models and sensor data.

Probabilistic robotics can be traced back to Sixties to advent of Kalman filters, which has been used extensively in robotics. First serious advance is Smith and Cheeseman in mid-80s, who proppose an algorithm for concurrent mapping and localization based on Kalman filters that is now in widespread use. Durrant-Whyte, Leonard, Castellanos, and many others.

Main activity in past five years. Main advances: more flexible representations (beyond Kalman filters), more efficient algorithms. Statistical approaches for solving hard correspondence problems.

What is new in probabilistic robotics, relative to approaches above?

- seamlessly blends models and perception in a novel way
- sound mathematical theory, clear assumptions, therefore it's easier to predict failure modes
- applies to all levels: lowest to highest, since uncertainty arises everywhere
- currently the best known solutions in a range of hard robotics problems

2

RECURSIVE STATE ESTIMATION

2.1 INTRODUCTION

At the core of probabilistic robotics is the idea of estimating state from sensor data. State estimation addresses the problem of estimating quantities from sensor data that are not directly observable, but that can be inferred. In most robotic applications, determining what to do is relatively easy if one only knew *certain* quantities. For example, moving a mobile robot is relatively easy if the exact location of the robot and all nearby obstacles are known. Unfortunately, these variables are not directly measurable. Instead, a robot has to rely on its sensors to gather this information. Sensors carry only partial information about those quantities, and their measurements are corrupted by noise. State estimation seeks to recover state variables from the data. Probabilistic state estimation algorithms compute belief distributions over possible world states. An example of probabilistic state estimation was already encountered in the introduction to this book: mobile robot localization.

The goal of this chapter is to introduce the basic vocabulary and mathematical tools for estimating state from sensor data.

- Section 2.2 introduces basic probabilistic concepts and notations used throughout the book.

- Section 2.3 describes our formal model of robot environment interaction, setting forth some of the key terminology used throughout the book.

- Section 2.4 introduces *Bayes filters*, the recursive algorithm for state estimation that forms the basis of virtually every technique presented in this book.

9

■   Section 2.5 discusses representational and computational issues that arise when implementing Bayes filters.

2.2   BASIC CONCEPTS IN PROBABILITY

This section familiarizes the reader with the basic notation and probabilistic facts and notation used throughout the book. In probabilistic robotics, quantities such as sensor measurements, controls, and the states a robot and its environment might assume are all modeled as random variables. Random variables can take on multiple values, and they do so according to specific probabilistic laws. Probabilistic inference is the process of calculating these laws for random variables that are derived from other random variables, such as those modeling sensor data.

Let X denote a random variable and x denote a specific event that X might take on. A standard example of a random variable is that of a coin flip, where X can take on the values head or tail. If the space of all values that X can take on is discrete, as is the case if X is the outcome of a coin flip, we write

$$p(X = x) \tag{2.1}$$

to denote the probability that the random variable X has value x. For example, a fair coin is characterized by $p(X = \text{head}) = p(X = \text{tail}) = \frac{1}{2}$. Discrete probabilities sum to one, that is,

$$\sum_x p(X = x) \quad = \quad 1 \,, \tag{2.2}$$

and of course, probabilities are always non-negative, that is, $p(X = x) \geq 0$. To simplify the notation, we will usually omit explicit mention of the random variable whenever possible, and instead use the common abbreviation $p(x)$ instead of writing $p(X = x)$.

Most techniques in this book address estimation and decision making in continuous spaces. Continuous spaces are characterized by random variables that can take on a continuum of values. Throughout this book, we assume that all continuous random variables possess probability density functions (PDFs). A common density function is that of the one-dimensional *normal distribution* with mean μ and variance σ^2. This

distribution is given by the following Gaussian function:

$$p(x) \quad = \quad \left(2\pi\sigma^2\right)^{-\frac{1}{2}} \exp\left\{-\tfrac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right\} \tag{2.3}$$

Normal distributions play a major role in this book. We will frequently abbreviate them as $\mathcal{N}(x; \mu, \sigma^2)$, which specifies the random variable, its mean, and its variance.

The Normal distribution (2.3) assumes that x is a scalar value. Often, x will be a multi-dimensional vector. Normal distributions over vectors are called *multivariate*. Multivariate normal distributions are characterized by density functions of the following form:

$$p(x) \quad = \quad \det\left(2\pi\Sigma\right)^{-\frac{1}{2}} \exp\left\{-\tfrac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right\} \tag{2.4}$$

Here μ is the mean vector and Σ a (positive semidefinite) symmetric matrix called *covariance matrix*. The superscript ^{T} marks the transpose of a vector. The reader should take a moment to realize that Equation (2.4) is a strict generalization of Equation (2.3); both definitions are equivalent if x is a scalar value. The PDFs of a one- and a two-dimensional normal distribution are graphically depicted in Figure 5.6.

Equations (2.3) and (2.4) are examples of PDFs. Just as discrete probability distributions always sums up to one, a PDF always integrates to 1:

$$\int p(x)\, dx \quad = \quad 1\,. \tag{2.5}$$

However, unlike a discrete probability, the value of a PDF is not bounded above by 1. Throughout this book, we will use the terms *probability*, *probability density* and *probability density function* interchangeably. We will silently assume that all continuous random variables are measurable, and we also assume that all continuous distributions actually possess densities.

The *joint distribution* of two random variables X and Y is given by

$$p(x, y) \quad = \quad p(X = x \text{ and } Y = y)\,. \tag{2.6}$$

This expression describes the probability of the event that the random variable X takes on the value x *and* that Y takes on the value y. If X and Y are *independent*, we have

$$p(x, y) \quad = \quad p(x)\, p(y)\,. \tag{2.7}$$

Often, random variables carry information about other random variables. Suppose we already know that Y's value is y, and we would like to know the probability that X's value is x conditioned on that fact. Such a probability will be denoted

$$p(x \mid y) \quad = \quad p(X = x \mid Y = y) \tag{2.8}$$

and is called *conditional* probability. If $p(y) > 0$, then the conditional probability is defined as

$$p(x \mid y) \quad = \quad \frac{p(x, y)}{p(y)}\,. \tag{2.9}$$

If X and Y are independent, we have

$$p(x \mid y) \quad = \quad \frac{p(x)\, p(y)}{p(y)} \quad = \quad p(x)\,. \tag{2.10}$$

In other words, if X and Y are independent, Y tells us nothing about the value of X. There is no advantage of knowing Y if our interest pertains to knowing X. Independence, and its generalization known as conditional independence, plays a major role throughout this book.

An interesting fact, which follows from the definition of conditional probability and the axioms of probability measures, is often referred to as *theorem of total probability*:

$$p(x) \quad = \quad \sum_y p(x \mid y)\, p(y) \qquad \text{(discrete case)} \tag{2.11}$$

$$p(x) \quad = \quad \int p(x \mid y)\, p(y)\, dy \qquad \text{(continuous case)} \tag{2.12}$$

If $p(x \mid y)$ or $p(y)$ are zero, we define the product $p(x \mid y)\, p(y)$ to be zero, regardless of the value of the remaining factor.

Equally important is *Bayes rule*, which relates conditionals of the type $p(x \mid y)$ to their "inverse," $p(y \mid x)$. The rule, as stated here, requires $p(y) > 0$:

$$p(x \mid y) \;=\; \frac{p(y \mid x)\, p(x)}{p(y)} \;=\; \frac{p(y \mid x)\, p(x)}{\sum_{x'} p(y \mid x')\, p(x')} \quad \text{(discrete)} \qquad (2.13)$$

$$p(x \mid y) \;=\; \frac{p(y \mid x)\, p(x)}{p(y)} \;=\; \frac{p(y \mid x)\, p(x)}{\int p(y \mid x')\, p(x')\, dx'} \quad \text{(continuous)} \quad (2.14)$$

Bayes rule plays a predominant role in probabilistic robotics. If x is a quantity that we would like to infer from y, the probability $p(x)$ will be referred to as *prior probability distribution*, and y is called the *data* (e.g., a sensor measurement). The distribution $p(x)$ summarizes the knowledge we have regarding X prior to incorporating the data y. The probability $p(x \mid y)$ is called the *posterior probability distribution* over X. As (2.14) suggests, Bayes rule provides a convenient way to compute a posterior $p(x \mid y)$ using the "inverse" conditional probability $p(y \mid x)$ along with the prior probability $p(x)$. In other words, if we are interested in inferring a quantity x from sensor data y, Bayes rule allows us to do so through the inverse probability, which specifies the probability of data y assuming that x was the case. In robotics, this inverse probability is often coined "generative model," since it describes, at some level of abstraction, how state variables X *cause* sensor measurements Y.

An important observation is that the denominator of Bayes rule, $p(y)$, does not depend on x. Thus, the factor $p(y)^{-1}$ in Equations (2.13) and (2.14) will be the same for any value x in the posterior $p(x \mid y)$. For this reason, $p(y)^{-1}$ is often written as a *normalizer* variable, and generically denoted η:

$$p(x \mid y) \;=\; \eta\, p(y \mid x)\, p(x)\,. \qquad (2.15)$$

If X is discrete, equations of this type can be computed as follows:

$$\forall x : \mathrm{aux}_{x|y} \;=\; p(y \mid x)\, p(x) \qquad (2.16)$$

$$\mathrm{aux}_y \;=\; \sum_x \mathrm{aux}_{x|y} \qquad (2.17)$$

$$\forall x : p(x \mid y) \;=\; \frac{\mathrm{aux}_{x|y}}{\mathrm{aux}_y}\,, \qquad (2.18)$$

where $\mathrm{aux}_{x|y}$ and aux_y are auxiliary variables. These instructions effectively calculate $p(x \mid y)$, but instead of explicitly computing $p(y)$, they instead just normalize the

result. The advantage of the notation in (2.15) lies in its brevity. Instead of explicitly providing the exact formula for a normalization constant—which can grow large very quickly in some of the mathematical derivations to follow—we simply will use the normalizer "η" to indicate that the final result has to be normalized to 1. Throughout this book, normalizers of this type will be denoted η (or η', η'', ...). We will freely use the same η in different equations to denote normalizers, even if their actual values are different.

The *expectation* of a random variable X is given by

$$
\begin{aligned}
E[X] &= \sum_x x\, p(x)\,, \\
E[X] &= \int x\, p(x)\, dx\,.
\end{aligned}
\tag{2.19}
$$

Not all random variables possess finite expectations; however, those that do not are of no relevance to the material presented in this book. The expectation is a linear function of a random variable. In particular, we have

$$
E[aX + b] = aE[X] + b
\tag{2.20}
$$

for arbitrary numerical values a and b. The covariance of X is obtained as follows

$$
\mathrm{Cov}[X] = E[X - E[X]]^2 = E[X^2] - E[X]^2
\tag{2.21}
$$

The covariance measures the squared expected deviation from the mean. As stated above, the mean of a multivariate normal distribution $\mathcal{N}(x; \mu, \Sigma)$ is μ, and its covariance is Σ.

Another important characteristic of a random variable is its *entropy*. For discrete random variables, the entropy is given by the following expression:

$$
H(P) = E[-\log_2 p(x)] = -\sum_x p(x)\, \log_2 p(x)\,.
\tag{2.22}
$$

The concept of entropy originates in information theory. The entropy is the expected information that the value of x carries: $-\log_2 p(x)$ is the number of bits required

to encode x using an optimal encoding, and $p(x)$ is the probability at which x will be observed. In this book, entropy will be used in robotic information gathering, to express the information a robot may receive upon executing specific actions.

Finally, we notice that it is perfectly fine to condition any of the rules discussed so far on arbitrary other random variables, such as the variable Z. For example, conditioning Bayes rule on $Z = z$ gives us:

$$p(x \mid y, z) \quad = \quad \frac{p(y \mid x, z) \, p(x \mid z)}{p(y \mid z)} \tag{2.23}$$

Similarly, we can condition the rule for combining probabilities of independent random variables (2.7) on other variables z:

$$p(x, y \mid z) \quad = \quad p(x \mid z) \, p(y \mid z) \,. \tag{2.24}$$

Such a relation is known as *conditional independence*. As the reader easily verifies, (2.24) is equivalent to

$$\begin{aligned} p(x \mid z) &= p(x \mid z, y) \\ p(y \mid z) &= p(y \mid z, x) \end{aligned} \tag{2.25}$$

Conditional independence plays an important role in probabilistic robotics. It applies whenever a variable y carries no information about a variable x if another variable's value z is known. Conditional independence does not imply (absolute) independence, that is,

$$p(x, y \mid z) = p(x \mid z) \, p(y \mid z) \quad \not\Rightarrow \quad p(x, y) = p(x) \, p(y) \tag{2.26}$$

The converse is also in general untrue: absolute independence does not imply conditional independence:

$$p(x, y) = p(x) \, p(y) \quad \not\Rightarrow \quad p(x, y \mid z) = p(x \mid z) \, p(y \mid z) \tag{2.27}$$

In special cases, however, conditional and absolute independence may coincide.
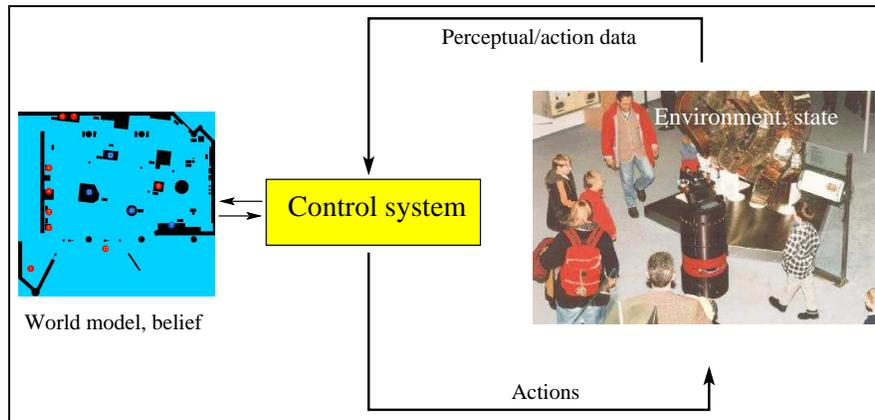
Figure 2.1 Robot Environment Interaction.

2.3 ROBOT ENVIRONMENT INTERACTION

Figure 2.1 illustrates the interaction of a robot with its environment. The *environment*, or *world*, of a robot is a dynamical system that possesses internal state. The robot can acquire information about its environment using its sensors. However, sensors are noisy, and there are usually many things that cannot be sensed directly. As a consequence, the robot maintains an internal belief with regards to the state of its environment, depicted on the left in this figure. The robot can also influence its environment through its actuators. However, the effect of doing so is often somewhat unpredictable. This interaction will now be described more formally.

2.3.1 State

Environments are characterized by *state*. For the material presented in this book, it will be convenient to think of state as the collection of all aspects of the robot and its environment that can impact the future. State may change over time, such as the location of people; or it may remain static throughout the robot's operation, such as the location of walls in (most) buildings. State that changes will be called *dynamic state*, which distinguishes it from *static*, or non-changing state. The state also includes variables regarding the robot itself, such as its pose, velocity, whether or not its sensors are functioning correctly, and so on. Throughout this book, state will be denoted x; although the specific variables included in x will depend on the context. The state at time t will be denoted x_t. Typical state variables used throughout this book are:

- The robot pose, which comprises its location and orientation relative to a global coordinate frame. Rigid mobile robots possess six such state variables, three for their Cartesian coordinates, and three for their angular orientation, also called Euler angles (pitch, roll, and yaw). For rigid mobile robots confined to planar environments, the pose is usually given by three variables, its two location coordinates in the plane and its heading direction (yaw). The robot pose is often referred to as *kinematic state*.

- The configuration of the robot's actuators, such as the joints of robotic manipulators. Each degree of freedom in a robot arm is characterized by a one-dimensional configuration at any point in time, which is part of the kinematic state of the robot.

- The robot velocity and the velocities of its joints. A rigid robot moving through space is characterized by up to six velocity variables, one for each pose variables. Velocities are commonly referred to as *dynamic state*. Dynamic state will play only a minor role in this book.

- The location and features of surrounding objects in the environment. An object may be a tree, a wall, or a pixel within a larger surface. Features of such objects may be their visual appearance (color, texture). Depending on the granularity of the state that is being modeled, robot environments possess between a few dozen and up to hundreds of billions of state variables (and more). Just imagine how many bits it will take to accurately describe your physical environment! For many of the problems studied in this book, the location of objects in the environment will be static. In some problems, objects will assume the form of *landmarks*, which are distinct, stationary features of the environment that can be recognized reliably.

- The location and velocities of moving objects and people. Often, the robot is not the only moving actor in its environment. Other moving entities possess their own kinematic and dynamic state.

- There can be a huge number of other state variables. For example, whether or not a sensor is broken is a state variable, as is the level of battery charge for a battery-powered robot.

A state x_t will be called *complete* if it is the best predictor of the future. Put differently, completeness entails that knowledge of past states, measurements, or controls carry no additional information that would help us to predict the future more accurately. It it important to notice that our definition of completeness does not require the future to be a *deterministic* function of the state. The future may be stochastic, but no variables prior to x_t may influence the stochastic evolution of future states, unless

this dependence is mediated through the state x_t. Temporal processes that meet these conditions are commonly known as *Markov chains*.

The notion of state completeness is mostly of theoretical importance. In practice, it is impossible to specify a complete state for any realistic robot system. A complete state includes not just all aspects of the environment that may have an impact on the future, but also the robot itself, the content of its computer memory, the brain dumps of surrounding people, etc. Those are hard to obtain. Practical implementations therefore single out a small subset of all state variables, such as the ones listed above. Such a state is called *incomplete state*.

In most robotics applications, the state is continuous, meaning that x_t is defined over a continuum. A good example of a continuous state space is that of a robot pose, that is, its location and orientation relative to an external coordinate system. Sometimes, the state is discrete. An example of a discrete state space is the (binary) state variable that models whether or not a sensor is broken. State spaces that contain both continuous and discrete variables are called *hybrid* state spaces.

In most cases of interesting robotics problems, state changes over time. Time, throughout this book, will be discrete, that is, all interesting events will take place at discrete time steps

$$0, 1, 2, \ldots . \tag{2.28}$$

If the robot starts its operation at a distinct point in time, we will denote this time as $t = 0$.

2.3.2   Environment Interaction

There are two fundamental types of interactions between a robot and its environment: The robot can influence the state of its environment through its actuators. And it can gather information about the state through its sensors. Both types of interactions may co-occur, but for didactic reasons we will distinguish them throughout this book. The interaction is illustrated in Figure 2.1:

- **Sensor measurements**. Perception is the process by which the robot uses its sensors to obtain information about the state of its environment. For example, a robot might take a camera image, a range scan, or query its tactile sensors to receive information about the state of the environment. The result of such a

perceptual interaction will be called a *measurement*, although we will sometimes also call it *observation* or *percept*. Typically, sensor measurements arrive with some delay. Hence they provide information about the state a few moments ago.

- **Control actions** change the state of the world. They do so by actively asserting forces on the robot's environment. Examples of control actions include robot motion and the manipulation of objects. Even if the robot does not perform any action itself, state usually changes. Thus, for consistency, we will assume that the robot *always* executes a control action, even if it chooses not to move any of its motors. In practice, the robot continuously executes controls and measurements are made concurrently.

Hypothetically, a robot may keep a record of all past sensor measurements and control actions. We will refer to such a the collection as the *data* (regardless of whether they are being memorized). In accordance with the two types of environment interactions, the robot has access to two different data streams.

- **Measurement data** provides information about a momentary state of the environment. Examples of measurement data include camera images, range scans, and so on. For most parts, we will simply ignore small timing effects (e.g., most ladar sensors scan environments sequentially at very high speeds, but we will simply assume the measurement corresponds to a specific point in time). The measurement data at time t will be denoted

$$z_t \tag{2.29}$$

Throughout most of this book, we simply assume that the robot takes exactly one measurement at a time. This assumption is mostly for notational convenience, as nearly all algorithms in this book can easily be extended to robots that can acquire variables numbers of measurements within a single time step. The notation

$$z_{t_1:t_2} \quad = \quad z_{t_1}, z_{t_1+1}, z_{t_1+2}, \dots, z_{t_2} \tag{2.30}$$

denotes the set of all measurements acquired from time t_1 to time t_2, for $t_1 \leq t_2$.

- **Control data** carry information about the *change of state* in the environment. In mobile robotics, a typical example of control data is the velocity of a robot. Setting the velocity to 10 cm per second for the duration of five seconds suggests that the robot's pose, after executing this motion command, is approximately 50 cm ahead of its pose before command execution. Thus, its main information regards the change of state.

An alternative source of control data are *odometers*. Odometers are sensors that measure the revolution of a robot's wheels. As such they convey information about the change of the state. Even though odometers are sensors, we will treat odometry as control data, since its main information regards the change of the robot's pose.

Control data will be denoted u_t. The variable u_t will always correspond to the change of state in the time interval $(t-1;t]$. As before, we will denote sequences of control data by $u_{t_1:t_2}$, for $t_1 \leq t_2$:

$$u_{t_1:t_2} \quad = \quad u_{t_1}, u_{t_1+1}, u_{t_1+2}, \ldots, u_{t_2} \;. \tag{2.31}$$

Since the environment may change even if a robot does not execute a specific control action, the fact that time passed by constitutes, technically speaking, control information. Hence, we assume that there is exactly one control data item per time step t.

The distinction between measurement and control is a crucial one, as both types of data play fundamentally different roles in the material yet to come. Perception provides information about the environment's state, hence it tends to increase the robot's knowledge. Motion, on the other hand, tends to induce a loss of knowledge due to the inherent noise in robot actuation and the stochasticity of robot environments; although sometimes a control makes the robot more certain about the state. By no means is our distinction intended to suggest that actions and perceptions are separated in time, i.e., that the robot does not move while taking sensor measurements. Rather, perception and control takes place concurrently; many sensors affect the environment; and the separation is strictly for convenience.

2.3.3   Probabilistic Generative Laws

The evolution of state and measurements is governed by probabilistic laws. In general, the state at time x_t is generated stochastically. Thus, it makes sense to specify the probability distribution from which x_t is generated. At first glance, the emergence of state x_t might be conditioned on all past states, measurements, and controls. Hence, the probabilistic law characterizing the evolution of state might be given by a probability distribution of the following form:

$$p(x_t \mid x_{0:t-1}, z_{1:t-1}, u_{1:t}) \tag{2.32}$$

(Notice that through no particular motivation we assume here that the robot executes a control action u_1 first, and then takes a measurement z_1.) However, if the state x is

complete then it is a sufficient summary of all that happened in previous time steps. In particular, x_{t-1} is a sufficient statistic of all previous controls and measurements up to this point, that is, $u_{1:t-1}$ and $z_{1:t-1}$. From all the variables in the expression above, only the control u_t matters if we know the state x_{t-1}. In probabilistic terms, this insight is expressed by the following equality:

$$p(x_t \mid x_{0:t-1}, z_{1:t-1}, u_{1:t}) \quad = \quad p(x_t \mid x_{t-1}, u_t) \tag{2.33}$$

The property expressed by this equality is an example of *conditional independence*. It states that certain variables are independent of others if one knows the values of a third group of variables, the conditioning variables. Conditional independence will be exploited pervasively in this book, as it is the main source of tractability of probabilistic robotics algorithms.

Similarly, one might want to model the process by which measurements are being generated. Again, if x_t is complete, we have an important conditional independence:

$$p(z_t \mid x_{0:t}, z_{1:t-1}, u_{1:t}) \quad = \quad p(z_t \mid x_t) \tag{2.34}$$

In other words, the state x_t is sufficient to predict the (potentially noisy) measurement z_t. Knowledge of any other variable, such as past measurements, controls or even past states, is irrelevant if x_t is complete.

This discussion leaves open as to what the two resulting conditional probabilities are: $p(x_t \mid x_{t-1}, u_t)$ and $p(z_t \mid x_t)$. The probability $p(x_t \mid x_{t-1}, u_t)$ is the *state transition probability*. It specifies how environmental state evolves over time as a function of robot controls u_t. Robot environments are stochastic, which is reflected by the fact that $p(x_t \mid x_{t-1}, u_t)$ is a probability distribution, not a deterministic function. Sometimes the state transition distribution does not depend on the time index t, in which case we may write it as $p(x' \mid u, x)$, where x' is the successor and x the predecessor state.

The probability $p(z_t \mid x_t)$ is called the *measurement probability*. It also may not depend on the time index t, in which case it shall be written as $p(z \mid x)$. The measurement probability specifies the probabilistic law according to which measurements z are generated from the environment state x. Measurements are usually noisy projections of the state.

The state transition probability and the measurement probability together describe the dynamical stochastic system of the robot and its environment. Figure **??** illustrates the evolution of states and measurements, defined through those probabilities. The

state at time t is stochastically dependent on the state at time $t - 1$ and the control u_t. The measurement z_t depends stochastically on the state at time t. Such a temporal generative model is also known as hidden Markov model (HMM) or dynamic Bayes network (DBN). To specify the model fully, we also need an initial state distribution $p(x_0)$.

2.3.4   Belief Distributions

Another key concept in probabilistic robotics is that of a *belief*. A belief reflects the robot's internal knowledge about the state of the environment. We already discussed that state cannot be measured directly. For example, a robot's pose might be $x = \langle 14.12, 12.7, 0.755 \rangle$ in some global coordinate system, but it usually cannot know its pose, since poses are not measurable directly (not even with GPS!). Instead, the robot must infer its pose from data. We therefore distinguish the true state from its internal *belief*, or *state of knowledge* with regards to that state.

Probabilistic robotics represents beliefs through conditional probability distributions. A belief distribution assigns a probability (or density value) to each possible hypothesis with regards to the true state. Belief distributions are posterior probabilities over state variables conditioned on the available data. We will denote belief over a state variable x_t by $bel(x_t)$, which is an abbreviation for the posterior

$$bel(x_t) \quad = \quad p(x_t \mid z_{1:t}, u_{1:t}) \,. \tag{2.35}$$

This posterior is the probability distribution over the state x_t at time t, conditioned on all past measurements $z_{1:t}$ and all past controls $u_{1:t}$.

The reader may notice that we silently assume that the belief is taken *after* incorporating the measurement z_t. Occasionally, it will prove useful to calculate a posterior *before* incorporating z_t, just after executing the control u_t. Such a posterior will be denoted as follows:

$$\overline{bel}(x_t) \quad = \quad p(x_t \mid z_{1:t-1}, u_{1:t}) \tag{2.36}$$

This probability distribution is often referred to as *prediction* in the context of probabilistic filtering. This terminology reflects the fact that $\overline{bel}(x_t)$ predicts the state at time t based on the previous state posterior, before incorporating the measurement at time t. Calculating $bel(x_t)$ from $\overline{bel}(x_t)$ is called *correction* or the *measurement update*.

2.4 BAYES FILTERS

2.4.1 The Bayes Filter Algorithm

The most general algorithm for calculating beliefs is given by the *Bayes filter* algorithm. This algorithm calculates the belief distribution bel from measurement and control data. We will first state the basic algorithm and elucidate it with a numerical example. After that, we will derive it mathematically from the assumptions made so far.

Table 2.1 depicts the basic Bayes filter in pseudo-algorithmic form. The Bayes filter is recursive, that is, the belief $bel(x_t)$ at time t is calculated from the belief $bel(x_{t-1})$ at time $t-1$. Its input is the belief bel at time $t-1$, along with the most recent control u_t and the most recent measurement z_t. Its output is the belief $bel(x_t)$ at time t. Table 2.1 only depicts a single step of the Bayes Filter algorithm: the *update rule*. This update rule is applied recursively, to calculate the belief $bel(x_t)$ from the belief $bel(x_{t-1})$, calculated previously.

The Bayes filter algorithm possesses two essential steps. In Line 3, it processes the control u_t. It does so by calculating a belief over the state x_t based on the prior belief over state x_{t-1} and the control u_t. In particular, the belief $\overline{bel}(x_t)$ that the robot assigns to state x_t is obtained by the integral (sum) of the product of two distributions: the prior assigned to x_{t-1}, and the probability that control u_t induces a transition from x_{t-1} to x_t. The reader may recognize the similarity of this update step to Equation (2.12). As noted above, this update step is called the control update, or *prediction*.

The second step of the Bayes filter is called the measurement update. In Line 4, the Bayes filter algorithm multiplies the belief $\overline{bel}(x_t)$ by the probability that the measurement z_t may have been observed. It does so for each hypothetical posterior state x_t. As will become apparent further below when actually deriving the basic filter equations, the resulting product is generally not a probability, that is, it may not integrate to 1. Hence, the result is normalized, by virtue of the normalization constant η. This leads to the final belief $bel(x_t)$, which is returned in Line 6 of the algorithm.

To compute the posterior belief recursively, the algorithm requires an initial belief $bel(x_0)$ at time $t = 0$ as boundary condition. If one knows the value of x_0 with certainty, $bel(x_0)$ should be initialized with a point mass distribution that centers all probability mass on the correct value of x_0, and assigns zero probability anywhere else. If one is entirely ignorant about the initial value x_0, $bel(x_0)$ may be initialized using a uniform distribution over the domain of x_0 (or related distribution from the Dirichlet family of distributions). Partial knowledge of the initial value x_0 can be

```
1:      Algorithm Bayes_filter(bel(x_{t-1}), u_t, z_t):
2:          for all x_t do
3:              \overline{bel}(x_t) = \int p(x_t \mid u_t, x_{t-1}) \, bel(x_{t-1}) \, dx
4:              bel(x_t) = \eta \, p(z_t \mid x_t) \, \overline{bel}(x_t)
5:          endfor
6:          return bel(x_t)
```

Table 2.1   The general algorithm for Bayes filtering.

expressed by non-uniform distributions; however, the two cases of full knowledge and full ignorance are the most common ones in practice.

The algorithm Bayes filter can only be implemented in the form stated here for very simple estimation problems. In particular, we either need to be able to carry out the integration in Line 3 and the multiplication in Line 4 in closed form, or we need to restrict ourselves to finite state spaces, so that the integral in Line 3 becomes a (finite) sum.

2.4.2   Example

Our illustration of the Bayes filter algorithm is based on the scenario in Figure 2.2, which shows a robot estimating the state of a door using its camera. To make this problem simple, let us assume that the door can be in one of two possible states, open or closed, and that only the robot can change the state of the door. Let us furthermore assume that the robot does not know the state of the door initially. Instead, it assigns equal prior probability to the two possible door states:

$$bel(X_0 = \mathbf{open}) \quad = \quad 0.5 \tag{2.37}$$
$$bel(X_0 = \mathbf{closed}) \quad = \quad 0.5 \tag{2.38}$$

Let us furthermore assume the robot's sensors are noisy. The noise is characterized by the following conditional probabilities:

$$p(Z_t = \mathbf{sense\_open} \mid X_t = \mathbf{is\_open}) \quad = \quad 0.6$$
$$p(Z_t = \mathbf{sense\_closed} \mid X_t = \mathbf{is\_open}) \quad = \quad 0.4 \tag{2.39}$$
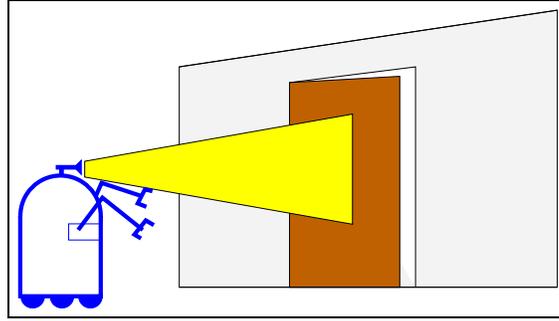
Figure 2.2 A mobile robot estimating the state of a door.

and

$$p(Z_t = \textbf{sense\_open} \mid X_t = \textbf{is\_closed}) \quad = \quad 0.2$$
$$p(Z_t = \textbf{sense\_closed} \mid X_t = \textbf{is\_closed}) \quad = \quad 0.8 \qquad (2.40)$$

These probabilities suggest that the robot's sensors are relatively reliable in detecting a *closed* door, in that the error probability is 0.2. However, when the door is open, it has a 0.4 probability of a false measurement.

Finally, let us assume the robot uses its manipulator to push the door open. If the door is already open, it will remain open. It it is closed, the robot has a 0.8 chance that it will be open afterwards:

$$p(X_t = \textbf{is\_open} \mid U_t = \textbf{push}, X_{t\text{-}1} = \textbf{is\_open}) \quad = \quad 1$$
$$p(X_t = \textbf{is\_closed} \mid U_t = \textbf{push}, X_{t\text{-}1} = \textbf{is\_open}) \quad = \quad 0 \qquad (2.41)$$
$$p(X_t = \textbf{is\_open} \mid U_t = \textbf{push}, X_{t\text{-}1} = \textbf{is\_closed}) \quad = \quad 0.8$$
$$p(X_t = \textbf{is\_closed} \mid U_t = \textbf{push}, X_{t\text{-}1} = \textbf{is\_closed}) \quad = \quad 0.2 \qquad (2.42)$$

It can also choose not to use its manipulator, in which case the state of the world does not change. This is stated by the following conditional probabilities:

$$p(X_t = \textbf{is\_open} \mid U_t = \textbf{do\_nothing}, X_{t\text{-}1} = \textbf{is\_open}) \quad = \quad 1$$
$$p(X_t = \textbf{is\_closed} \mid U_t = \textbf{do\_nothing}, X_{t\text{-}1} = \textbf{is\_open}) \quad = \quad 0 \qquad (2.43)$$
$$p(X_t = \textbf{is\_open} \mid U_t = \textbf{do\_nothing}, X_{t\text{-}1} = \textbf{is\_closed}) \quad = \quad 0$$
$$p(X_t = \textbf{is\_closed} \mid U_t = \textbf{do\_nothing}, X_{t\text{-}1} = \textbf{is\_closed}) \quad = \quad 1 \qquad (2.44)$$

Suppose at time t, the robot takes no control action but it senses an open door. The resulting posterior belief is calculated by the Bayes filter using the prior belief $bel(X_0)$, the control $u_1 = \mathbf{do\_nothing}$, and the measurement $\mathbf{sense\_open}$ as input. Since the state space is finite, the integral in Line 3 turns into a finite sum:

$$
\begin{aligned}
\overline{bel}(x_1) &= \int p(x_1 \mid u_1, x_0)\, bel(x_0)\, dx_0 \\
&= \sum_{x_0} p(x_1 \mid u_1, x_0)\, bel(x_0) \\
&= p(x_1 \mid U_1 = \mathbf{do\_nothing}, X_0 = \mathbf{is\_open})\, bel(X_0 = \mathbf{is\_open}) \\
&\quad + p(x_1 \mid U_1 = \mathbf{do\_nothing}, X_0 = \mathbf{is\_closed})\, bel(X_0 = \mathbf{is\_closed})
\end{aligned}
\tag{2.45}
$$

We can now substitute the two possible values for the state variable X_1. For the hypothesis $X_1 = \mathbf{is\_open}$, we obtain

$$
\begin{aligned}
\overline{bel}&(X_1 = \mathbf{is\_open}) \\
&= p(X_1 = \mathbf{is\_open} \mid U_1 = \mathbf{do\_nothing}, X_0 = \mathbf{is\_open})\, bel(X_0 = \mathbf{is\_open}) \\
&\quad + p(X_1 = \mathbf{is\_open} \mid U_1 = \mathbf{do\_nothing}, X_0 = \mathbf{is\_closed})\, bel(X_0 = \mathbf{is\_closed}) \\
&= 1 \cdot 0.5 + 0 \cdot 0.5 = 0.5
\end{aligned}
\tag{2.46}
$$

Likewise, for $X_1 = \mathbf{is\_closed}$ we get

$$
\begin{aligned}
\overline{bel}&(X_1 = \mathbf{is\_closed}) \\
&= p(X_1 = \mathbf{is\_closed} \mid U_1 = \mathbf{do\_nothing}, X_0 = \mathbf{is\_open})\, bel(X_0 = \mathbf{is\_open}) \\
&\quad + p(X_1 = \mathbf{is\_closed} \mid U_1 = \mathbf{do\_nothing}, X_0 = \mathbf{is\_closed})\, bel(X_0 = \mathbf{is\_closed}) \\
&= 0 \cdot 0.5 + 1 \cdot 0.5 = 0.5
\end{aligned}
\tag{2.47}
$$

The fact that the belief $\overline{bel}(x_1)$ equals our prior belief $bel(x_0)$ should not surprise, as the action $\mathbf{do\_nothing}$ does not affect the state of the world; neither does the world change over time by itself in our example.

Incorporating the measurement, however, changes the belief. Line 4 of the Bayes filter algorithm implies

$$
bel(x_1) = \eta\, p(Z_1 = \mathbf{sense\_open} \mid x_1)\, \overline{bel}(x_1) .
\tag{2.48}
$$

For the two possible cases, $X_1 = \textbf{is\_open}$ and $X_1 = \textbf{is\_closed}$, we get

$$
\begin{aligned}
bel(X_1 &= \textbf{is\_open}) \\
&= \eta \, p(Z_1 = \textbf{sense\_open} \mid X_1 = \textbf{is\_open}) \, \overline{bel}(X_1 = \textbf{is\_open}) \\
&= \eta \, 0.6 \cdot 0.5 \;=\; \eta \, 0.3
\end{aligned}
\tag{2.49}
$$

and

$$
\begin{aligned}
bel(X_1 &= \textbf{is\_closed}) \\
&= \eta \, p(Z_1 = \textbf{sense\_open} \mid X_1 = \textbf{is\_closed}) \, \overline{bel}(X_1 = \textbf{is\_closed}) \\
&= \eta \, 0.2 \cdot 0.5 \;=\; \eta \, 0.1
\end{aligned}
\tag{2.50}
$$

The normalizer η is now easily calculated:

$$
\eta \;=\; (0.3 + 0.1)^{-1} \;=\; 2.5
\tag{2.51}
$$

Hence, we have

$$
\begin{aligned}
bel(X_1 = \textbf{is\_open}) &= 0.75 \\
bel(X_1 = \textbf{is\_closed}) &= 0.25
\end{aligned}
\tag{2.52}
$$

This calculation is now easily iterated for the next time step. As the reader easily verifies, for $u_2 = \textbf{push}$ and $z_2 = \textbf{sense\_open}$ we get

$$
\begin{aligned}
\overline{bel}(X_2 = \textbf{is\_open}) &= 1 \cdot 0.75 + 0.8 \cdot 0.25 = 0.95 \\
\overline{bel}(X_2 = \textbf{is\_closed}) &= 0 \cdot 0.75 + 0.2 \cdot 0.25 = 0.05 \,,
\end{aligned}
\tag{2.53}
$$

and

$$
\begin{aligned}
bel(X_2 = \textbf{is\_open}) &= \eta \, 0.6 \cdot 0.95 \approx 0.983 \\
bel(X_2 = \textbf{is\_closed}) &= \eta \, 0.2 \cdot 0.05 \approx 0.017 \,.
\end{aligned}
\tag{2.54}
$$

At this point, the robot believes that with 0.983 probability the door is open, hence both its measurements were correct. At first glance, this probability may appear to be

sufficiently high to simply accept this hypothesis as the world state and act accordingly. However, such an approach may result in unnecessarily high costs. If mistaking a closed door for an open one incurs costs (e.g., the robot crashes into a door), considering both hypotheses in the decision making process will be essential, as unlikely as one of them may be. Just imagine flying an aircraft on auto pilot with a perceived chance of 0.983 for not crashing!

2.4.3 Mathematical Derivation of the Bayes Filter

The correctness of the Bayes filter algorithm is shown by induction. To do so, we need to show that it correctly calculates the posterior distribution $p(x_t \mid z_{1:t}, u_{1:t})$ from the corresponding posterior one time step earlier, $p(x_{t-1} \mid z_{1:t-1}, u_{1:t-1})$. The correctness follows then by induction under the assumption that we correctly initialized the prior belief $bel(x_0)$ at time $t = 0$.

Our derivation requires that the state x_t is complete, as defined in Section 2.3.1, and it requires that controls are chosen at random. The first step of our derivation involves the application of Bayes rule (2.23) to the target posterior:

$$
\begin{aligned}
p(x_t \mid z_{1:t}, u_{1:t}) &= \frac{p(z_t \mid x_t, z_{1:t-1}, u_{1:t}) \, p(x_t \mid z_{1:t-1}, u_{1:t})}{p(z_t \mid z_{1:t-1}, u_{1:t})} \\
&= \eta \, p(z_t \mid x_t, z_{1:t-1}, u_{1:t}) \, p(x_t \mid z_{1:t-1}, u_{1:t}) \qquad (2.55)
\end{aligned}
$$

We now exploit the assumption that our state is complete. In Section 2.3.1, we defined a state x_t to be complete if no variables prior to x_t may influence the stochastic evolution of future states. In particular, if we (hypothetically) knew the state x_t and were interested in predicting the measurement z_t, no past measurement or control would provide us additional information. In mathematical terms, this is expressed by the following conditional independence:

$$
p(z_t \mid x_t, z_{1:t-1}, u_{1:t}) = p(z_t \mid x_t) . \qquad (2.56)
$$

Such a statement is another example of *conditional independence*. It allows us to simplify (2.55) as follows:

$$
p(x_t \mid z_{1:t}, u_{1:t}) = \eta \, p(z_t \mid x_t) \, p(x_t \mid z_{1:t-1}, u_{1:t}) \qquad (2.57)
$$

and hence

$$bel(x_t) = \eta \, p(z_t \mid x_t) \, \overline{bel}(x_t) \tag{2.58}$$

This equation is implemented in Line 4 of the Bayes filter algorithm in Table 2.1.

Next, we expand the term $\overline{bel}(x_t)$, using (2.12):

$$
\begin{aligned}
\overline{bel}(x_t) &= p(x_t \mid z_{1:t-1}, u_{1:t}) \\
&= \int p(x_t \mid x_{t-1}, z_{1:t-1}, u_{1:t}) \, p(x_{t-1} \mid z_{1:t-1}, u_{1:t}) \, dx_{t-1}
\end{aligned}
\tag{2.59}
$$

Once again, we exploit the assumption that our state is complete. This implies if we know x_{t-1}, past measurements and controls convey no information regarding the state x_t. This gives us

$$p(x_t \mid x_{t-1}, z_{1:t-1}, u_{1:t}) = p(x_t \mid x_{t-1}, u_t) \tag{2.60}$$

Here we retain the control variable u_t, since it does *not* predate the state x_{t-1}. Finally, we note that the control u_t can safely be omitted from the set of conditioning variables in $p(x_{t-1} \mid z_{1:t-1}, u_{1:t})$ for randomly chosen controls. This gives us the recursive update equation

$$\overline{bel}(x_t) = \int p(x_t \mid x_{t-1}, u_t) \, p(x_{t-1} \mid z_{1:t-1}, u_{1:t-1}) \, dx_{t-1} \tag{2.61}$$

As the reader easily verifies, this equation is implemented by Line 3 of the Bayes filter algorithm in Table 2.1. To summarize, the Bayes filter algorithm calculates the posterior over the state x_t conditioned on the measurement and control data up to time t. The derivation assumes that the world is Markov, that is, the state is complete.

Any concrete implementation of this algorithm requires three probability distributions: The initial belief $p(x_0)$, the measurement probability $p(z_t \mid x_t)$, and the state transition probability $p(x_t \mid u_t, x_{t-1})$. We have not yet specified these densities, but will do so in later chapters (Chapters 5 and **??**). Additionally, we also need a representation for the belief $bel(x_t)$, which will also be discussed further below.

2.4.4   The Markov Assumption

A word is in order on the Markov assumption, or the complete state assumption, since it plays such a fundamental role in the material presented in this book. The Markov assumption postulates that past and future data are independent if one knows the current state x_t. To see how severe an assumption this is, let us consider our example of mobile robot localization. In mobile robot localization, x_t is the robot's pose, and Bayes filters are applied to estimate the pose relative to a fixed map. The following factors may have a systematic effect on sensor readings. Thus, they induce violations of the Markov assumption:

- Unmodeled dynamics in the environment not included in x_t (e.g., moving people and their effects on sensor measurements in our localization example),

- inaccuracies in the probabilistic models $p(z_t \mid x_t)$ and $p(x_t \mid u_t, x_{t-1})$,

- approximation errors when using approximate representations of belief functions (e.g., grids or Gaussians, which will be discussed below), and

- software variables in the robot control software that influence multiple control selection (e.g., the variable "target location" typically influences an entire sequence of control commands).

In principle, many of these variables can be included in state representations. However, incomplete state representations are often preferable to more complete ones to reduce the computational complexity of the Bayes filter algorithm. In practice Bayes filters have been found to be surprisingly robust to such violations. As a general rule of thumb, one has to exercise care when defining the state x_t, so that the effect of unmodeled state variables has close-to-random effects.

2.5   REPRESENTATION AND COMPUTATION

In probabilistic robotics, Bayes filters are implemented in several different ways. As we will see in the next two chapters, there exist quite a variety of techniques and algorithms that are all derived from the Bayes filter. Each such technique relies on different assumptions regarding the measurement and state transition probabilities and the initial belief. Those assumptions then give rise to different types of posterior distributions, and the algorithms for computing (or approximating) those have different

computational characteristics. As a general rule of thumb, exact techniques for calculating beliefs exist only for highly specialized cases; in general robotics problems, beliefs have to be approximated. The nature of the approximation has important ramifications on the complexity of the algorithm. Finding a suitable approximation is usually a challenging problem, with no unique best answer for all robotics problems.

When choosing an approximation, one has to trade off a range of properties:

1. **Computational efficiency.** Some approximations, such as linear Gaussian approximations that will be discussed further below, make it possible to calculate beliefs in time polynomial in the dimension of the state space. Others may require exponential time. Particle based techniques, discussed further below, have an *any-time* characteristic, enabling them to trade off accuracy with computational efficiency.

2. **Accuracy of the approximation.** Some approximations can approximate a wider range of distributions more tightly than others. For example, linear Gaussian approximations are limited to unimodal distributions, whereas histogram representations can approximate multi-modal distributions, albeit with limited accuracy. Particle representations can approximate a wide array of distributions, but the number of particles needed to attain a desired accuracy can be large.

3. **Ease of implementation.** The difficulty of implementing probabilistic algorithms depends on a variety of factors, such as the form of the measurement probability $p(z_t \mid x_t)$ and the state transition probability $p(x_t \mid u_t, x_{t-1})$. Particle representations often yield surprisingly simple implementations for complex nonlinear systems—one of the reasons for their recent popularity.

The next two chapters will introduce concrete implementable algorithms, which fare quite differently relative to the criteria described above.

2.6 SUMMARY

In this section, we introduced the basic idea of Bayes filters in robotics, as a means to estimate the state of an environment (which may include the state of the robot itself).

- The interaction of a robot and its environment is modeled as a coupled dynamical system, in which the robot can manipulate its environment by choosing controls, and in which it can perceive its environment through sensor measurements.

- In probabilistic robotics, the dynamics of the robot and its environment are characterized in the form of two probabilistic laws: the state transition distribution, and the measurement distribution. The state transition distribution characterizes how state changes over time, possible as the effect of a robot control. The measurement distribution characterizes how measurements are governed by states. Both laws are probabilistic, accounting for the inherent uncertainty in state evolution and sensing.

- The *belief* of a robot is the posterior distribution over the state of the environment (including the robot state), given all past sensor measurements and all past controls. The *Bayes filter* is the principal algorithm for calculating the belief in robotics. The Bayes filter is recursive; the belief at time t is calculated from the belief at time $t - 1$.

- The Bayes filter makes a *Markov assumption* that specifies that the state is a complete summary of the past. This assumption implies the belief is sufficient to represent the past history of the robot. In robotics, the Markov assumption is usually only an approximation. We identified conditions under which it is violated.

- Since the Bayes filter is not a practical algorithm, in that it cannot be implemented on a digital computer, probabilistic algorithms use tractable approximations. Such approximations may be evaluated according to different criteria, relating to their accuracy, efficiency, and ease of implementation.

The next two chapters discuss two popular families of recursive state estimation techniques that are both derived from the Bayes filter.

2.7   BIBLIOGRAPHICAL REMARKS

3

GAUSSIAN FILTERS

3.1 INTRODUCTION

This chapter describes an important family of recursive state estimators, collectively called *Gaussian Filters*. Historically, Gaussian filters constitute the earliest tractable implementations of the Bayes filter for continuous spaces. They are also by far the most popular family of techniques to date—despite a number of shortcomings.

Gaussian techniques all share the basic idea that beliefs are represented by multivariate normal distributions. We already encountered a definition of the multivariate normal distribution in Equation (2.4), which is restated here:

$$p(x) \quad = \quad \det (2\pi\Sigma)^{-\frac{1}{2}} \exp \left\{ -\tfrac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu) \right\} \tag{3.1}$$

This density over the variable x is characterized by two sets of parameters: The mean μ and the covariance Σ. The mean μ is a vector that possesses the same dimensionality as the state x. The covariance is a quadratic matrix that is symmetric and positive-semidefinite. Its dimension is the dimensionality of the state x squared. Thus, the number of elements in the covariance matrix depends quadratically on the number of elements in the state vector.

The commitment to represent the posterior by a Gaussian has important ramifications. Most importantly, Gaussians are unimodal, that is, they posses a single maximum. Such a posterior is characteristic of many tracking problems in robotics, in which the posterior is focused around the true state with a small margin of uncertainty. Gaussian posteriors are a poor match for many global estimation problems in which many distinct hypotheses exist, each of which forming its own mode in the posterior.

The representation of a Gaussian by its mean and covariance is called the *moments representation.* This is because the mean and covariance are the first and second moments of a probability distribution; all other moments are zero for normal distributions. In this chapter, we will also discuss an alternative representation, called *canonical representation*, or sometimes *natural representation.* Both representations, the moments and the canonical representations, are functionally equivalent in that a bijective mapping exists that transforms one into the other (and back). However, they lead to filter algorithms with orthogonal computational characteristics.

This chapter introduces the two basic Gaussian filter algorithms.

■ Section 3.2 describes the Kalman filter, which implements the Bayes filter using the moments representation for a restricted class of problems with linear dynamics and measurement functions.

■ The Kalman filter is extended to nonlinear problems in Section 3.3, which describes the extended Kalman filter.

■ Section 3.4 describes the information filter, which is the dual of the Kalman filter using the canonical representation of Gaussians.

3.2   THE KALMAN FILTER

3.2.1   Linear Gaussian Systems

Probably the best studied technique for implementing Bayes filters is the *Kalman filter (KF)*. The Kalman filter was invented in the 1950s by Rudolph Emil Kalman, as a technique for filtering and prediction in linear systems. The Kalman filter implements belief computation for continuous states. It is not applicable to discrete or hybrid state spaces.

The Kalman filter represents beliefs by the moments representation: At time t, the belief is represented by the the mean μ_t and the covariance Σ_t. Posteriors are Gaussian if the following three properties hold, in addition to the Markov assumptions of the Bayes filter.

1. The next state probability $p(x_t \mid u_t, x_{t-1})$ must be a *linear* function in its arguments with added Gaussian noise. This is expressed by the following equation:

$$x_t \quad = \quad A_t x_{t-1} + B_t u_t + \varepsilon_t \; . \tag{3.2}$$

Here x_t and x_{t-1} are state vectors, and u_t is the control vector at time t. In our notation, both of these vectors are vertical vectors, that is, they are of the form

$$x_t = \begin{pmatrix} x_{1,t} \\ x_{2,t} \\ \vdots \\ x_{n,t} \end{pmatrix} \quad \text{and} \quad u_t = \begin{pmatrix} u_{1,t} \\ u_{2,t} \\ \vdots \\ u_{m,t} \end{pmatrix}. \tag{3.3}$$

A_t and B_t are matrices. A_t is a square matrix of size $n \times n$, where n is the dimension of the state vector x_t. B_t is of size $n \times m$, with m being the dimension of the control vector u_t. By multiplying the state and control vector with the matrices A_t and B_t, respectively, the state transition function becomes *linear* in its arguments. Thus, Kalman filters assume linear system dynamics.

The random variable ε_t in (3.2) is a Gaussian random vector that models the randomness in the state transition. It is of the same dimension as the state vector. Its mean is zero and its covariance will be denoted R_t. A state transition probability of the form (3.2) is called a *linear Gaussian*, to reflect the fact that it is linear in its arguments with additive Gaussian noise.

Equation (3.2) defines the state transition probability $p(x_t \mid u_t, x_{t-1})$. This probability is obtained by plugging Equation (3.2) into the definition of the multivariate normal distribution (3.1). The mean of the posterior state is given by $A_t x_{t-1} + B_t u_t$ and the covariance by R_t:

$$p(x_t \mid u_t, x_{t-1}) \tag{3.4}$$
$$= \det(2\pi R_t)^{-\frac{1}{2}} \exp\left\{-\tfrac{1}{2}(x_t - A_t x_{t-1} - B_t u_t)^T R_t^{-1}(x_t - A_t x_{t-1} - B_t u_t)\right\}$$

2. The measurement probability $p(z_t \mid x_t)$ must also be *linear* in its arguments, with added Gaussian noise:

$$z_t = C_t x_t + \delta_t. \tag{3.5}$$

Here C_t is a matrix of size $k \times n$, where k is the dimension of the measurement vector z_t. The vector δ_t describes the measurement noise. The distribution of δ_t is a multivariate Gaussian with zero mean and covariance Q_t. The measurement probability is thus given by the following multivariate normal distribution:

$$p(z_t \mid x_t) = \det(2\pi Q_t)^{-\frac{1}{2}} \exp\left\{-\tfrac{1}{2}(z_t - C_t x_t)^T Q_t^{-1}(z_t - C_t x_t)\right\} \tag{3.6}$$

1:         **Algorithm Kalman_filter**($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$)**:**
2:             $\bar{\mu}_t = A_t \, \mu_{t-1} + B_t \, u_t$
3:             $\bar{\Sigma}_t = A_t \, \Sigma_{t-1} \, A_t^T + R_t$
4:             $K_t = \bar{\Sigma}_t \, C_t^T (C_t \, \bar{\Sigma}_t \, C_t^T + Q_t)^{-1}$
5:             $\mu_t = \bar{\mu}_t + K_t(z_t - C_t \, \bar{\mu}_t)$
6:             $\Sigma_t = (I - K_t \, C_t) \, \bar{\Sigma}_t$
7:             return μ_t, Σ_t

Table 3.1   The Kalman filter algorithm for linear Gaussian state transitions and measurements.

3. Finally, the initial belief $bel(x_0)$ must be normal distributed. We will denote the mean of this belief by μ_0 and the covariance by Σ_0:

$$bel(x_0) \;\;=\;\; p(x_0) \;\;=\;\; \det(2\pi\Sigma_0)^{-\frac{1}{2}} \, \exp\left\{-\tfrac{1}{2}\,(x_0 - \mu_0)^T \Sigma_0^{-1}(x_0 - \mu_0)\right\}$$

These three assumptions are sufficient to ensure that the posterior $bel(x_t)$ is always a Gaussian, for any point in time t. The proof of this non-trivial result can be found below, in the mathematical derivation of the Kalman filter (Section 3.2.4).

3.2.2   The Kalman Filter Algorithm

The Kalman filter algorithm is depicted in Table 3.1. Kalman filters represent the belief $bel(x_t)$ at time t by the mean μ_t and the covariance Σ_t. The input of the Kalman filter is the belief at time $t - 1$, represented by μ_{t-1} and Σ_{t-1}. To update these parameters, Kalman filters require the control u_t and the measurement z_t. The output is the belief at time t, represented by μ_t and Σ_t.

In Lines 2 and 3, the predicted belief $\bar{\mu}$ and $\bar{\Sigma}$ is calculated representing the belief $\overline{bel}(x_t)$ one time step later, but before incorporating the measurement z_t. This belief is obtained by incorporating the control u_t. The mean is updated using the deterministic version of the state transition function (3.2), with the mean μ_{t-1} substituted for the state x_{t-1}. The update of the covariance considers the fact that states depend on previous states through the linear matrix A_t. This matrix is multiplied twice into the covariance, since the covariance is a quadratic matrix.

The belief $\overline{bel}(x_t)$ is subsequently transformed into the desired belief $bel(x_t)$ in Lines 4 through 6, by incorporating the measurement z_t. The variable K_t, computed in Line 4 is called *Kalman gain*. It specifies the degree to which the measurement is incorporated into the new state estimate. Line 5 manipulates the mean, by adjusting it in proportion to the Kalman gain K_t and the deviation of the actual measurement, z_t, and the measurement predicted according to the measurement probability (3.5). Finally, the new covariance of the posterior belief is calculated in Line 6, adjusting for the information gain resulting from the measurement.

The Kalman filter is computationally quite efficient. For today's best algorithms, the complexity of matrix inversion is approximately $O(d^{2.8})$ for a matrix of size $d \times d$. Each iteration of the Kalman filter algorithm, as stated here, is lower bounded by (approximately) $O(k^{2.8})$, where k is the dimension of the measurement vector z_t. This (approximate) cubic complexity stems from the matrix inversion in Line 4. It is also at least in $O(n^2)$, where n is the dimension of the state space, due to the multiplication in Line 6 (the matrix $K_t C_t$ may be sparse). In many applications—such as the robot mapping applications discussed in later chapters—-the measurement space is much lower dimensional than the state space, and the update is dominated by the $O(n^2)$ operations.

3.2.3   Illustration

Figure 3.2 illustrates the Kalman filter algorithm for a simplistic one-dimensional localization scenario. Suppose the robot moves along the horizontal axis in each diagram in Figure 3.2. Let the prior over the robot location be given by the normal distribution shown in Figure 3.2a. The robot queries its sensors on its location (e.g., a GPS system), and those return a measurement that is centered at the peak of the bold Gaussian in Figure 3.2b. This bold Gaussian illustrates this measurement: Its peak is the value predicted by the sensors, and its width (variance) corresponds to the uncertainty in the measurement. Combining the prior with the measurement, via Lines 4 through 6 of the Kalman filter algorithm in Table 3.1, yields the bold Gaussian in Figure 3.2c. This belief's mean lies between the two original means, and its uncertainty radius is smaller than both contributing Gaussians. The fact that the residual uncertainty is smaller than the contributing Gaussians may appear counter-intuitive, but it is a general characteristic of information integration in Kalman filters.

Next, assume the robot moves towards the right. Its uncertainty grows due to the fact that the next state transition is stochastic. Lines 2 and 3 of the Kalman filter provides us with the Gaussian shown in bold in Figure 3.2d. This Gaussian is shifted by the amount the robot moved, and it is also wider for the reasons just explained. Next, the
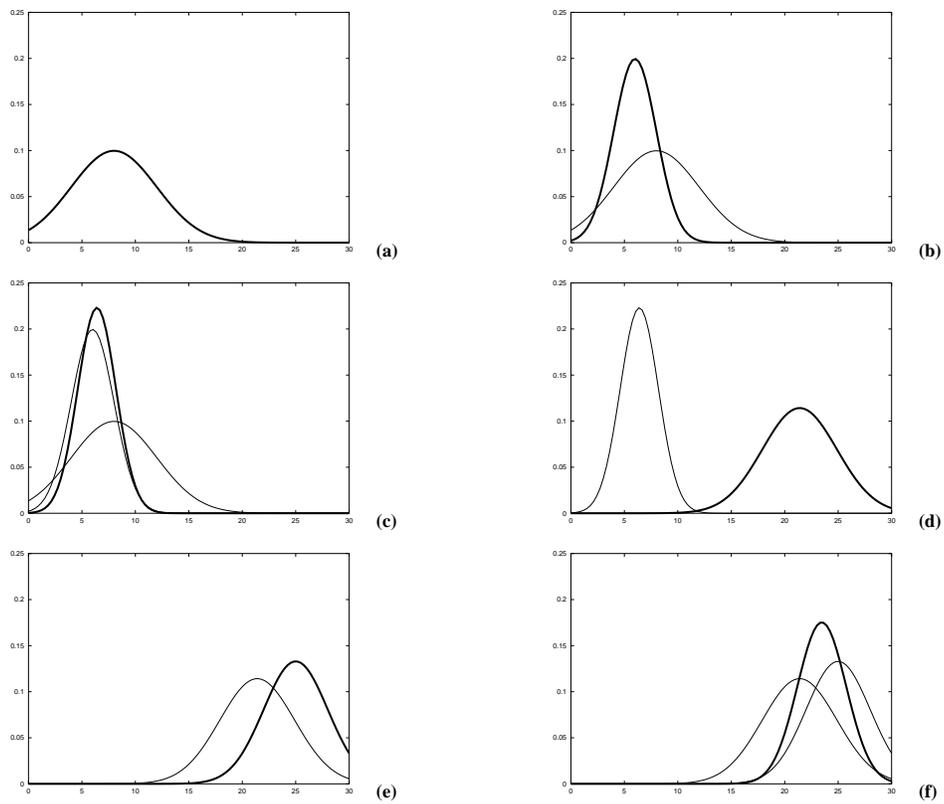
Figure 3.2 Illustration of Kalman fi lters: (a) initial belief, (b) a measurement (in bold) with the associated uncertainty, (c) belief after integrating the measurement into the belief using the Kalman fi lter algorithm, (d) belief after motion to the right (which introduces uncertainty), (e) a new measurement with associated uncertainty, and (f) the resulting belief.

robot receives a second measurement illustrated by the bold Gaussian in Figure 3.2e, which leads to the posterior shown in bold in Figure 3.2f.

As this example illustrates, the Kalman filter alternates a *measurement update step* (Lines 5-7), in which sensor data is integrated into the present belief, with a *prediction step* (or control update step), which modifies the belief in accordance to an action. The update step decreases and the prediction step increases uncertainty in the robot's belief.

3.2.4 Mathematical Derivation of the KF

This section derives the Kalman filter algorithm in Table 3.1. The section can safely be skipped at first reading.

Part 1: Prediction. Our derivation begins with Lines 2 and 3 of the algorithm, in which the belief $\overline{bel}(x_t)$ is calculated from the belief one time step earlier, $bel(x_{t-1})$. Lines 2 and 3 implement the update step described in Equation (2.61), restated here for the reader's convenience:

$$\overline{bel}(x_t) \;\; = \;\; \int \underbrace{p(x_t \mid x_{t-1}, u_t)}_{\sim \mathcal{N}(x_t; A_t x_{t-1}+B_t u_t, R_t)} \quad \underbrace{bel(x_{t-1})}_{\sim \mathcal{N}(x_{t-1}; \mu_{t-1}, \Sigma_{t-1})} \quad dx_{t-1} \tag{3.7}$$

The "prior" belief $bel(x_{t-1})$ is represented by the mean μ_{t-1} and the covariance Σ_{t-1}. The state transition probability $p(x_t \mid x_{t-1}, u_t)$ was given in (3.4) as a normal distribution over x_t with mean $A_t x_{t-1} + B_t u_t$ and covariance R_t. As we shall show now, the outcome of (3.7) is again a Gaussian with mean $\bar{\mu}_t$ and covariance $\bar{\Sigma}_t$ as stated in Table 3.1.

We begin by writing (3.7) in its Gaussian form:

$$\overline{bel}(x_t) \;\; = \;\; \eta \int \exp\left\{-\tfrac{1}{2}\,(x_t - A_t\,x_{t-1} - B_t\,u_t)^T\,R_t^{-1}\,(x_t - A_t\,x_{t-1} - B_t\,u_t)\right\}$$
$$\exp\left\{-\tfrac{1}{2}\,(x_{t-1} - \mu_{t-1})^T\Sigma_{t-1}^{-1}(x_{t-1} - \mu_{t-1})\right\}\,dx_{t-1}\,. \tag{3.8}$$

In short, we have

$$\overline{bel}(x_t) \;\; = \;\; \eta \int \exp\left\{-L_t\right\}\,dx_{t-1} \tag{3.9}$$

with

$$L_t = \frac{1}{2} (x_t - A_t \, x_{t-1} - B_t \, u_t)^T \, R_t^{-1} \, (x_t - A_t \, x_{t-1} - B_t \, u_t)$$
$$+ \frac{1}{2} (x_{t-1} - \mu_{t-1})^T \, \Sigma_{t-1}^{-1} \, (x_{t-1} - \mu_{t-1}). \tag{3.10}$$

Notice that L_t is quadratic in x_{t-1}; it is also quadratic in x_t.

Expression (3.9) contains an integral. To solve this integral in closed form, we will now decompose L_t into two functions, $L_t(x_{t-1}, x_t)$ and $L_t(x_t)$:

$$L_t = L_t(x_{t-1}, x_t) + L_t(x_t) \tag{3.11}$$

so that all terms containing x_{t-1} are collected in $L_t(x_{t-1}, x_t)$. This decomposition will allow us to move $L_t(x_t)$ outside the integration, since its value does not depend on the integration variable x_{t-1}:

$$\overline{bel}(x_t) = \eta \int \exp\{-L_t\} \, dx_{t-1}$$
$$= \eta \int \exp\{-L_t(x_{t-1}, x_t) - L_t(x_t)\} \, dx_{t-1}$$
$$= \eta \, \exp\{-L_t(x_t)\} \int \exp\{-L_t(x_{t-1}, x_t)\} \, dx_{t-1} \tag{3.12}$$

Furthermore, we will choose $L_t(x_{t-1}, x_t)$ such that the value of the integral in (3.12) does not depend on x_t. Thus, the integral will simply become a constant relative to the problem of estimating the belief distribution over x_t. The resulting distribution over x_t will be entirely defined through $L_t(x_t)$.

$$\overline{bel}(x_t) = \eta \, \exp\{-L_t(x_t)\} \tag{3.13}$$

Let us now perform this decomposition. We are seeking a function $L_t(x_{t-1}, x_t)$ quadratic in x_{t-1}. (This function will also depend on x_t, but that shall not concern us at this point.) To determine the coefficients of this quadratic, we calculate the first two derivatives of L_t:

$$\frac{\partial L_t}{\partial x_{t-1}} = -A_t^T \, R_t^{-1} \, (x_t - A_t \, x_{t-1} - B_t \, u_t) + \Sigma_{t-1}^{-1} \, (x_{t-1} - \mu_{t-1}) \tag{3.14}$$

$$\frac{\partial^2 L_t}{\partial x_{t-1}^2} \;\; = \;\; A_t^T \; R_t^{-1} \; A_t + \Sigma_{t-1}^{-1} \;\; =: \;\; \Psi_t^{-1} \tag{3.15}$$

Ψ_t defines the curvature of $L_t(x_{t-1}, x_t)$. Setting the first derivative of L_t to 0 gives us the mean:

$$A_t^T \; R_t^{-1} \; (x_t - A_t \; x_{t-1} - B_t \; u_t) \;\; = \;\; \Sigma_{t-1}^{-1} \; (x_{t-1} - \mu_{t-1}) \tag{3.16}$$

This expression is now solved for x_{t-1}

$$
\begin{aligned}
\Longleftrightarrow \quad & A_t^T \; R_t^{-1} \; (x_t - B_t \; u_t) - A_t^T \; R_t^{-1} \; A_t \; x_{t-1} \;\; = \;\; \Sigma_{t-1}^{-1} \; x_{t-1} - \Sigma_{t-1}^{-1} \; \mu_{t-1} \\
\Longleftrightarrow \quad & A_t^T \; R_t^{-1} \; A_t \; x_{t-1} + \Sigma_{t-1}^{-1} \; x_{t-1} \;\; = \;\; A_t^T \; R_t^{-1} \; (x_t - B_t \; u_t) + \Sigma_{t-1}^{-1} \; \mu_{t-1} \\
\Longleftrightarrow \quad & (A_t^T \; R_t^{-1} \; A_t \; + \Sigma_{t-1}^{-1}) \; x_{t-1} \;\; = \;\; A_t^T \; R_t^{-1} \; (x_t - B_t \; u_t) + \Sigma_{t-1}^{-1} \; \mu_{t-1} \\
\Longleftrightarrow \quad & \Psi_t^{-1} \; x_{t-1} \;\; = \;\; A_t^T \; R_t^{-1} \; (x_t - B_t \; u_t) + \Sigma_{t-1}^{-1} \; \mu_{t-1} \\
\Longleftrightarrow \quad & x_{t-1} \;\; = \;\; \Psi_t \; [A_t^T \; R_t^{-1} \; (x_t - B_t \; u_t) + \Sigma_{t-1}^{-1} \; \mu_{t-1}]
\end{aligned} \tag{3.17}
$$

Thus, we now have a quadratic function $L_t(x_{t-1}, x_t)$, defined as follows:

$$
\begin{aligned}
L_t(x_{t-1}, x_t) \;\; = \;\; & \tfrac{1}{2}(x_{t-1} - \Psi_t \; [A_t^T \; R_t^{-1} \; (x_t - B_t \; u_t) + \Sigma_{t-1}^{-1} \; \mu_{t-1}])^T \; \Psi^{-1} \\
& (x_{t-1} - \Psi_t \; [A_t^T \; R_t^{-1} \; (x_t - B_t \; u_t) + \Sigma_{t-1}^{-1} \; \mu_{t-1}])
\end{aligned} \tag{3.18}
$$

Clearly, this is not the only quadratic function satisfying our decomposition in (3.11). However, $L_t(x_{t-1}, x_t)$ is of the common quadratic form of the negative exponent of a normal distribution. In fact the function

$$\det(2\pi\Psi)^{-\frac{1}{2}} \; \exp\{-L_t(x_{t-1}, x_t)\} \tag{3.19}$$

is a valid probability density function (PDF) for the variable x_{t-1}. As the reader easily verifies, this function is of the form defined in (3.1). We know from (2.5) that PDFs integrate to 1. Thus, we have

$$\int \; \det(2\pi\Psi)^{-\frac{1}{2}} \; \exp\{-L_t(x_{t-1}, x_t)\} \; dx_{t-1} \;\; = \;\; 1 \tag{3.20}$$

From this it follows that

$$\int \exp\{-L_t(x_{t-1}, x_t)\}\, dx_{t-1} \quad = \quad \det(2\pi\Psi)^{\frac{1}{2}}\,. \tag{3.21}$$

The important thing to notice is that the value of this integral is *independent* of x_t, our target variable. Thus, for our problem of calculating a distribution over x_t, this integral is constant. Subsuming this constant into the normalizer η, we get the following expression for Equation (3.12):

$$
\begin{aligned}
\overline{bel}(x_t) \quad &= \quad \eta \, \exp\{-L_t(x_t)\} \int \exp\{-L_t(x_{t-1}, x_t)\}\, dx_{t-1} \\
&= \quad \eta \, \exp\{-L_t(x_t)\} \tag{3.22}
\end{aligned}
$$

Notice that the normalizers η left and right of the equal sign are *not* the same. This decomposition establishes the correctness of (3.13).

It remains to determine the function $L_t(x_t)$, which is the difference of L_t, defined in (3.10), and $L_t(x_{t-1}, x_t)$, defined in (3.18):

$$
\begin{aligned}
L_t(x_t) \quad = \quad & L_t - L_t(x_{t-1}, x_t) \\
= \quad & \tfrac{1}{2}\,(x_t - A_t\, x_{t-1} - B_t\, u_t)^T\, R_t^{-1}\,(x_t - A_t\, x_{t-1} - B_t\, u_t) \\
& + \tfrac{1}{2}\,(x_{t-1} - \mu_{t-1})^T\, \Sigma_{t-1}^{-1}\,(x_{t-1} - \mu_{t-1}) \\
& - \tfrac{1}{2}(x_{t-1} - \Psi_t\,[A_t^T\, R_t^{-1}\,(x_t - B_t\, u_t) + \Sigma_{t-1}^{-1}\,\mu_{t-1}])^T\, \Psi^{-1} \\
& (x_{t-1} - \Psi_t\,[A_t^T\, R_t^{-1}\,(x_t - B_t\, u_t) + \Sigma_{t-1}^{-1}\,\mu_{t-1}]) \tag{3.23}
\end{aligned}
$$

Let us quickly verify that $L_t(x_t)$ indeed does not depend on x_{t-1}. To do so, we substitute back $\Psi_t = (A_t^T\, R_t^{-1}\, A_t + \Sigma_{t-1}^{-1})^{-1}$, and multiply out the terms above. For the reader's convenience, terms that contain x_{t-1} are underlined (doubly if they are quadratic in x_{t-1}).

$$
\begin{aligned}
L_t(x_t) \quad = \quad & \underline{\underline{\tfrac{1}{2}\, x_{t-1}^T A_t^T\, R_t^{-1}\, A_t\, x_{t-1}}} - \underline{x_{t-1}^T A_t^T\, R_t^{-1}\,(x_t - B_t\, u_t)} \\
& + \tfrac{1}{2}\,(x_t - B_t\, u_t)^T\, R_t^{-1}\,(x_t - B_t\, u_t) \\
& + \underline{\underline{\tfrac{1}{2}\, x_{t-1}^T\, \Sigma_{t-1}^{-1}\, x_{t-1}}} - \underline{x_{t-1}^T\, \Sigma_{t-1}^{-1}\,\mu_{t-1}} + \tfrac{1}{2}\,\mu_{t-1}^T\, \Sigma_{t-1}^{-1}\,\mu_{t-1} \\
& - \underline{\underline{\tfrac{1}{2}\, x_{t-1}^T\,(A_t^T\, R_t^{-1}\, A_t + \Sigma_{t-1}^{-1})\, x_{t-1}}}
\end{aligned}
$$

$$+x_{t-1}^T \, [A_t^T \, R_t^{-1} \, (x_t - B_t \, u_t) + \Sigma_{t-1}^{-1} \, \mu_{t-1}]$$
$$-\tfrac{1}{2} \, [A_t^T \, R_t^{-1} \, (x_t - B_t \, u_t) + \Sigma_{t-1}^{-1} \, \mu_{t-1}]^T \, (A_t^T \, R_t^{-1} \, A_t + \Sigma_{t-1}^{-1})^{-1}$$
$$[A_t^T \, R_t^{-1} \, (x_t - B_t \, u_t) + \Sigma_{t-1}^{-1} \, \mu_{t-1}] \tag{3.24}$$

It is now easily seen that all terms that contain x_{t-1} cancel out. This should come at no surprise, since it is a consequence of our construction of $L_t(x_{t-1}, x_t)$.

$$
\begin{aligned}
L_t(x_t) \;=\; & +\tfrac{1}{2} \, (x_t - B_t \, u_t)^T \, R_t^{-1} \, (x_t - B_t \, u_t) \; + \tfrac{1}{2} \, \mu_{t-1}^T \, \Sigma_{t-1}^{-1} \, \mu_{t-1} \\
& -\tfrac{1}{2} \, [A_t^T \, R_t^{-1} \, (x_t - B_t \, u_t) + \Sigma_{t-1}^{-1} \, \mu_{t-1}]^T \, (A_t^T \, R_t^{-1} \, A_t + \Sigma_{t-1}^{-1})^{-1} \\
& \qquad [A_t^T \, R_t^{-1} \, (x_t - B_t \, u_t) + \Sigma_{t-1}^{-1} \, \mu_{t-1}] \tag{3.25}
\end{aligned}
$$

Furthermore, $L_t(x_t)$ is quadratic in x_t. This observation means that $\overline{bel}(x_t)$ is indeed normal distributed. The mean and covariance of this distribution are of course the minimum and curvature of $L_t(x_t)$, which we now easily obtain by by computing the first and second derivatives of $L_t(x_t)$ with respect to x_t:

$$
\begin{aligned}
\frac{\partial L_t(x_t)}{\partial x_t} \;=\; & R_t^{-1} \, (x_t - B_t \, u_t) \; - R_t^{-1} \, A_t \, (A_t^T \, R_t^{-1} \, A_t + \Sigma_{t-1}^{-1})^{-1} \\
& [A_t^T \, R_t^{-1} \, (x_t - B_t \, u_t) + \Sigma_{t-1}^{-1} \, \mu_{t-1}] \\
\;=\; & [R_t^{-1} - R_t^{-1} \, A_t \, (A_t^T \, R_t^{-1} \, A_t + \Sigma_{t-1}^{-1})^{-1} A_t^T \, R_t^{-1}] \, (x_t - B_t \, u_t) \\
& - R_t^{-1} \, A_t \, (A_t^T \, R_t^{-1} \, A_t + \Sigma_{t-1}^{-1})^{-1} \, \Sigma_{t-1}^{-1} \, \mu_{t-1} \tag{3.26}
\end{aligned}
$$

The *inversion lemma* stated (and proved) in Table 3.2 allows us to express the first factor as follows:

$$R_t^{-1} - R_t^{-1} \, A_t \, (A_t^T \, R_t^{-1} \, A_t + \Sigma_{t-1}^{-1})^{-1} A_t^T \, R_t^{-1} \;=\; (R_t + A_t \, \Sigma_{t-1} \, A_t^T)^{-1} \tag{3.27}$$

Hence the desired derivative is given by the following expression:

$$
\begin{aligned}
\frac{\partial L_t(x_t)}{\partial x_t} \;=\; & (R_t + A_t \, \Sigma_{t-1} \, A_t^T)^{-1} \, (x_t - B_t \, u_t) \\
& - R_t^{-1} \, A_t \, (A_t^T \, R_t^{-1} \, A_t + \Sigma_{t-1}^{-1})^{-1} \, \Sigma_{t-1}^{-1} \, \mu_{t-1} \tag{3.28}
\end{aligned}
$$

Inversion Lemma. For any invertible quadratic matrices R and Q and any matrix P with appropriate dimension, the following holds true

$$(R + P\,Q\,P^T)^{-1} \;=\; R^{-1} - R^{-1}\,P\,(Q^{-1} + P^T\,R^{-1}\,P)^{-1}\,P^T\,R^{-1}$$

assuming that all above matrices can be inverted as stated.

Proof. It suffices to show that

$$(R^{-1} - R^{-1}\,P\,(Q^{-1} + P^T\,R^{-1}\,P)^{-1}\,P^T\,R^{-1})\,(R + P\,Q\,P^T) \;=\; I$$

This is shown through a series of transformations:

$$\begin{aligned}
&= \underbrace{R^{-1}\,R}_{=\,I} + R^{-1}\,P\,Q\,P^T \;-\; R^{-1}\,P\,(Q^{-1} + P^T\,R^{-1}\,P)^{-1}\,P^T\,\underbrace{R^{-1}\,R}_{=\,I} \\
&\quad - R^{-1}\,P\,(Q^{-1} + P^T\,R^{-1}\,P)^{-1}\,P^T\,R^{-1}\,P\,Q\,P^T \\
&= I + R^{-1}\,P\,Q\,P^T \;-\; R^{-1}\,P\,(Q^{-1} + P^T\,R^{-1}\,P)^{-1}\,P^T \\
&\quad - R^{-1}\,P\,(Q^{-1} + P^T\,R^{-1}\,P)^{-1}\,P^T\,R^{-1}\,P\,Q\,P^T \\
&= I + R^{-1}\,P\,[Q\,P^T \;-\; (Q^{-1} + P^T\,R^{-1}\,P)^{-1}\,P^T \\
&\quad - (Q^{-1} + P^T\,R^{-1}\,P)^{-1}\,P^T\,R^{-1}\,P\,Q\,P^T] \\
&= I + R^{-1}\,P\,[Q\,P^T \;-\; (Q^{-1} + P^T\,R^{-1}\,P)^{-1}\,\underbrace{Q^{-1}\,Q}_{=\,I}\,P^T \\
&\quad - (Q^{-1} + P^T\,R^{-1}\,P)^{-1}\,P^T\,R^{-1}\,P\,Q\,P^T] \\
&= I + R^{-1}\,P\,[Q\,P^T \;-\; \underbrace{(Q^{-1} + P^T\,R^{-1}\,P)^{-1}\,(Q^{-1} + P^T\,R^{-1}\,P)}_{=\,I}\,Q\,P^T] \\
&= I + R^{-1}\,P\,\underbrace{[Q\,P^T \;-\; Q\,P^T]}_{=\,0} \;=\; I
\end{aligned}$$

Table 3.2   The (specialized) inversion lemma.

The minimum of $L_t(x_t)$ is attained when the first derivative is zero.

$$(R_t + A_t\,\Sigma_{t-1}\,A_t^T)^{-1}\,(x_t - B_t\,u_t) \;=\; R_t^{-1}\,A_t\,(A_t^T\,R_t^{-1}\,A_t + \Sigma_{t-1}^{-1})^{-1}\,\Sigma_{t-1}^{-1}\,\mu_{t-1}$$

$$(3.29)$$

Solving this for the target variable x_t gives us the surprisingly compact result

$$
\begin{aligned}
x_t &= B_t\, u_t + \underbrace{(R_t + A_t\, \Sigma_{t-1}\, A_t^T)R_t^{-1}\, A_t}_{A_t + A_t\, \Sigma_{t-1}\, A_t^T R_t^{-1}\, A_t}\ \underbrace{(A_t^T\, R_t^{-1}\, A_t + \Sigma_{t-1}^{-1})^{-1}\, \Sigma_{t-1}^{-1}}_{(\Sigma_{t-1} A_t^T\, R_t^{-1}\, A_t + I)^{-1}}\ \mu_{t-1} \\
&= B_t\, u_t + A_t\ \underbrace{(I + \Sigma_{t-1}\, A_t^T R_t^{-1}\, A_t)\,(\Sigma_{t-1} A_t^T\, R_t^{-1}\, A_t + I)^{-1}}_{=\,I}\ \mu_{t-1} \\
&= B_t\, u_t + A_t\, \mu_{t-1}
\end{aligned}
\tag{3.30}
$$

Thus, the mean of the belief $\overline{bel}(x_t)$ after incorporating the motion command u_t is $B_t\, u_t + A_t\, \mu_{t-1}$. This proves the correctness of Line 2 of the Kalman filter algorithm in Table 3.1. Line 3 is now obtained by calculating the second derivative of $L_t(x_t)$:

$$
\frac{\partial^2 L_t(x_t)}{\partial x_t^2} = (R_t + A_t\, \Sigma_{t-1}\, A_t^T)^{-1}
\tag{3.31}
$$

This is the curvature of the quadratic function $L_t(x_t)$, whose inverse is the covariance of the belief $\overline{bel}(x_t)$.

To summarize, we showed that the prediction steps in Lines 2 and 3 of the Kalman filter algorithm indeed implement the Bayes filter prediction step. To do so, we first decomposed the exponent of the belief $\overline{bel}(x_t)$ into two functions, $L_t(x_{t-1}, x_t)$ and $L_t(x_t)$. Then we showed that $L_t(x_{t-1}, x_t)$ changes the predicted belief $\overline{bel}(x_t)$ only by a constant factor, which can be subsumed into the normalizing constant η. Finally, we determined the function $L_t(x_t)$ and showed that it results in the mean $\bar{\mu}_t$ and covariance $\bar{\Sigma}_t$ of the Kalman filter prediction $\overline{bel}(x_t)$.

Part 2: Measurement Update. We will now derive the measurement update in Lines 4, 5, and 6 (Table 3.1) of our Kalman filter algorithm. We begin with the general Bayes filter mechanism for incorporating measurements, stated in Equation (2.58) and restated here in annotated form:

$$
bel(x_t) = \eta\ \underbrace{p(z_t \mid x_t)}_{\sim \mathcal{N}(z_t;\, C_t x_t,\, Q_t)}\ \underbrace{\overline{bel}(x_t)}_{\sim \mathcal{N}(x_t;\, \bar{\mu}_t,\, \bar{\Sigma}_t)}
\tag{3.32}
$$

The mean and covariance of $\overline{bel}(x_t)$ are obviously given by $\bar{\mu}_t$ and $\bar{\Sigma}_t$. The measurement probability $p(z_t \mid x_t)$ was defined in (3.6) to be normal as well, with mean $C_t\, x_t$

and covariance Q_t. Thus, the product is given by an exponential

$$bel(x_t) \quad = \quad \eta \, \exp\{-J_t\} \tag{3.33}$$

with

$$J_t \quad = \quad \tfrac{1}{2} \, (z_t - C_t x_t)^T \, Q_t^{-1} \, (z_t - C_t x_t) + \tfrac{1}{2} \, (x_t - \bar{\mu}_t)^T \, \bar{\Sigma}_t^{-1} \, (x_t - \bar{\mu}_t) \tag{3.34}$$

This function is quadratic in x_t, hence $bel(x_t)$ is a Gaussian. To calculate its parameters, we once again calculate the first two derivatives of J_t with respect to x_t:

$$\frac{\partial J}{\partial x_t} \quad = \quad -C_t^T \, Q_t^{-1} \, (z_t - C_t x_t) + \bar{\Sigma}_t^{-1} \, (x_t - \bar{\mu}_t) \tag{3.35}$$

$$\frac{\partial^2 J}{\partial x_t^2} \quad = \quad C_t^T \, Q_t^{-1} \, C_t + \bar{\Sigma}_t^{-1} \tag{3.36}$$

The second term is the inverse of the covariance of $bel(x_t)$:

$$\Sigma_t \quad = \quad (C_t^T \, Q_t^{-1} \, C_t + \bar{\Sigma}_t^{-1})^{-1} \tag{3.37}$$

The mean of $bel(x_t)$ is the minimum of this quadratic function, which we now calculate by setting the first derivative of J_t to zero (and substituting μ_t for x_t):

$$C_t^T \, Q_t^{-1} \, (z_t - C_t \mu_t) \quad = \quad \bar{\Sigma}_t^{-1} \, (\mu_t - \bar{\mu}_t) \tag{3.38}$$

The expression on the left of the equal sign can be transformed as follows:

$$
\begin{aligned}
C_t^T \, Q_t^{-1} \, & (z_t - C_t \mu_t) \\
= \quad & C_t^T \, Q_t^{-1} \, (z_t - C_t \, \mu_t + C_t \, \bar{\mu}_t - C_t \, \bar{\mu}_t) \\
= \quad & C_t^T \, Q_t^{-1} \, (z_t - C_t \bar{\mu}_t) - C_t^T \, Q_t^{-1} \, C_t \, (\mu_t - \bar{\mu}_t)
\end{aligned} \tag{3.39}
$$

Substituting this back into (3.38) gives us

$$C_t^T \, Q_t^{-1} \, (z_t - C_t \bar{\mu}_t) \quad = \quad \underbrace{(C_t^T \, Q_t^{-1} \, C_t + \bar{\Sigma}_t^{-1})}_{= \, \Sigma_t^{-1}} \, (\mu_t - \bar{\mu}_t) \tag{3.40}$$

and hence we have

$$\Sigma_t \, C_t^T \, Q_t^{-1} \, (z_t - C_t \, \bar{\mu}_t) \;\; = \;\; \mu_t - \bar{\mu}_t \tag{3.41}$$

We now define the Kalman gain as

$$K_t \;\; = \;\; \Sigma_t \, C_t^T \, Q_t^{-1} \tag{3.42}$$

and obtain

$$\mu_t \;\; = \;\; \bar{\mu}_t + K_t \, (z_t - C_t \, \bar{\mu}_t) \tag{3.43}$$

This proves the correctness of Line 5 in the Kalman filter algorithm in Table 3.1.

The Kalman gain, as defined in (3.42), is a function of Σ_t. This is at odds with the fact that we utilize K_t to calculate Σ_t in Line 6 of the algorithm. The following transformation shows us how to expresses K_t in terms of covariances other than Σ_t. It begins with the definition of K_t in (3.42):

$$
\begin{aligned}
K_t \;\; &= \;\; \Sigma_t \, C_t^T \, Q_t^{-1} \\
&= \;\; \Sigma_t \, C_t^T \, Q_t^{-1} \, \underbrace{(C_t \, \bar{\Sigma}_t \, C_t^T + Q_t) \, (C_t \, \bar{\Sigma}_t \, C_t^T + Q_t)^{-1}}_{=\,I} \\
&= \;\; \Sigma_t \, (C_t^T \, Q_t^{-1} \, C_t \, \bar{\Sigma}_t \, C_t^T + C_t^T \, \underbrace{Q_t^{-1} \, Q_t}_{=\,I}) \, (C_t \, \bar{\Sigma}_t \, C_t^T + Q_t)^{-1} \\
&= \;\; \Sigma_t \, (C_t^T \, Q_t^{-1} \, C_t \, \bar{\Sigma}_t \, C_t^T + C_t^T) \, (C_t \, \bar{\Sigma}_t \, C_t^T + Q_t)^{-1} \\
&= \;\; \Sigma_t \, (C_t^T \, Q_t^{-1} \, C_t \, \bar{\Sigma}_t \, C_t^T + \underbrace{\bar{\Sigma}_t^{-1} \, \bar{\Sigma}_t}_{=\,I} \, C_t^T) \, (C_t \, \bar{\Sigma}_t \, C_t^T + Q_t)^{-1} \\
&= \;\; \Sigma_t \, \underbrace{(C_t^T \, Q_t^{-1} \, C_t + \bar{\Sigma}_t^{-1})}_{=\,\Sigma_t^{-1}} \, \bar{\Sigma}_t \, C_t^T \, (C_t \, \bar{\Sigma}_t \, C_t^T + Q_t)^{-1} \\
&= \;\; \underbrace{\Sigma_t \, \Sigma_t^{-1}}_{=\,I} \, \bar{\Sigma}_t \, C_t^T \, (C_t \, \bar{\Sigma}_t \, C_t^T + Q_t)^{-1} \\
&= \;\; \bar{\Sigma}_t \, C_t^T \, (C_t \, \bar{\Sigma}_t \, C_t^T + Q_t)^{-1} \tag{3.44}
\end{aligned}
$$

This expression proves the correctness of Line 4 of our Kalman filter algorithm. Line 6 is obtained by expressing the covariance using the Kalman gain K_t. The advantage

of the calculation in Table 3.1 over the definition in Equation (3.37) lies in the fact that we can avoid inverting the state covariance matrix. This is essential for applications of Kalman filters to high-dimensional state spaces.

Our transformation is once again carried out using the *inversion lemma*, which was already stated in Table 3.2. Here we restate it using the notation of Equation (3.37):

$$(\bar{\Sigma}_t^{-1} + C_t^T \, Q_t^{-1} \, C_t)^{-1} \;\; = \;\; \bar{\Sigma}_t - \bar{\Sigma}_t \, C_t^T \, (Q_t + C_t \, \bar{\Sigma}_t \, C_t^T)^{-1} \, C_t \, \bar{\Sigma}_t \quad (3.45)$$

This lets us arrive at the following expression for the covariance:

$$
\begin{aligned}
\Sigma_t \;\; &= \;\; (C_t^T \, Q_t^{-1} \, C_t + \bar{\Sigma}_t^{-1})^{-1} \\
&= \;\; \bar{\Sigma}_t - \bar{\Sigma}_t \, C_t^T \, (Q_t + C_t \, \bar{\Sigma}_t \, C_t^T)^{-1} \, C_t \, \bar{\Sigma}_t \\
&= \;\; [I - \underbrace{\bar{\Sigma}_t \, C_t^T \, (Q_t + C_t \, \bar{\Sigma}_t \, C_t^T)^{-1}}_{=\, K_t,\ \text{see Eq. (3.44)}} \, C_t] \, \bar{\Sigma}_t \\
&= \;\; (I - K_t \, C_t) \, \bar{\Sigma}_t \qquad\qquad\qquad\qquad\qquad (3.46)
\end{aligned}
$$

This proves the correctness of Line 6 of our Kalman filter algorithm.

3.3   THE EXTENDED KALMAN FILTER

The assumptions of linear state transitions and linear measurements with added Gaussian noise are rarely fulfilled in practice. For example, a robot that moves with constant translational and rotational velocity typically moves on a circular trajectory, which cannot be described by linear next state transitions. This observation, along with the assumption of unimodal beliefs, renders plain Kalman filters, as discussed so far, inapplicable to all but the most trivial robotics problems.

The extended Kalman filter (EKF) overcomes one of these assumptions: the linearity assumption. Here the assumption is that the next state probability and the measurement probabilities are governed by nonlinear functions g and h, respectively:

$$
\begin{aligned}
x_t \;\; &= \;\; g(u_t, x_{t-1}) + \varepsilon_t & (3.47) \\
z_t \;\; &= \;\; h(x_t) + \delta_t \, . & (3.48)
\end{aligned}
$$

This model strictly generalizes the linear Gaussian model underlying Kalman filters, postulated in Equations (3.2) and (3.5). The function g replaces the matrices A_t and B_t in (3.2), and h replaces the matrix C_t in (3.5). Unfortunately, with arbitrary functions g and h, the belief is no longer a Gaussian. In fact, performing the belief update exactly is usually impossible for nonlinear functions g and h, in the sense that the Bayes filter does not possess a closed-form solution.

The extended Kalman filter (EKF) calculates an approximation to the true belief. It represents this approximation by a Gaussian. In particular, the belief $bel(x_t)$ at time t is represented by a mean μ_t and a covariance Σ_t. Thus, the EKF inherits from the Kalman filter the basic belief representation, but it differs in that this belief is only approximate, not exact as was the case in Kalman filters.

3.3.1   Linearization Via Taylor Expansion

The key idea underlying the EKF is called *linearization*. Figure **??** illustrates the basic concept. Suppose we are given a nonlinear next state function g. A Gaussian projected through this function is typically non-Gaussian. This is because nonlinearities in g distort the belief in ways that destroys its nice Gaussian shape, as illustrated in the figure. Linearization approximates g by a linear function that is tangent to g at the mean of the Gaussian. By projecting the Gaussian through this linear approximation, the posterior is Gaussian. In fact, once g is linearized, the mechanics of belief propagation are equivalent to those of the Kalman filter. The same argument applies to the multiplication of Gaussians when a measurement function h is involved. Again, the EKF approximates h by a linear function tangent to h, thereby retaining the Gaussian nature of the posterior belief.

There exist many techniques for linearizing nonlinear functions. EKFs utilize a method called (first order) *Taylor expansion*. Taylor expansion construct a linear approximation to a function g from g's value and slope. The slope is given by the partial derivative

$$g'(u_t, x_{t-1}) \quad := \quad \frac{\partial g(u_t, x_{t-1})}{\partial x_{t-1}} \tag{3.49}$$

Clearly, both the value of g and its slope depend on the argument of g. A logical choice for selecting the argument is to chose the state deemed most likely at the time of linearization. For Gaussians, the most likely state is the mean of the posterior μ_{t-1}. In other words, g is approximated by its value at μ_{t-1} (and at u_t), and the linear

extrapolation is achieved by a term proportional to the gradient of g at μ_{t-1} and u_t:

$$
\begin{aligned}
g(u_t, x_{t-1}) &\approx g(u_t, \mu_{t-1}) + \underbrace{g'(u_t, \mu_{t-1})}_{=:\, G_t} (x_{t-1} - \mu_{t-1}) \\
&= g(u_t, \mu_{t-1}) + G_t\, (x_{t-1} - \mu_{t-1}) \quad\quad\quad (3.50)
\end{aligned}
$$

Written as Gaussian, the next state probability is approximated as follows:

$$
\begin{aligned}
&p(x_t \mid u_t, x_{t-1}) \\
&\approx \det (2\pi R_t)^{-\frac{1}{2}} \exp \left\{ -\tfrac{1}{2} [x_t - g(u_t, \mu_{t-1}) - G_t\, (x_{t-1} - \mu_{t-1})]^T \right. \\
&\qquad \left. R_t^{-1} [x_t - g(u_t, \mu_{t-1}) - G_t\, (x_{t-1} - \mu_{t-1})] \right\} \quad\quad (3.51)
\end{aligned}
$$

Notice that G_t is a matrix of size $n \times n$, with n denoting the dimension of the state. This matrix is often called the *Jacobian*. The value of the Jacobian depends on u_t and μ_{t-1}, hence it differs for different points in time.

EKFs implement the exact same linearization for the measurement function h. Here the Taylor expansion is developed around $\bar{\mu}_t$, the state deemed most likely by the robot at the time it linearizes h:

$$
\begin{aligned}
h(x_t) &\approx h(\bar{\mu}_t) + \underbrace{h'(\bar{\mu}_t)}_{=:\, H_t} (x_t - \bar{\mu}_t) \\
&= h(\bar{\mu}_t) + H_t\, (x_t - \bar{\mu}_t) \quad\quad\quad\quad\quad (3.52)
\end{aligned}
$$

with $h'(x_t) = \frac{\partial h(x_t)}{\partial x_t}$. Written as a Gaussian, we have

$$
\begin{aligned}
p(z_t \mid x_t) &= \det (2\pi Q_t)^{-\frac{1}{2}} \exp \left\{ -\tfrac{1}{2} [z_t - h(\bar{\mu}_t) - H_t\, (x_t - \bar{\mu}_t)]^T \right. \\
&\qquad \left. Q_t^{-1} [z_t - h(\bar{\mu}_t) - H_t\, (x_t - \bar{\mu}_t)] \right\} \quad\quad (3.53)
\end{aligned}
$$

3.3.2   The EKF Algorithm

Table 3.3 states the EKF algorithm. In many ways, this algorithm is similar to the Kalman filter algorithm stated in Table 3.1. The most important differences are summarized by the following table:

1:      **Algorithm Extended_Kalman_filter**$(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$**:**
2:          $\bar{\mu}_t = g(u_t, \mu_{t-1})$
3:          $\bar{\Sigma}_t = G_t \, \Sigma_{t-1} \, G_t^T + R_t$
4:          $K_t = \bar{\Sigma}_t \, H_t^T (H_t \, \bar{\Sigma}_t \, H_t^T + Q_t)^{-1}$
5:          $\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$
6:          $\Sigma_t = (I - K_t \, H_t) \, \bar{\Sigma}_t$
7:      return μ_t, Σ_t

Table 3.3   The extended Kalman filter (EKF) algorithm.

	Kalman filter	EKF
state prediction (Line 2)	$A_t \, \mu_{t-1} + B_t \, u_t$	$g(u_t, \mu_{t-1})$
measurement prediction (Line 5)	$C_t \, \bar{\mu}_t$	$h(\bar{\mu}_t)$

That is, the linear predictions in Kalman filters are replaced by their nonlinear generalizations in EKFs. Moreover, EKFs use Jacobians G_t and H_t instead of the corresponding linear system matrices A_t, B_t, and C_t in Kalman filters. The Jacobian G_t corresponds to the matrices A_t and B_t, and the Jacobian H_t corresponds to C_t. A detailed example for extended Kalman filters will be given in Chapter **??**.

3.3.3   Mathematical Derivation of the EKF

The mathematical derivation of the EKF parallels that of the Kalman filter in Section 3.2.4, and hence shall only be sketched here. The prediction is calculated as follows (cf. (3.7)):

$$\overline{bel}(x_t) \;\; = \;\; \int \;\; \underbrace{p(x_t \mid x_{t-1}, u_t)}_{\sim \mathcal{N}(x_t; g(u_t, \mu_{t-1}) + G_t(x_{t-1} - \mu_{t-1}), R_t)} \;\; \underbrace{bel(x_{t-1})}_{\sim \mathcal{N}(x_{t-1}; \mu_{t-1}, \Sigma_{t-1})} \;\; dx_{t-1}$$

(3.54)

This distribution is the EKF analog of the prediction distribution in the Kalman filter, stated in (3.7). The Gaussian $p(x_t \mid x_{t-1}, u_t)$ can be found in Equation (3.51). The

function L_t is given by (cf. (3.10))

$$
\begin{aligned}
L_t &= \tfrac{1}{2}\,(x_t - g(u_t, \mu_{t-1}) - G_t(x_{t-1} - \mu_{t-1}))^T \\
&\quad R_t^{-1}\,(x_t - g(u_t, \mu_{t-1}) - G_t(x_{t-1} - \mu_{t-1})) \\
&\quad + \tfrac{1}{2}\,(x_{t-1} - \mu_{t-1})^T\,\Sigma_{t-1}^{-1}\,(x_{t-1} - \mu_{t-1})
\end{aligned}
\tag{3.55}
$$

which is quadratic in both x_{t-1} and x_t, as above. As in (3.11), we decompose L_t into $L_t(x_{t-1}, x_t)$ and $L_t(x_t)$:

$$
\begin{aligned}
&L_t(x_{t-1}, x_t) \\
&= \tfrac{1}{2}\,(x_{t-1} - \Phi_t\,[G_t^T\,R_t^{-1}\,(x_t - g(u_t, \mu_{t-1}) + G_t\mu_{t-1}) + \Sigma_{t-1}^{-1}\mu_{t-1}])^T\,\Phi^{-1} \\
&\quad (x_{t-1} - \Phi_t\,[G_t^T\,R_t^{-1}\,(x_t - g(u_t, \mu_{t-1}) + G_t\mu_{t-1}) + \Sigma_{t-1}^{-1}\mu_{t-1}])
\end{aligned}
\tag{3.56}
$$

with

$$
\Phi_t = (G_t^T\,R_t^{-1}\,G_t + \Sigma_{t-1}^{-1})^{-1}
\tag{3.57}
$$

and hence

$$
\begin{aligned}
L_t(x_t) &= \tfrac{1}{2}\,(x_t - g(u_t, \mu_{t-1}) + G_t\mu_{t-1})^T\,R_t^{-1}\,(x_t - g(u_t, \mu_{t-1}) + G_t\mu_{t-1}) \\
&\quad + \tfrac{1}{2}\,(x_{t-1} - \mu_{t-1})^T\,\Sigma_{t-1}^{-1}\,(x_{t-1} - \mu_{t-1}) \\
&\quad - \tfrac{1}{2}\,[G_t^T\,R_t^{-1}\,(x_t - g(u_t, \mu_{t-1}) + G_t\mu_{t-1}) + \Sigma_{t-1}^{-1}\mu_{t-1}]^T \\
&\quad \Phi_t\,[G_t^T\,R_t^{-1}\,(x_t - g(u_t, \mu_{t-1}) + G_t\mu_{t-1}) + \Sigma_{t-1}^{-1}\mu_{t-1}]
\end{aligned}
\tag{3.58}
$$

As the reader easily verifies, setting the first derivative of $L_t(x_t)$ to zero gives us the update $\mu_t = g(u_t, \mu_{t-1})$, in analogy to the derivation in Equations (3.26) through (3.30). The second derivative is given by $(R_t + G_t\,\Sigma_{t-1}\,G_t^T)^{-1}$ (see (3.31)).

The measurement update is also derived analogously to the Kalman filter in Section 3.2.4. In analogy to (3.32), we have for the EKF

$$
bel(x_t) = \eta \underbrace{p(z_t \mid x_t)}_{\sim \mathcal{N}(z_t; h(\bar{\mu}_t) + H_t\,(x_t - \bar{\mu}_t), Q_t)} \underbrace{\overline{bel}(x_t)}_{\sim \mathcal{N}(x_t; \bar{\mu}_t, \bar{\Sigma}_t)}
\tag{3.59}
$$

using the linearized next state transition function from (3.52). This leads to the exponent (see (3.34)):

$$
\begin{aligned}
J_t &= \tfrac{1}{2} \left(z_t - h(\bar{\mu}_t) - H_t \left(x_t - \bar{\mu}_t \right) \right)^T Q_t^{-1} \left(z_t - h(\bar{\mu}_t) - H_t \left(x_t - \bar{\mu}_t \right) \right) \\
&\quad + \tfrac{1}{2} (x_t - \bar{\mu}_t)^T \bar{\Sigma}_t^{-1} (x_t - \bar{\mu}_t)
\end{aligned}
\tag{3.60}
$$

The resulting mean and covariance is given by

$$
\begin{aligned}
\mu_t &= \bar{\mu}_t + K_t (z_t - h(\bar{\mu}_t)) & \text{(3.61)}\\
\Sigma_t &= (I - K_t \, H_t) \, \bar{\Sigma}_t & \text{(3.62)}
\end{aligned}
$$

with the Kalman gain

$$
K_t = \bar{\Sigma}_t \, H_t^T \left(H_t \, \bar{\Sigma}_{t-1} \, H_t^T + Q_t \right)^{-1}
\tag{3.63}
$$

The derivation of these equations is analogous to Equations (3.35) through (3.46).

3.3.4 Practical Considerations

The EKF has become just about the most popular tool for state estimation in robotics. Its strength lies in its simplicity and in its computational efficiency. As was the case for the Kalman filter, each update requires time $O(k^{2.8} + n^2)$, where k is the dimension of the measurement vector z_t, and n is the dimension of the state vector x_t. Other algorithms, such as the particle filter discussed further below, may require time exponential in n.

The EKF owes its computational efficiency to the fact that it represents the belief by a multivariate Gaussian distribution. A Gaussian is a unimodal distribution, which can be thought of as a single guess, annotated with an uncertainty ellipse. In many practical problems, Gaussians are robust estimators. Applications of the Kalman filter to state spaces with 1,000 dimensions or more will be discussed in later chapters of this book. EKFs have been applied with great success to a number of state estimation problems that violate the underlying assumptions.

Sometimes, one might want to pursue multiple distinct hypotheses. For example, a robot might have two distinct hypotheses as to where it is, but the arithmetic mean of

these hypotheses is not a likely contender. Such situations require multi-modal representations for the posterior belief. EKFs, in the form described here, are incapable of representing such multimodal beliefs. A common extension of EKFs is to represent posteriors using *mixtures*, or *sums*, of Gaussians. A mixture of J Gaussians may be of the form (cf. (**??**)):

$$bel(x) \;\; = \;\; \sum_j a_j \; \det \left(2\pi\Sigma_{j,t}\right)^{-\frac{1}{2}} \; \exp\left\{-\tfrac{1}{2}(x_t - \mu_{j,t})^T \, \Sigma_{j,t}^{-1} \, (x_t - \mu_{j,t})\right\}$$

(3.64)

where a_j are mixture parameters with $a_j \geq 0$ and $\sum_j a_j = 1$. EKFs that utilize such mixture representations are called *multi-hypothesis (extended) Kalman filters*, of MHEKF.

An important limitation of the EKF arises from the fact that it approximates state transitions and measurements using linear Taylor expansions. In virtually all robotics problems, these functions are nonlinear. The goodness of this approximation depends on two main factors. First, it depends on the degree of nonlinearity of the functions that are being approximated. If these functions are approximately linear, the EKF approximation may generally be a good one, and EKFs may approximate the posterior belief with sufficient accuracy. However, sometimes, the functions are not only nonlinear, but are also multi-modal, in which case the linearization may be a poor approximation. The goodness of the linearization also depends on the degree of uncertainty. The less certain the robot, the wider its Gaussian belief, and the more it is affected by nonlinearities in the state transition and measurement functions. In practice, when applying EKFs it is therefore important to keep the uncertainty of the state estimate small.

We also note that Taylor series expansion is only one way to linearize. Two other approaches have often been found to yield superior results. One is the *unscented Kalman filter*, which probes the function to be linearized at selected points and calculates a linearized approximation based on the outcomes of these probes. Another is known as *moments matching*, in which the linearization is calculated in a way that preserves the true mean and the true covariance of the posterior distribution (which is not the case for EKFs). Both techniques are relatively recent but appear to be superior to the EKF linearization.

3.4   THE INFORMATION FILTER

The dual of the Kalman filter is the information filter. Just like the KF and its nonlinear version, the EKF, the information filter (IF) represents the belief by a Gaussian. Thus, the standard information filter is subject to the same assumptions underlying the Kalman filter. The key difference between the KF and the IF arises from the way the Gaussian belief is represented. Whereas in the Kalman filter family of algorithms, Gaussians are represented by their moments (mean, covariance), information filters represent Gaussians in their canonical representation, which is comprised of an information matrix and an information vector. The difference in representation leads to different update equations. In particular, what is computationally complex in one representation happens to be simple in the other (and vice versa). The canonical and the moments representations are often considered *dual* to each other, and thus are the IF and the KF.

3.4.1   Canonical Representation

The canonical representation of a multivariate Gaussian is given by a matrix Ω and a vector ξ. The matrix Ω is the inverse of the covariance matrix:

$$\Omega = \Sigma^{-1}.$$
(3.65)

Ω is called the *information matrix*, or sometimes the *precision matrix*. The vector ξ is called the *information vector*. It is defined as

$$\xi = \Sigma^{-1}\mu.$$
(3.66)

It is easy to see that Ω and ξ are a complete parameterization of a Gaussian. In particular, the mean and covariance of the Gaussian can easily be obtained from the canonical representation by the inverse of (3.65) and (3.66):

$$\Sigma = \Omega^{-1}$$
(3.67)
$$\mu = \Omega^{-1}\xi$$
(3.68)

The canonical representation is often derived by multiplying out the exponent of a Gaussian. In (3.1), we defined the multivariate normal distribution as follows:

$$p(x) \;=\; \det\,(2\pi\Sigma)^{-\frac{1}{2}}\,\exp\left\{-\tfrac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right\} \tag{3.69}$$

A straightforward sequence of transformations leads to the following parameterization:

$$
\begin{aligned}
p(x) \;&=\; \det\,(2\pi\Sigma)^{-\frac{1}{2}}\,\exp\left\{-\tfrac{1}{2}x^T\Sigma^{-1}x + x^T\Sigma^{-1}\mu - \tfrac{1}{2}\mu^T\Sigma^{-1}\mu\right\} \\
&=\; \underbrace{\det\,(2\pi\Sigma)^{-\frac{1}{2}}\,\exp\left\{-\tfrac{1}{2}\mu^T\Sigma^{-1}\mu\right\}}_{\text{const.}}\,\exp\left\{-\tfrac{1}{2}x^T\Sigma^{-1}x + x^T\Sigma^{-1}\mu\right\}
\end{aligned}
\tag{3.70}
$$

The term labeled "const." does not depend on the target variable x. Hence, it can be subsumed into the normalizer η.

$$p(x) \;=\; \eta\,\exp\left\{-\tfrac{1}{2}x^T\,\Sigma^{-1}\,x + x^T\,\Sigma^{-1}\,\mu\right\} \tag{3.71}$$

This form motivates the parameterization of a Gaussian by its canonical parameters Ω and ξ.

$$p(x) \;=\; \eta\,\exp\left\{-\tfrac{1}{2}x^T\,\Omega\,x + x^T\,\xi\right\} \tag{3.72}$$

In many ways, the canonical representation is more elegant than the moments representation. In particular, the negative logarithm of the Gaussian (which plays an essential role in information theory) is a quadratic function in the canonical parameters Ω and ξ:

$$-\log p(x) \;=\; \text{const.} + \tfrac{1}{2}x^T\,\Omega\,x - x^T\,\xi \tag{3.73}$$

Here "const." is a constant. The reader may notice that we cannot use the symbol η to denote this constant, since negative logarithms of probabilities do not normalize to 1. The negative logarithm of our distribution $p(x)$ is quadratic in x, with the quadratic term parameterized by Ω and the linear term by ξ. In fact, for Gaussians, Ω must

---

1:        **Algorithm Information_filter**$(\xi_{t-1}, \Omega_{t-1}, u_t, z_t)$**:**
2:           $\bar{\Omega}_t = (A_t \ \Omega_{t-1}^{-1} \ A_t^T + R_t)^{-1}$
3:           $\bar{\xi}_t = \bar{\Omega}_t(A_t \ \Omega_{t-1}^{-1} \ \xi_{t-1} + B_t \ u_t)$
4:           $\Omega_t = C_t^T \ Q_t^{-1} \ C_t + \bar{\Omega}_t$
5:           $\xi_t = C_t^T \ Q_t^{-1} \ z_t + \bar{\xi}_t$
6:           return ξ_t, Ω_t

Table 3.4   The information filter (IF) algorithm.

be positive semidefinite, hence $-\log p(x)$ is a quadratic distance function with mean $\mu = \Omega^{-1} \ \xi$. This is easily verified by setting the first derivative of (3.73) to zero:

$$\frac{\partial[-\log p(x)]}{\partial x} \ = \ 0 \ \iff \ \Omega x - \xi \ = \ 0 \ \iff \ x = \Omega^{-1}\xi \qquad (3.74)$$

The matrix Ω determines the rate at which the distance function increases in the different dimensions of the variable x. A quadratic distance that is weighted by a matrix Ω is called *Mahalanobis distance*.

3.4.2   The Information Filter Algorithm

Table 3.4 states the update algorithm known as information filter. Its input is a Gaussian in its canonical representation ξ_{t-1} and Ω_{t-1}, representing the belief at time $t-1$. Just like all Bayes filters, its input includes the control u_t and the measurement z_t. The output are the parameters ξ_t and Ω_t of the updated Gaussian.

The update involves matrices A_t, B_t, C_t, R_t, and Q_t. Those were defined in Section 3.2. The information filter assumes that the state transition and measurement probabilities are governed by the following linear Gaussian equations, originally defined in (3.2) and (3.5):

$$x_t \ = \ A_t x_{t-1} + B_t u_t + \varepsilon_t \qquad (3.75)$$
$$z_t \ = \ C_t x_t + \delta_t \qquad (3.76)$$

Here R_t and Q_t are the covariances of the zero-mean noise variables ε_t and δ_t, respectively.

Just like the Kalman filter, the information filter is updated in two steps, a prediction step and a measurement update step. The prediction step is implemented in Lines 2 and 3 in Table 3.4. The parameters $\bar{\xi}_t$ and $\bar{\Omega}_t$ describe the Gaussian belief over x_t after incorporating the control u_t, but before incorporating the measurement z_t. The latter is done through Lines 4 and 5. Here the belief is updated based on the measurement z_t.

These two update steps can be vastly different in complexity, especially if the state space possesses many dimensions. The prediction step, as stated in Table 3.4, involves the inversion of two matrices of the size $n \times n$, where n is the dimension of the state space. This inversion requires approximately $O(n^{2.8})$ time. In Kalman filters, the update step is additive and requires at most $O(n^2)$ time; it requires less time if only a subset of variables is affected by a control, or if variables transition independently of each other. These roles are reversed for the measurement update step. Measurement updates are additive in the information filter. They require at most $O(n^2)$ time, and they are even more efficient if measurements carry only information about a subset of all state variables at a time. The measurement update is the difficult step in Kalman filters. It requires matrix inversion whose worst case complexity is $O(n^{2.8})$. This illustrates the dual character of Kalman and information filters.

3.4.3   Mathematical Derivation of the Information Filter

The derivation of the information filter is analogous to that of the Kalman filter. To derive the prediction step (Lines 2 and 3 in Table 3.4), we begin with the corresponding update equations of the Kalman filters, which can be found in Lines 2 and 3 of the algorithm in Table 3.1 and are restated here for the reader's convenience:

$$
\begin{aligned}
\bar{\mu}_t &= A_t\,\mu_{t-1} + B_t\,u_t & (3.77) \\
\bar{\Sigma}_t &= A_t\,\Sigma_{t-1}\,A_t^T + R_t & (3.78)
\end{aligned}
$$

The information filter prediction step follows now directly by substituting the moments μ and Σ by the canonical parameters ξ and Ω according to their definitions in

(3.67) and (3.72):

$$\mu_{t-1} = \Omega_{t-1}^{-1}\,\xi_{t-1}$$
$$\Sigma_{t-1} = \Omega_{t-1}^{-1} \qquad (3.79)$$

Substituting these expressions in (3.77) and (3.78) gives us the set of prediction equations

$$\bar\Omega_t = (A_t\,\Omega_{t-1}^{-1}\,A_t^T + R_t)^{-1} \qquad (3.80)$$
$$\bar\xi_t = \bar\Omega_t(A_t\,\Omega_{t-1}^{-1}\,\xi_{t-1} + B_t\,u_t) \qquad (3.81)$$

These equations are identical to those in Table 3.4. As is easily seen, the prediction step involves two nested inversions of a potentially large matrix. These nested inversions can be avoided when only a small number of state variables is affected by the motion update, a topic which will be discussed later in this book.

The derivation of the measurement update is even simpler. We begin with the Gaussian of the belief at time t, which was provided in Equation (3.34) and is restated here once again:

$$bel(x_t) \qquad\qquad (3.82)$$
$$= \eta\,\exp\left\{-\tfrac{1}{2}\,(z_t - C_t x_t)^T\,Q_t^{-1}\,(z_t - C_t x_t) - \tfrac{1}{2}\,(x_t - \bar\mu_t)^T\,\bar\Sigma_t^{-1}\,(x_t - \bar\mu_t)\right\}$$

For Gaussians represented in their canonical form this distribution is given by

$$bel(x_t) \qquad\qquad (3.83)$$
$$= \eta\,\exp\left\{-\tfrac{1}{2}\,x_t^T\,C_t^T\,Q_t^{-1}\,C_t\,x_t + x_t^T\,C_t^T\,Q_t^{-1}\,z_t - \tfrac{1}{2}\,x_t^T\,\bar\Omega_t x_t + x_t^T\bar\xi_t\right\}$$

which by reordering the terms in the exponent resolves to

$$bel(x_t) = \eta\,\exp\left\{-\tfrac{1}{2}\,x_t^T\,[C_t^T\,Q_t^{-1}\,C_t + \bar\Omega_t]\,x_t + x_t^T\,[C_t^T\,Q_t^{-1}\,z_t + \bar\xi_t]\right\}$$

We can now read off the measurement update equations, by collecting the terms in the squared brackets:

$$\xi_t = C_t^T\,Q_t^{-1}\,z_t + \bar\xi_t \qquad (3.84)$$

1:        **Algorithm Extended_information_filter$(\xi_{t-1}, \Omega_{t-1}, u_t, z_t)$:**
2:            $\mu_{t-1} = \Omega_{t-1}^{-1}\, \xi_{t-1}$
3:            $\bar{\Omega}_t = (G_t\, \Omega_{t-1}^{-1}\, G_t^T + R_t)^{-1}$
4:            $\bar{\xi}_t = \bar{\Omega}_t\, g(u_t, \mu_{t-1})$
5:            $\bar{\mu}_t = g(u_t, \mu_{t-1})$
6:            $\Omega_t = \bar{\Omega}_t + H_t^T\, Q_t^{-1}\, H_t$
7:            $\xi_t = \bar{\xi}_t + H_t^T\, Q_t^{-1}\, [z_t - h(\bar{\mu}_t) - H_t\, \bar{\mu}_t]$
8:        return ξ_t, Ω_t

Table 3.5   The extended information filter (EIF) algorithm.

$$\Omega_t \;\;=\;\; C_t^T\, Q_t^{-1}\, C_t + \bar{\Omega}_t \tag{3.85}$$

These equations are identical to the measurement update equations in Lines 4 and 5 of Table 3.4.

3.4.4   The Extended Information Filter Algorithm

The extended version of the information filter is analog to the EKF. Table 3.5 depicts the algorithm. The prediction is realized in Lines 2 through 4, and the measurement update in Lines 5 through 7. These update equations are largely analog to the linear information filter, with the functions g and h (and their Jacobian G_t and H_t) replacing the parameters of the linear model A_t, B_t, and C_t. As before, g and h specify the nonlinear next state function and measurement function, respectively. Those were defined in (3.47) and (3.48) and are restated here:

$$x_t \;\;=\;\; g(u_t, x_{t-1}) + \varepsilon_t \tag{3.86}$$
$$z_t \;\;=\;\; h(x_t) + \delta_t \,. \tag{3.87}$$

Unfortunately, both g and h require a state as an input. This mandates the recovery of a state estimate μ from the canonical parameters. The recovery takes place in Line 2, in which the state μ_{t-1} is calculated from Ω_{t-1} and ξ_{t-1} in the obvious way. Line 5 computes the state $\bar{\mu}_t$ using the equation familiar from the EKF (Line 2 in Table 3.3).

The necessity to recover the state estimate seems at odds with the desire to represent the filter using its canonical parameters. We will revisit this topic when discussing the use of extended information filters in the context of robotic mapping.

3.4.5 Mathematical Derivation of the Extended Information Filter

The extended information filter is easily derived by essentially performing the same linearization that led to the extended Kalman filter above. As in (3.50) and (3.52), EIFs approximate g and h by a Taylor expansion:

$$
\begin{aligned}
g(u_t, x_{t-1}) &\approx g(u_t, \mu_{t-1}) + G_t (x_{t-1} - \mu_{t-1}) & (3.88) \\
h(x_t) &\approx h(\bar{\mu}_t) + H_t (x_t - \bar{\mu}_t) & (3.89)
\end{aligned}
$$

Here G_t and H_t are the Jacobians of g and h at μ_{t-1} and $\bar{\mu}_t$, respectively:

$$
\begin{aligned}
G_t &= g'(u_t, \mu_{t-1}) & (3.90) \\
H_t &= h'(\bar{\mu}_t) & (3.91)
\end{aligned}
$$

These definitions are equivalent to those in the EKF. The prediction step is now derived from Lines 2 and 3 of the EKF algorithm (Table 3.3), which are restated here:

$$
\begin{aligned}
\bar{\Sigma}_t &= G_t \Sigma_{t-1} G_t^T + R_t & (3.92) \\
\bar{\mu}_t &= g(u_t, \mu_{t-1}) & (3.93)
\end{aligned}
$$

Substituting Σ_{t-1} by Ω_{t-1}^{-1} and $\bar{\mu}_t$ by $\bar{\Omega}_t^{-1} \bar{\xi}_t$ gives us the prediction equations of the extended information filter:

$$
\begin{aligned}
\bar{\Omega}_t &= (G_t \Omega_{t-1}^{-1} G_t^T + R_t)^{-1} & (3.94) \\
\bar{\xi}_t &= \bar{\Omega}_t \, g(u_t, \Omega_{t-1}^{-1} \xi_{t-1}) & (3.95)
\end{aligned}
$$

The measurement update is derived from Equations (3.59) and (3.60). In particular, (3.60) defines the following Gaussian posterior:

$$
bel(x_t) = \eta \, \exp\left\{ -\tfrac{1}{2} (z_t - h(\bar{\mu}_t) - H_t (x_t - \bar{\mu}_t))^T Q_t^{-1} \right.
$$

$$(z_t - h(\bar{\mu}_t) - H_t\,(x_t - \bar{\mu}_t)) - \tfrac{1}{2}(x_t - \bar{\mu}_t)^T \bar{\Sigma}_t^{-1}(x_t - \bar{\mu}_t) \} \quad (3.96)$$

Multiplying out the exponent and reordering the terms gives us the following expression for the posterior:

$$
\begin{aligned}
bel(x_t) &= \eta\ \exp\left\{-\tfrac{1}{2}\,x_t^T\,H_t^T\,Q_t^{-1}\,H_t\,x_t + x_t^T\,H_t^T\,Q_t^{-1}\,[z_t - h(\bar{\mu}_t) - H_t\,\bar{\mu}_t] \right.\\
&\qquad\left. -\tfrac{1}{2}x_t^T\,\bar{\Sigma}_t^{-1}\,x_t + x_t^T\,\bar{\Sigma}_t^{-1}\,\bar{\mu}_t\right\}\\
&= \eta\ \exp\left\{-\tfrac{1}{2}\,x_t^T\,\left[H_t^T\,Q_t^{-1}\,H_t + \bar{\Sigma}_t^{-1}\right]\,x_t \right.\\
&\qquad\left. + x_t^T\,\left[H_t^T\,Q_t^{-1}\,[z_t - h(\bar{\mu}_t) - H_t\,\bar{\mu}_t] + \bar{\Sigma}_t^{-1}\,\bar{\mu}_t\right]\right\} \quad (3.97)
\end{aligned}
$$

With $\bar{\Sigma}_t^{-1} = \bar{\Omega}_t$ this expression resolves to the following information form:

$$
\begin{aligned}
bel(x_t) &= \eta\ \exp\left\{-\tfrac{1}{2}\,x_t^T\,\left[H_t^T\,Q_t^{-1}\,H_t + \bar{\Omega}_t\right]\,x_t \right.\\
&\qquad\left. + x_t^T\,\left[H_t^T\,Q_t^{-1}\,[z_t - h(\bar{\mu}_t) - H_t\,\bar{\mu}_t] + \bar{\xi}_t\right]\right\} \quad (3.98)
\end{aligned}
$$

We can now read off the measurement update equations by collecting the terms in the squared brackets:

$$
\begin{aligned}
\Omega_t &= \bar{\Omega}_t + H_t^T\,Q_t^{-1}\,H_t & (3.99)\\
\xi_t &= \bar{\xi}_t + H_t^T\,Q_t^{-1}\,[z_t - h(\bar{\mu}_t) - H_t\,\bar{\mu}_t] & (3.100)
\end{aligned}
$$

3.4.6   Practical Considerations

When applied to robotics problems, the information filter possesses several advantages over the Kalman filter. For example, representing global uncertainty is simple in the information filter: simply set $\Omega = 0$. When using moments, such global uncertainty amounts to a covariance of infinite magnitude. This is especially problematic when sensor measurements carry information about a strict subset of all state variables, a situation often encountered in robotics. Special provisions have to be made to handle such situations in EKFs. Furthermore, the information filter tends to be numerically more stable than the Kalman filter in many of the applications discussed later in this book.

Another advantage of the information filter over the Kalman filter arises from its natural fit for multi-robot problems. Multi-robot problems often involve the integration

of sensor data collected decentrally. Such integration is commonly performed through Bayes rule. When represented in logarithmic form, Bayes rule becomes an addition. As noted above, the canonical parameters of information filters represent a probability in logarithmic form. Thus, information integration is achieved by summing up information from multiple robots. Addition is commutative. Because of this, information filters can often integrate information in arbitrary order, with arbitrary delays, and in a completely decentralized manner. While the same is possible using the moments representation—after all, they represent the same information—the necessary overhead for doing so is much higher. Despite this advantage, the use of information filters in multi-robot systems remains largely under-explored.

These advantages of the information filter are offset by important limitations. A primary disadvantage of the EIF is the need to recover a state estimate in the update step, when applied to nonlinear systems. This step, if implemented as stated here, requires the inversion of the information matrix. Further matrix inversions are required for the prediction step of the information filters. In many robotics problems, the EKF does not involve the inversion of matrices of comparable size. For high dimensional state spaces, the information filter is generally believed to be computationally inferior to the Kalman filter. In fact, this is one of the reasons why the EKF has been vastly more popular than the EIF.

As we will see later in this book, these limitations do not necessarily apply to problems in which the information matrix possess structure. In many robotics problems, the interaction of state variables is local; as a result, the information matrix may be sparse. Such sparseness does *not* translate to sparseness of the covariance.

Information filters can be thought of as graphs, where states are connected whenever the corresponding off-diagonal element in the information matrix is non-zero. Sparse information matrices correspond to sparse graphs; in fact, such graphs are commonly known as Gaussian Markov random fields. A flurry of algorithms exist to perform the basic update and estimation equations efficiently for such fields, under names like "loopy belief propagation." In this book, we will encounter a mapping problem in which the information matrix is (approximately) sparse, and develop an extended information filter that is significantly more efficient than both Kalman filters and nonsparse information filters.

3.5   SUMMARY

In this section, we introduced efficient Bayes filter algorithms that represent the posterior by multivariate Gaussians. We noted that

- Gaussians can be represented in two different ways: The moments representation and the canonical representation. The moments representation consists of the mean (first moment) and the covariance (second moment) of the Gaussian. The canonical, or natural, representation consists of an information matrix and an information vector. Both representations are duals of each other, and each can be recovered from the other via matrix inversion.

- Bayes filters can be implemented for both representations. When using the moments representation, the resulting filter is called Kalman filter. The dual of the Kalman filter is the information filter, which represents the posterior in the canonical representation. Updating a Kalman filter based on a control is computationally simple, whereas incorporating a measurement is more difficult. The opposite is the case for the information filter, where incorporating a measurement is simple, but updating the filter based on a control is difficult.

- For both filters to calculate the correct posterior, three assumptions have to be fulfilled. First, the initial belief must be Gaussian. Second, the state transition probability must be composed of a function that is linear in its argument with added independent Gaussian noise. Third, the same applies to the measurement probability. It must also be linear in its argument, with added Gaussian noise. Systems that meet these assumptions are called linear Gaussian systems.

- Both filters can be extended to nonlinear problems. The technique described in this chapter calculates a tangent to the nonlinear function. Tangents are linear, making the filters applicable. The technique for finding a tangent is called Taylor expansion. Performing a Taylor expansion involves calculating the first derivative of the target function, and evaluating it at a specific point. The result of this operation is a matrix known as the Jacobian. The resulting filters are called "extended."

- The accuracy of Taylor series expansions depends on two factors: The degree of nonlinearity in the system, and the width of the posterior. Extended filters tend to yield good results if the state of the system is known with relatively high accuracy, so that the remaining covariance is small. The larger the uncertainty, the higher the error introduced by the linearization.

- One of the primary advantages of Gaussian filters is computational: The update requires time polynomial in the dimensionality of the state space. This is not

the case of some of the techniques described in the next chapter. The primary disadvantage is their confinement to unimodal Gaussian distributions.

- Within the multivariate Gaussian regime, both filters, the Kalman filter and the information filter, have orthogonal strengths and weaknesses. However, the Kalman filter and its nonlinear extension, the extended Kalman filter, are vastly more popular than the information filter.

The selection of the material in this chapter is based on today's most popular techniques in robotics. There exists a huge number of variations and extensions of the Gaussian filters presented here, that address the various limitations and shortcomings. One of the most apparent limitations of the material presented thus far is the fact that the posterior is represented by a single Gaussian. This confines these filters to situations where the posterior can be described by a unimodal distribution. This is often appropriate in tracking applications, where a robot tracks a state variable with limited uncertainty. When uncertainty grows more global, a single mode can be insufficient, and Gaussians become too crude an approximation to the true posterior belief. This limitation has been well recognized, and even within the Gaussian paradigm extensions exist that can represent multimodal beliefs, e.g., using mixtures of Gaussians. Popular non-Gaussian approaches are described in the next chapter.

3.6 BIBLIOGRAPHICAL REMARKS

Inversion lemma: G.H. Golub, C.F. Van Loan, Matrix Computations, North Oxford Academic, 1986.

4

NONPARAMETRIC FILTERS

A popular alternative to Gaussian techniques are nonparametric filters. Nonparametric filters do not rely on a fixed functional form of the posterior, such as Gaussians. Instead, they approximate posteriors by a finite number of values, each roughly corresponding to a region in state space. Some nonparametric Bayes filters rely on a decomposition of the state space, in which each such value corresponds to the cumulative probability of the posterior density in a compact subregion of the state space. Others approximate the state space by random samples drawn from the posterior distribution. In all cases, the number of parameters used to approximate the posterior can be varied. The quality of the approximation depends on the number of parameters used to represent the posterior. As the number of parameters goes to infinity, nonparametric techniques tend to converge uniformly to the correct posterior (under specific smoothness assumptions).

This chapter discusses two nonparametric approaches for approximating posteriors over continuous spaces with finitely many values. The first decomposes the state space into finitely many regions, and represents the posterior by a histogram. A histogram assigns to each region a single cumulative probability; they are best thought of as piecewise constant approximations to a continuous density. The second technique represents posteriors by finitely many samples. The resulting filter is known as particle filter and has gained immense popularity in certain robotics problems.

Both types of techniques, histograms and particle filters, do not make strong parametric assumptions on the posterior density. In particular, they are well-suited to represent complex multimodal beliefs. For this reason, they are often the method of choice when a robot has to cope with phases of global uncertainty, and when it faces hard data association problems that yield separate, distinct hypotheses. However, the representational power of these techniques comes at the price of added computational com-

---

1:          **Algorithm Discrete_Bayes_filter($\{p_{k,t-1}\}, u_t, z_t$):**
2:               for all k do
3:                    $\bar{p}_{k,t} = \sum_i p(X_t = x_k \mid u_t, X_{t-1} = x_i) \, p_{i,t-1}$
4:                    $p_{k,t} = \eta \, p(z_t \mid X_t = x_k) \, \bar{p}_{k,t}$
5:               endfor
6:               return $\{p_{k,t}\}$

Table 4.1   The discrete Bayes filter. Here x_i, x_k denote individual states.

plexity. Fortunately, both nonparametric techniques described in this chapter make it possible to adapt the number of parameters to the (suspected) complexity of the posterior. When the posterior is of low complexity (e.g., focused on a single state with a small margin of uncertainty), they use only small numbers of parameters. For complex posteriors, e.g., posteriors with many modes scattered across the state space, the number of parameters grows larger.

Techniques that can adapt the number of parameters to represent the posterior online are called *adaptive*. They are called *resource-adaptive* if they can adapt based on the computational resources available for belief computation. Resource-adaptive techniques play an important role in robotics. They enable robots to make decisions in real time, regardless of the computational resources available. Particle filters are often implemented as a resource-adaptive algorithm, by adapting the number of particles online based on the available computational resources.

4.1   THE HISTOGRAM FILTER

Histogram filters decompose the state space into finitely many regions, and represent the cumulative posterior for each region by a single probability value. When applied to discrete spaces, such filters are known as *discrete Bayes filters*. In continuous state spaces, they are known as *histogram filters*. We will first describe the discrete Bayes filter, and then discuss its use in continuous state spaces.

4.1.1   The Discrete Bayes Filter Algorithm

Discrete Bayes filters apply to problems with *finite* state spaces, that is, where the random variable X_t can take on finitely many values. We already encountered a discrete Bayes filter in Section 2.4.2, when discussing the example of a robot estimating the probability that a door is open. Some of the robotic mapping problems discussed in later chapters also involve discrete random variables. For example, occupancy grid mapping algorithms assume that each location in the environment is either occupied or free. The corresponding random variable is binary, that is, it can take on two different values. Thus, finite state spaces play an important role in robotics.

Table 4.1 provides pseudo-code for the discrete Bayes filter. This code is derived from the general Bayes filter in Table 2.1 by replacing the integration with a finite sum. The variables x_i and x_k denote individual states, of which there may only be finitely many. The belief at time t is an assignment of a probability to each state x_k, denoted $p_{k,t}$. Thus, the input to the algorithm is a discrete probability distribution $\{p_{k,t}\}$, along with the most recent control u_t and measurement z_t. Line 3 calculates the prediction, the belief for the new state based on the control alone. This prediction is then updated in Line 4, so as to incorporate the measurement. The discrete Bayes filter algorithm is popular in many areas of signal processing (such as speech recognition), where it is often referred to as the forward pass of a hidden Markov models.

4.1.2   Continuous State

Of particular interest in this book will be the use of discrete Bayes filters as an approximate inference tool for *continuous* state spaces. Such filters are called *histogram filters*. Histogram filters decompose a continuous state space into finitely many regions:

$$\text{range}(X_t) \quad = \quad \mathbf{x}_{1,t} \cup \mathbf{x}_{2,t} \cup \ldots \mathbf{x}_{K,t} \tag{4.1}$$

Here X_t is the familiar random variable describing the state of the robot at time t. The function $\text{range}(X_t)$ denotes the state space, that is, the universe of possible values that X_t might assume. Each $\mathbf{x}_{k,t}$ describes a convex region. These regions together form a partitioning of the state space, that is, for each $i \neq k$ we have $\mathbf{x}_{i,t} \cap \mathbf{x}_{k,t} = \emptyset$ and $\bigcup_k \mathbf{x}_{k,t} = \text{range}(X_t)$. A straightforward decomposition of a continuous state space is a multi-dimensional grid, where each $\mathbf{x}_{k,t}$ is a grid cell. Through the granularity of the decomposition, we can trade off accuracy and computational efficiency. Fine-

grained decompositions infer smaller approximation errors than coarse ones, but at the expense of increased computational complexity.

As we already discussed, the discrete Bayes filter assigns to each region $\mathbf{x}_{k,t}$ a probability, $p_{k,t}$. Within each region, the discrete Bayes filter carries no further information on the belief distribution. Thus, the posterior becomes a piecewise constant PDF, which assigns a uniform probability to each state x_t within each region $\mathbf{x}_{k,t}$:

$$p(x_t) \quad = \quad \frac{p_{k,t}}{|\mathbf{x}_{k,t}|} \tag{4.2}$$

Here $|\mathbf{x}_{k,t}|$ is the volume of the region $\mathbf{x}_{k,t}$.

If the state space is truly discrete, the conditional probabilities $p(\mathbf{x}_{k,t} \mid u_t, \mathbf{x}_{i,t-1})$ and $p(z_t \mid \mathbf{x}_{k,t})$ are well-defined, and the algorithm can be implemented as stated. In continuous state spaces, one is usually given the densities $p(x_t \mid u_t, x_{t-1})$ and $p(z_t \mid x_t)$, which are defined for individual states (and not for regions in state space). For cases where each region $\mathbf{x}_{k,t}$ is small and of the same size, these densities are usually approximated by substituting $\mathbf{x}_{k,t}$ by a representative of this region. For example, we might simply "probe" using the mean state in $\mathbf{x}_{k,t}$

$$\hat{x}_{k,t} \quad = \quad |\mathbf{x}_{k,t}|^{-1} \int_{\mathbf{x}_{k,t}} x_t \; dx_t \tag{4.3}$$

One then simply replaces

$$p(z_t \mid \mathbf{x}_{k,t}) \quad \approx \quad p(z_t \mid \hat{x}_{k,t}) \tag{4.4}$$

$$p(\mathbf{x}_{k,t} \mid u_t, \mathbf{x}_{i,t-1}) \quad \approx \quad \frac{\eta}{|\mathbf{x}_{k,t}|} \; p(\hat{x}_{k,t} \mid u_t, \hat{x}_{i,t-1}) \tag{4.5}$$

These approximations are the result of the piecewise uniform interpretation of the discrete Bayes filter stated in (4.2), and a linearization similar to the one used by EKFs.

To see that (4.4) is a reasonable approximation, we note that $p(z_t \mid \mathbf{x}_{k,t})$ can be expressed as the following integral:

$$p(z_t \mid \mathbf{x}_{k,t}) \quad = \quad \frac{p(z_t, \mathbf{x}_{k,t})}{p(\mathbf{x}_{k,t})}$$

$$
= \frac{\int_{\mathbf{x}_{k,t}} p(z_t, x_t) \, dx_t}{\int_{\mathbf{x}_{k,t}} p(x_t) \, dx_t}
$$

$$
= \frac{\int_{\mathbf{x}_{k,t}} p(z_t \mid x_t) \, p(x_t) \, dx_t}{\int_{\mathbf{x}_{k,t}} p(x_t) \, dx_t}
$$

$$
\overset{(4.2)}{=} \frac{\int_{\mathbf{x}_{k,t}} p(z_t \mid x_t) \, \frac{p_{k,t}}{|\mathbf{x}_{k,t}|} \, dx_t}{\int_{\mathbf{x}_{k,t}} \frac{p_{k,t}}{|\mathbf{x}_{k,t}|} \, dx_t}
$$

$$
= \frac{\int_{\mathbf{x}_{k,t}} p(z_t \mid x_t) \, dx_t}{\int_{\mathbf{x}_{k,t}} 1 \, dx_t}
$$

$$
= |\mathbf{x}_{k,t}|^{-1} \int_{\mathbf{x}_{k,t}} p(z_t \mid x_t) \, dx_t \tag{4.6}
$$

This expression is an exact description of the desired probability under the piecewise uniform distribution model in (4.2). If we now approximate $p(z_t \mid x_t)$ by $p(z_t \mid \hat{x}_{k,t})$ for $x_t \in \mathbf{x}_{k,t}$, we obtain

$$
\begin{aligned}
p(z_t \mid \mathbf{x}_{k,t}) &\approx |\mathbf{x}_{k,t}|^{-1} \int_{\mathbf{x}_{k,t}} p(z_t \mid \hat{x}_{k,t}) \, dx_t \\
&= |\mathbf{x}_{k,t}|^{-1} \, p(z_t \mid \hat{x}_{k,t}) \int_{\mathbf{x}_{k,t}} 1 \, dx_t \\
&= |\mathbf{x}_{k,t}|^{-1} \, p(z_t \mid \hat{x}_{k,t}) \, |\mathbf{x}_{k,t}| \\
&= p(z_t \mid \hat{x}_{k,t})
\end{aligned} \tag{4.7}
$$

which is the approximation stated above in (4.4).

The derivation of the approximation to $p(\mathbf{x}_{k,t} \mid u_t, \mathbf{x}_{i,t-1})$ in (4.5) is slightly more involved, since regions occur on both sides of the conditioning bar. In analogy to our transformation above, we obtain:

$$
\begin{aligned}
&p(\mathbf{x}_{k,t} \mid u_t, \mathbf{x}_{i,t-1}) \\
&= \frac{p(\mathbf{x}_{k,t}, \mathbf{x}_{i,t-1} \mid u_t)}{p(\mathbf{x}_{i,t-1} \mid u_t)} \\
&= \frac{\int_{\mathbf{x}_{k,t}} \int_{\mathbf{x}_{i,t-1}} p(x_t, x_{t-1} \mid u_t) \, dx_t, dx_{t-1}}{\int_{\mathbf{x}_{i,t-1}} p(x_{t-1} \mid u_t) \, dx_{t-1}}
\end{aligned}
$$

$$= \frac{\int_{\mathbf{x}_{k,t}} \int_{\mathbf{x}_{i,t-1}} p(x_t \mid u_t, x_{t-1}) \, p(x_{t-1} \mid u_t) \, dx_t, dx_{t-1}}{\int_{\mathbf{x}_{i,t-1}} p(x_{t-1} \mid u_t) \, dx_{t-1}} \qquad (4.8)$$

We now exploit the Markov assumption, which implies independence between x_{t-1} and u_t, and thus $p(x_{t-1} \mid u_t) = p(x_{t-1})$:

$$p(\mathbf{x}_{k,t} \mid u_t, \mathbf{x}_{i,t-1})$$

$$= \frac{\int_{\mathbf{x}_{k,t}} \int_{\mathbf{x}_{i,t-1}} p(x_t \mid u_t, x_{t-1}) \, p(x_{t-1}) \, dx_t, dx_{t-1}}{\int_{\mathbf{x}_{i,t-1}} p(x_{t-1}) \, dx_{t-1}}$$

$$= \frac{\int_{\mathbf{x}_{k,t}} \int_{\mathbf{x}_{i,t-1}} p(x_t \mid u_t, x_{t-1}) \, \frac{p_{i,t-1}}{|\mathbf{x}_{i,t-1}|} \, dx_t, dx_{t-1}}{\int_{\mathbf{x}_{i,t-1}} \frac{p_{i,t-1}}{|\mathbf{x}_{i,t-1}|} \, dx_{t-1}}$$

$$= \frac{\int_{\mathbf{x}_{k,t}} \int_{\mathbf{x}_{i,t-1}} p(x_t \mid u_t, x_{t-1}) \, dx_t, dx_{t-1}}{\int_{\mathbf{x}_{i,t-1}} 1 \, dx_{t-1}}$$

$$= |\mathbf{x}_{i,t-1}|^{-1} \int_{\mathbf{x}_{k,t}} \int_{\mathbf{x}_{i,t-1}} p(x_t \mid u_t, x_{t-1}) \, dx_t, dx_{t-1} \qquad (4.9)$$

If we now approximate $p(x_t \mid u_t, x_{t-1})$ by $p(\hat{x}_{k,t} \mid u_t, \hat{x}_{i,t-1})$ as before, we obtain the following approximation. Note that the normalizer η becomes necessary to ensure that the approximation is a valid probability distribution:

$$p(\mathbf{x}_{k,t} \mid u_t, \mathbf{x}_{i,t-1})$$

$$\approx \eta \, |\mathbf{x}_{i,t-1}|^{-1} \int_{\mathbf{x}_{k,t}} \int_{\mathbf{x}_{i,t-1}} p(\hat{x}_{k,t} \mid u_t, \hat{x}_{i,t-1}) \, dx_t, dx_{t-1}$$

$$= \eta \, |\mathbf{x}_{i,t-1}|^{-1} \, p(\hat{x}_{k,t} \mid u_t, \hat{x}_{i,t-1}) \int_{\mathbf{x}_{k,t}} \int_{\mathbf{x}_{i,t-1}} 1 \, dx_t, dx_{t-1}$$

$$= \eta \, |\mathbf{x}_{i,t-1}|^{-1} \, p(\hat{x}_{k,t} \mid u_t, \hat{x}_{i,t-1})|\mathbf{x}_{k,t}| \, |\mathbf{x}_{i,t-1}|$$

$$= \eta \, |\mathbf{x}_{k,t}| \, p(\hat{x}_{k,t} \mid u_t, \hat{x}_{i,t-1}) \qquad (4.10)$$

If all regions are of equal size (meaning that $|\mathbf{x}_{k,t}|$ is the same for all k), we can simply omit the factor $|\mathbf{x}_{k,t}|$, since it is subsumed by the normalizer. The resulting discrete Bayes filter is then equivalent to the algorithm outlined in Table 4.1. If implemented as stated there, the auxiliary parameters $\bar{p}_k$ do not constitute a probability distribution, since they are not normalized (compare Line 3 to (4.10)). However, normalization takes place in Line 4, so that the output parameters are indeed a valid probability distribution.

4.1.3 Decomposition Techniques

In robotics, decomposition techniques of continuous state spaces come into two basic flavors: *static* and *dynamic*. Static techniques rely on a fixed decomposition that is chosen in advance, irrespective of the shape of the posterior that is being approximated. Dynamic techniques adapt the decomposition to the specific shape of the posterior distribution. Static techniques are usually easier to implement, but they can be wasteful with regards to computational resources.

A primary example of a dynamic decomposition technique is the family of *density trees*. Density trees decompose the state space recursively, in ways that adapt the resolution to the posterior probability mass. The intuition behind this decomposition is that the level of detail in the decomposition is a function of the posterior probability: The less likely a region, the coarser the decomposition. Figure 4.1 illustrates the difference between a static grid representation and a density tree representation for a two-dimensional probability density. While both representations have the same approximation quality, the density tree is more compact than the grid representation. Dynamic techniques like density trees can often cut the computation complexity by orders of magnitude over static ones, yet they require additional implementation effort.

An effect similar to that of dynamic decompositions can be achieved by *selective updating*. When updating a posterior represented by a grid, selective techniques update a fraction of all grid cells only. A common implementation of this idea updates only those grid cells whose posterior probability exceeds a user-specified threshold. Selective updating can be viewed as a hybrid decomposition, which decomposes the state space into a fine grained grid and one large set that contains all regions not chosen by the selective update procedure. In this light, it can be thought of as a dynamic decomposition technique, since the decision as to which grid cells to consider during the update is made online, based on the shape of the posterior distribution. Selective updating techniques can reduce the computational effort involved in updating beliefs by many orders of magnitude. They make it possible to use grid decompositions in spaces of three or more dimensions.

The mobile robotics literature often distinguishes topological from metric representations of space. While no clear definition of these terms exist, topological representations are often thought of as course graph-like representations, where nodes in the graph correspond to significant places (or features) in the environment. For indoor environments, such places may correspond to intersections, T-junctions, dead ends, and so on. The resolution of such decompositions, thus, depends on the structure of the environment. Alternatively, one might decompose the state space using regularly-
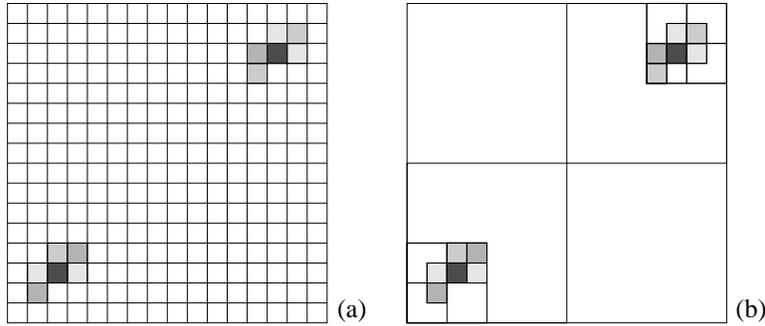
Figure 4.1   (a) Grid based representation of a two-dimensional probability density. The probability density concentrates on the upper right and lower left part of the state space. (b) Density tree representation of the same distribution.

spaced grids. Such a decomposition does not depend on the shape and location of the environmental features. Grid representations are often thought of as metric although, strictly speaking, it is the embedding space that is metric, not the decomposition. In mobile robotics, the spatial resolution of grid representations tends to be higher than that of topological representations. For instance, some of the examples in Chapter 7 use grid decompositions with cell sizes of 10 centimeters or less. This increased accuracy comes at the expense of increased computational costs.

4.1.4   Binary Bayes Filters With Static State

Certain problems in robotics are best formulated as estimation problems with binary state that does not change over time. Problems of this type arise if a robot estimates a fixed binary quantity in the environment from a sequence of sensor measurements. Since the state is static, the belief is a function of the measurements:

$$bel_t(x) \quad = \quad p(x \mid z_{1:t}, u_{1:t}) \quad = \quad p(x \mid z_{1:t}) \tag{4.11}$$

where the state is chosen from two possible values, denoted by x and $\neg x$. In particular, we have $bel_t(\neg x) = 1 - bel_t(x)$. The lack of a time index for the state x reflects the fact that the state does not change.

Naturally, binary estimation problems of this type can be tackled using the discrete Bayes filter in Table 4.1. However, the belief is commonly implemented as a *log odds ratio*. The *odds* of a state x is defined as the ratio of the probability of this event

---

1:       **Algorithm binary_Bayes_filter(l_{t-1}, z_t):**
2:           $l_t = l_{t-1} + \log \frac{p(x|z_t)}{1-p(x|z_t)} - \log \frac{p(x)}{1-p(x)}$
3:           return l_t

Table 4.2   The binary Bayes filter in log odds form with an inverse measurement model. Here l_t is the log odds of the posterior belief over a binary state variable that does not change over time.

divided by the probability of its negate

$$\frac{p(x)}{p(\neg x)} \quad = \quad \frac{p(x)}{1 - p(x)} \tag{4.12}$$

The log odds is the logarithm of this expression

$$l(x) \quad := \quad \log \frac{p(x)}{1 - p(x)} . \tag{4.13}$$

Log odds assume values from $-\infty$ to ∞. The Bayes filter for updating beliefs in log odds representation is computationally elegant. It avoids truncation problems that arise for probabilities close to 0 or 1.

Table 4.2 provides the basic update algorithm. This algorithm is additive; in fact, any algorithm that increment and decrement a variable in response to measurements can be interpreted as a Bayes filter in log odds form. This binary Bayes filter uses an *inverse measurement model* $p(x \mid z_t)$, instead of the familiar forward model $p(z_t \mid x)$. The inverse measurement model specifies a distribution over the (binary) state variable as a function of the measurement z_t. Inverse models are often used in situations where measurements are more complex than the binary state. An example of such a situation is the problem of estimating whether or not a door is closed, from camera images. Here the state is extremely simple, but the space of all measurements is huge. It is easier to devise a function that calculates a probability of a door being closed from a camera image, than describing the distribution over all camera images that show a closed door. In other words, it is easier to implement an inverse than a forward sensor model.

As the reader easily verifies from our definition of the log odds (4.13), the belief $bel_t(x)$ can be recovered from the log odds ratio l_t by the following equation:

$$bel_t(x) \;=\; 1 - \frac{1}{1 + \exp\{l_t\}} \tag{4.14}$$

To verify the correctness of our binary Bayes filter algorithm, we briefly restate the basic filter equation with the Bayes normalizer made explicit:

$$\begin{aligned}
p(x \mid z_{1:t}) &= \frac{p(z_t \mid x, z_{1:t-1})\, p(x \mid z_{1:t-1})}{p(z_t \mid z_{1:t-1})} \\[2mm]
&= \frac{p(z_t \mid x)\, p(x \mid z_{1:t-1})}{p(z_t \mid z_{1:t-1})}
\end{aligned} \tag{4.15}$$

We now apply Bayes rule to the measurement model $p(z_t \mid x)$:

$$p(z_t \mid x) \;=\; \frac{p(x \mid z_t)\, p(z_t)}{p(x)} \tag{4.16}$$

and obtain

$$p(x \mid z_{1:t}) \;=\; \frac{p(x \mid z_t)\, p(z_t)\, p(x \mid z_{1:t-1})}{p(x)\, p(z_t \mid z_{1:t-1})}. \tag{4.17}$$

By analogy, we have for the opposite event $\neg x$:

$$p(\neg x \mid z_{1:t}) \;=\; \frac{p(\neg x \mid z_t)\, p(z_t)\, p(\neg x \mid z_{1:t-1})}{p(\neg x)\, p(z_t \mid z_{1:t-1})} \tag{4.18}$$

Dividing (4.17) by (4.18) leads to cancellation of various difficult-to-calculate probabilities:

$$\begin{aligned}
\frac{p(x \mid z_{1:t})}{p(\neg x \mid z_{1:t})} &= \frac{p(x \mid z_t)}{p(\neg x \mid z_t)} \frac{p(x \mid z_{1:t-1})}{p(\neg x \mid z_{1:t-1})} \frac{p(\neg x)}{p(x)} \\[2mm]
&= \frac{p(x \mid z_t)}{1 - p(x \mid z_t)} \frac{p(x \mid z_{1:t-1})}{1 - p(x \mid z_{1:t-1})} \frac{1 - p(x)}{p(x)}
\end{aligned} \tag{4.19}$$

We denote the log odds ratio of the belief $bel_t(x)$ by $l_t(x)$. The log odds belief at time t is given by the logarithm of (4.19).

$$
\begin{aligned}
l_t(x) &= \log \frac{p(x \mid z_t)}{1 - p(x \mid z_t)} + \log \frac{p(x \mid z_{1:t-1})}{1 - p(x \mid z_{1:t-1})} + \log \frac{1 - p(x)}{p(x)} \\
&= \log \frac{p(x \mid z_t)}{1 - p(x \mid z_t)} - \log \frac{p(x)}{1 - p(x)} + l_{t-1}(x)
\end{aligned}
\tag{4.20}
$$

Here $p(x)$ is the *prior* probability of the state x. As (4.20), each measurement update involves the addition of the prior (in log odds form). The prior also defines the log odds of the initial belief before processing any sensor measurement:

$$
l_0(x) = \log \frac{1 - p(x)}{p(x)}
\tag{4.21}
$$

4.2 THE PARTICLE FILTER

4.2.1 Basic Algorithm

The particle filter is an alternative nonparametric implementation of the Bayes filter. Just like histogram filters, particle filters approximate the posterior by a finite number of parameters. However, they differ in the way these parameters are generated, and in which they populate the state space. The key idea of the particle filter is to represent the posterior $bel(x_t)$ by a set of random state samples drawn from this posterior. Figure **??** illustrates this idea for a Gaussian. Instead of representing the distribution by a parametric form (the exponential function that defines the density of a normal distribution), particle filters represent a distribution by a set of samples drawn from this distribution. Such a representation is approximate, but it is nonparametric, and therefore can represent a much broader space of distributions than, for example, Gaussians.

In particle filters, the samples of a posterior distribution are called *particles* and are denoted

$$
\mathcal{X}_t := x_t^{[1]}, x_t^{[2]}, \ldots, x_t^{[M]}
\tag{4.22}
$$

Each particle $x_t^{[m]}$ (with $1 \le m \le M$) is a concrete instantiation of the state at time t, that is, a hypothesis as to what the true world state may be at time t. Here M denotes

```
1:      Algorithm Particle_filter($\mathcal{X}_{t-1}, u_t, z_t$):
2:          $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
3:          for $m = 1$ to $M$ do
4:              sample $x_t^{[m]} \sim p(x_t \mid u_t, x_{t-1}^{[m]})$
5:              $w_t^{[m]} = p(z_t \mid x_t^{[m]})$
6:              $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
7:          endfor
8:          for $m = 1$ to $M$ do
9:              draw $i$ with probability $\propto w_t^{[i]}$
10:             add $x_t^{[i]}$ to $\mathcal{X}_t$
11:         endfor
12:         return $\mathcal{X}_t$
```

Table 4.3   The particle filter algorithm, a variant of the Bayes filter based on importance sampling.

the number of particles in the particle set $\mathcal{X}_t$. In practice, the number of particles M is often a large number, e.g., $M = 1,000$. In some implementations M is a function of t or of other quantities related to the belief $bel(x_t)$.

The intuition behind particle filters is to approximate the belief $bel(x_t)$ by the set of particles $\mathcal{X}_t$. Ideally, the likelihood for a state hypothesis x_t to be included in the particle set $\mathcal{X}_t$ shall be proportional to its Bayes filter posterior $bel(x_t)$:

$$x_t^{[m]} \quad \sim \quad p(x_t \mid z_{1:t}, u_{1:t}) \tag{4.23}$$

As a consequence of (4.23), the denser a subregion of the state space is populated by samples, the more likely it is that the true state falls into this region. As we will discuss below, the property (4.23) holds only asymptotically for $M \uparrow \infty$ for the standard particle filter algorithm. For finite M, particles are drawn from a slightly different distribution. In practice, this difference is negligible as long as the number of particles is not too small (e.g., $M \geq 100$).

Just like all other Bayes filter algorithms discussed thus far, the particle filter algorithm constructs the belief $bel(x_t)$ recursively from the belief $bel(x_{t-1})$ one time step earlier. Since beliefs are represented by sets of particles, this means that particle filters

construct the particle set $\mathcal{X}_t$ recursively from the set $\mathcal{X}_{t-1}$. The most basic variant of the particle filter algorithm is stated in Table 4.3. The input of this algorithm is the particle set $\mathcal{X}_{t-1}$, along with the most recent control u_t and the most recent measurement z_t. The algorithm then first constructs a temporary particle set $\bar{\mathcal{X}}$ which is reminiscent (but not equivalent) to the belief $\overline{bel}(x_t)$. It does this by systematically processing each particle $x_{t-1}^{[m]}$ in the input particle set $\mathcal{X}_{t-1}$ as follows.

1. Line 4 generates a hypothetical state $x_t^{[m]}$ for time t based on the particle $x_{t-1}^{[m]}$ and the control u_t. The resulting sample is indexed by m, indicating that it is generated from the m-th particle in $\mathcal{X}_{t-1}$. This step involves sampling from the next state distribution $p(x_t \mid u_t, x_{t-1})$. To implement this step, one needs to be able to sample from $p(x_t \mid u_t, x_{t-1})$. The ability to sample from the state transition probability is not given for arbitrary distributions $p(x_t \mid u_t, x_{t-1})$. However, many major distributions in this book possess efficient algorithms for generating samples. The set of particles resulting from iterating Step 4 M times is the filter's representation of $\overline{bel}(x_t)$.

2. Line 5 calculates for each particle $x_t^{[m]}$ the so-called *importance factor*, denoted $w_t^{[m]}$. Importance factors are used to incorporate the measurement z_t into the particle set. The importance, thus, is the probability of the measurement z_t under the particle $x_t^{[m]}$, that is, $w_t^{[m]} = p(z_t \mid x_t^{[m]})$. If we interpret $w_t^{[m]}$ as the *weight* of a particle, the set of weighted particles represents (in approximation) the Bayes filter posterior $bel(x_t)$.

3. The real "trick" of the particle filter algorithm occurs in Lines 8 through 11 in Table 4.3. These lines implemented what is known as *resampling* or *importance resampling*. The algorithm draws with replacement M particles from the temporary set $\bar{\mathcal{X}}_t$. The probability of drawing each particle is given by its importance weight. Resampling transforms a particle set of M particles into another particle set of the same size. By incorporating the importance weights into the resampling process, the distribution of the particles change: whereas before the resampling step, they were distribution according to $\overline{bel}(x_t)$, after the resampling they are distributed (approximately) according to the posterior $bel(x_t) = \eta \, p(z_t \mid x_t^{[m]}) \overline{bel}(x_t)$. In fact, the resulting sample set usually possesses many duplicates, since particles are drawn with replacement. More important are the particles that are *not* contained in $\mathcal{X}_t$: those tend to be the particles with lower importance weights.

The resampling step has the important function to force particles back to the posterior $bel(x_t)$. In fact, an alternative (and usually inferior) version of the particle filter would never resample, but instead would maintain for each particle an importance weight

that is initialized by 1 and updated multiplicatively:

$$w_t^{[m]} \;\; = \;\; p(z_t \mid x_t^{[m]}) \, w_{t-1}^{[m]} \tag{4.24}$$

Such a particle filter algorithm would still approximate the posterior, but many of its particles would end up in regions of low posterior probability. As a result, it would require many more particles; how many depends on the shape of the posterior. The resampling step is a probabilistic implementation of the Darwinian idea of *survival of the fittest*: It refocuses the particle set to regions in state space with high posterior probability. By doing so, it focuses the computational resources of the filter algorithm to regions in the state space where they matter the most.

4.2.2   Importance Sampling

For the derivation of the particle filter, it shall prove useful to discuss the resampling step in more detail. Figure 4.2 illustrates the intuition behind the resampling step. Figure 4.2a shows a density function f of a probability distribution called the *target distribution*. What we would like to achieve is to obtain a sample from f. However, sampling from f directly may not be possible. Instead, we can generate particles from a related density, labeled g in Figure 4.2b. The distribution that corresponds to the density g is called *proposal distribution*. The density g must be such that $f(x) > 0$ implies $g(x) > 0$, so that there is a non-zero probability to generate a particle when sampling from g for any state that might be generated by sampling from f. However, the resulting particle set, shown at the bottom of Figure 4.2b, is distributed according to g, not to f. In particular, for any interval $A \subseteq \mathrm{range}(X)$ (or more generally, any Borel set A) the empirical count of particles that fall into A converges to the integral of g under A:

$$\frac{1}{M} \sum_{m=1}^{M} I(x^{[m]} \in A) \;\; \longrightarrow \;\; \int_A g(x) \; dx \tag{4.25}$$

To offset this difference between f and g, particles $x^{[m]}$ are weighted by the quotient

$$w^{[m]} \;\; = \;\; \frac{f(x^{[m]})}{g(x^{[m]})} \tag{4.26}$$
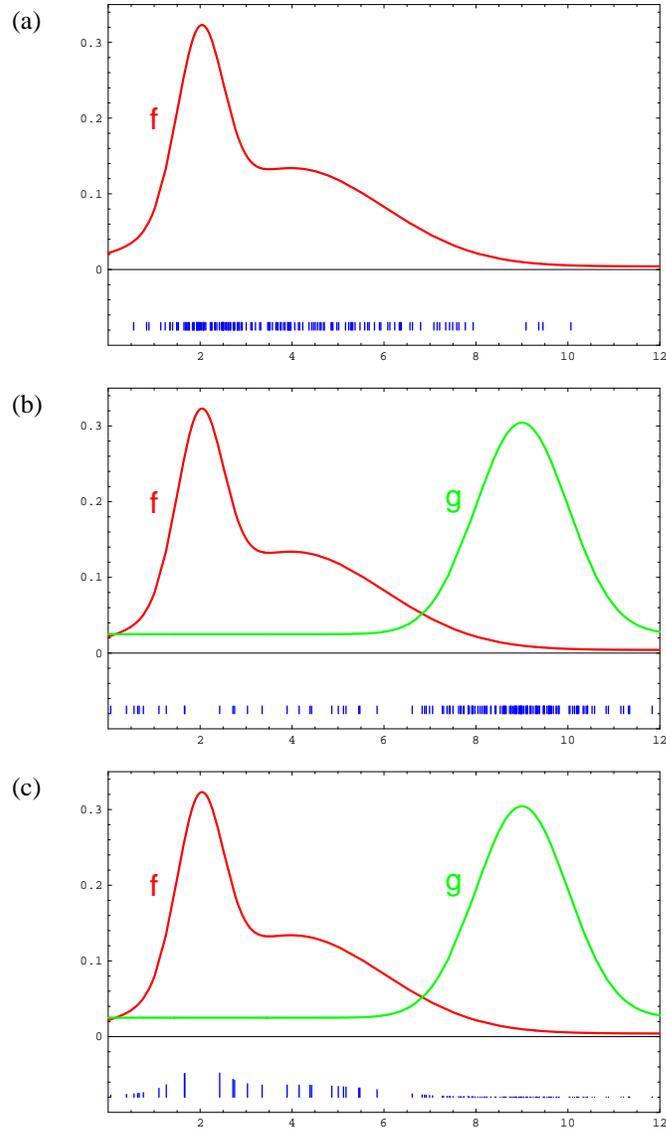
Figure 4.2 Illustration of importance factors in particle filters: (a) We seek to approximate the target density f. (b) Instead of sampling from f directly, we can only generate samples from a different density, g. Samples drawn from g are shown at the bottom of this diagram. (c) A sample of f is obtained by attaching the weight $f(x)/g(x)$ to each sample x. In particle filters, f corresponds to the belief $bel(x_t)$ and g to the belief $\overline{bel}(x_t)$.

This is illustrated by Figure 4.2c: The vertical bars in this figure indicate the magnitude of the importance weights. Importance weights are the non-normalized probability mass of each particle. In particular, we have

$$\left[\sum_{m=1}^{M} w^{[m]}\right]^{-1} \sum_{m=1}^{M} I(x^{[m]} \in A) \, w^{[m]} \quad \longrightarrow \quad \int_A f(x) \, dx \qquad (4.27)$$

where the first term serves as the normalizer for all importance weights. In other words, even though we generated the particles from the density g, the appropriately weighted particles converge to the density f.

The specific convergence involves an integration over a set A. Clearly, a particle set represents a discrete distribution, whereas f is continuous in our example. Because of this, there is no density that could be associated with a set of particles. The convergence, thus, is over the cumulative distribution function of f, not the density itself (hence the integration over A). A nice property of importance sampling is that it converges to the true density if $g(x) > 0$ whenever $f(x) > 0$. In most cases, the rate of convergence is in $O(\frac{1}{\sqrt{M}})$, where M is the number of samples. The constant factor depends on the similarity of $f(s)$ and $g(s)$.

In particle filters, the density f corresponds to the target belief $bel(x_t)$. Under the (asymptotically correct) assumption that the particles in $\mathcal{X}_{t-1}$ are distributed according to $bel(x_{t-1})$, the density g corresponds to the product distribution:

$$p(x_t \mid u_t, x_{t-1}) \, bel(x_{t-1}) \qquad (4.28)$$

This distribution is called the *proposal distribution*.

4.2.3   Mathematical Derivation of the PF

To derive particle filters mathematically, it shall prove useful to think of particles as samples of state sequences

$$x_{0:t}^{[m]} \quad = \quad x_0^{[m]}, x_1^{[m]}, \ldots, x_t^{[m]} \qquad (4.29)$$

It is easy to modify the algorithm accordingly: Simply append to the particle $x_t^{[m]}$ the sequence of state samples from which it was generated $x_{0:t-1}^{[m]}$. This particle filter

calculates the posterior over all state sequences:

$$bel(x_{0:t}) \quad = \quad p(x_{0:t} \mid u_{1:t}, z_{1:t}) \tag{4.30}$$

instead of the belief $bel(x_t) = p(x_t \mid u_{1:t}, z_{1:t})$. Admittedly, the space over all state sequences is huge, and covering it with particles is usually plainly infeasible. However, this shall not deter us here, as this definition serves only as the means to derive the particle filter algorithm in Table 4.2.

The posterior $bel(x_{0:t})$ is obtained analogously to the derivation of $bel(x_t)$ in Section 2.4.3. In particular, we have

$$
\begin{aligned}
p(x_{0:t} &\mid z_{1:t}, u_{1:t}) \\
&\stackrel{\text{Bayes}}{=} \eta\, p(z_t \mid x_{0:t}, z_{1:t-1}, u_{1:t})\, p(x_{0:t} \mid z_{1:t-1}, u_{1:t}) \\
&\stackrel{\text{Markov}}{=} \eta\, p(z_t \mid x_t)\, p(x_{0:t} \mid z_{1:t-1}, u_{1:t}) \\
&= \eta\, p(z_t \mid x_t)\, p(x_t \mid x_{0:t-1}, z_{1:t-1}, u_{1:t})\, p(x_{0:t-1} \mid z_{1:t-1}, u_{1:t}) \\
&\stackrel{\text{Markov}}{=} \eta\, p(z_t \mid x_t)\, p(x_t \mid x_{t-1}, u_t)\, p(x_{0:t-1} \mid z_{1:t-1}, u_{1:t-1}) \tag{4.31}
\end{aligned}
$$

Notice the absence of integral signs in this derivation, which is the result of maintaining all states in the posterior, not just the most recent one as in Section 2.4.3.

The derivation is now carried out by induction. The initial condition is trivial to verify, assuming that our first particle set is obtained by sampling the prior $p(x_0)$. Let us assume that the particle set at time $t-1$ is distributed according to $bel(x_{0:t-1})$. For the m-th particle $x_{0:t-1}^{[m]}$ in this set, the sample $x_t^{[m]}$ generated in Step 4 of our algorithm is generated from the proposal distribution:

$$
\begin{aligned}
p(x_t \mid x_{t-1}, u_t)\, &bel(x_{0:t-1}) \\
&= p(x_t \mid x_{t-1}, u_t)\, p(x_{0:t-1} \mid z_{0:t-1}, u_{0:t-1}) \tag{4.32}
\end{aligned}
$$

With

$$
\begin{aligned}
w_t^{[m]} &= \frac{\text{target distribution}}{\text{proposal distribution}} \\
&= \frac{\eta\, p(z_t \mid x_t)\, p(x_t \mid x_{t-1}, u_t)\, p(x_{0:t-1} \mid z_{1:t-1}, u_{1:t-1})}{p(x_t \mid x_{t-1}, u_t)\, p(x_{0:t-1} \mid z_{0:t-1}, u_{0:t-1})} \\
&= \eta\, p(z_t \mid x_t) \tag{4.33}
\end{aligned}
$$

The constant η plays no role since the resampling takes place with probabilities *proportional* to the importance weights. By resampling particles with probability proportional to $w_t^{[m]}$, the resulting particles are indeed distributed according to the product of the proposal and the importance weights $w_t^{[m]}$:

$$\eta \; w_t^{[m]} \; p(x_t \mid x_{t-1}, u_t) \; p(x_{0:t-1} \mid z_{0:t-1}, u_{0:t-1}) \quad = \quad bel(x_{0:t}) \qquad (4.34)$$

(Notice that the constant factor η here differs from the one in (4.33).) The algorithm in Table 4.2 follows now from the simple observation that if $x_{0:t}^{[m]}$ is distributed according to $bel(x_{0:t})$, then the state sample $x_t^{[m]}$ is (trivially) distributed according to $bel(x_t)$.

As we will argue below, this derivation is only correct for $M \longrightarrow \infty$, due to a laxness in our consideration of the normalization constants. However, even for finite M it explains the intuition behind the particle filter.

4.2.4   Properties of the Particle Filter

Particle filters are approximate and as such subject to approximation errors. There are four complimentary sources of approximation error, each of which gives rise to improved versions of the particle filter.

1. The first approximation error relates to the fact that only finitely many particles are used. This artifact introduces a systematic *bias* in the posterior estimate. To see, consider the extreme case of $M = 1$ particle. In this case, the loop in Lines 3 through 7 in Table 4.3 will only be executed once, and $\bar{\mathcal{X}}_t$ will contain only a single particle, sampled from the motion model. The key insight is that the resampling step (Lines 8 through 11 in Table 4.3) will now *deterministically* accept this sample, regardless of its importance factor $w_t^{[m]}$. Put differently, the measurement probability $p(z_t \mid x_t^{[m]})$ plays no role in the result of the update, and neither does z_t. Thus, if $M = 1$, the particle filter generates particles from the probability

$$p(x_t \mid u_{1:t}) \qquad\qquad\qquad\qquad\qquad\qquad (4.35)$$

   instead of the desired posterior $p(x_t \mid u_{1:t}, z_{1:t})$. It flatly ignores all measurements. How can this happen?

   The culprit is the normalization, implicit in the resampling step. When sampling in proportion to the importance weights (Line 9 of the algorithm), $w_t^{[m]}$ becomes

its own normalizer if $M = 1$:

$$p(\text{draw } x_t^{[m]} \text{ in Line 9}) \quad = \quad \frac{w_t^{[m]}}{w_t^{[m]}} \quad = 1 \tag{4.36}$$

In general, the problem is that the non-normalized values $w_t[m]$ are drawn from an M-dimensional space, but after normalization they reside in a space of dimension $M-1$. This is because after normalization, the m-th weight can be recovered from the $M-1$ other weights by subtracting those from 1. Fortunately, for larger values of M, the effect of loss of dimensionality, or degrees of freedom, becomes less and less pronounced.

2. A second source of error in the particle filter relates to the randomness introduced in the resampling phase. To understand this error, it will once again be useful to consider the extreme case, which is that of a robot whose state does not change. Sometimes, we know for a fact that $x_t = x_{t-1}$. A good example is that of mobile robot localization, for a non-moving robot. Let us furthermore assume that the robot possesses no sensors, hence it cannot estimate the state, and that it is unaware of the state. Initially, our particle set $\mathcal{X}_0$ will be generated from the prior; hence particles will be spread throughout the state space. The random nature of the resampling step (Line 8 in the algorithm) will regularly fail to draw a state sample $x^{[m]}$. However, since our state transition is deterministic, no new states will be introduced in the forward sampling step (Line 4). The result is quite daunting: With probability one, M identical copies of a single state will survive; the diversity will disappear due to the repetitive resampling. To an outside observer, it may appear that the robot has uniquely determined the world state—an apparent contradiction to the fact that the robot possesses no sensors.

This example hints at an important limitation of particle filters with immense practical ramifications. In particular, the resampling process induces a loss of diversity in the particle population, which in fact manifests itself as approximation error. Such error is called *variance* of the estimator: Even though the variance of the particle set itself decreases, the variance of the particle set as an estimator of the true belief increases. Controlling this variance, or error, of the particle filter is essential for any practical implementation.

There exist two major strategies for variance reduction. First, one may reduce the frequency at which resampling takes place. When the state is known to be static $(x_t = x_{t-1})$ one should never resample. This is the case, for example, in mobile robot localization: When the robot stops, resampling should be suspended (and in fact it is usually a good idea to suspend the integration of measurements as well). Even if the state changes, it is often a good idea to reduce the frequency of resampling. Multiple measurements can always be integrated via multiplicatively

1:          **Algorithm Low_variance_sampler($\mathcal{X}_t, \mathcal{W}_t$):**
2:              $\bar{\mathcal{X}}_t = \emptyset$
3:              $r = \text{rand}(0; M^{-1})$
4:              $c = w_t^{[1]}$
5:              $i = 1$
6:              for $m = 1$ to M do
7:                  $u = r + (m-1) \cdot M^{-1}$
8:                  while $u > c$
9:                      $i = i + 1$
10:                     $c = c + w_t^{[i]}$
11:                 endwhile
12:                 add $x_t^{[i]}$ to $\bar{\mathcal{X}}_t$
13:             endfor
14:             return $\bar{\mathcal{X}}_t$

Table 4.4 Low variance resampling for the particle filter. This routine uses a single random number to sample from the particle set $\mathcal{X}$ with associated weights $\mathcal{W}$, yet the probability of a particle to be resampled is still proportional to its weight. Furthermore, the sampler is efficient: Sampling M particles requires $O(M)$ time.

updating the importance factor as noted above. More specifically, it maintains the importance weight in memory and updates them as follows:

$$
w_t^{[m]} \quad = \quad \left\{ \begin{array}{ll} 1 & \text{if resampling took place} \\ p(z_t \mid x_t^{[m]})\, w_{t-1}^{[m]} & \text{if no resampling took place} \end{array} \right. \tag{4.37}
$$

The choice of when to resample is intricate and requires practical experience: Resampling too often increases the risk of losing diversity. If one samples too infrequently, many samples might be wasted in regions of low probability. A standard approach to determining whether or not resampling should be performed is to measure the variance of the importance weights. The variance of the weights relates to the efficiency of the sample based representation. If all weights are identical, then the variance is zero and no resampling should be performed. If, on the other hand, the weights are concentrated on a small number of samples, then the weight variance is high and resampling should be performed.

The second strategy for reducing the sampling error is known as *low variance sampling*. Table 4.4 depicts an implementation of a low variance sampler. The basic idea is that instead of selecting samples independently of each other in the
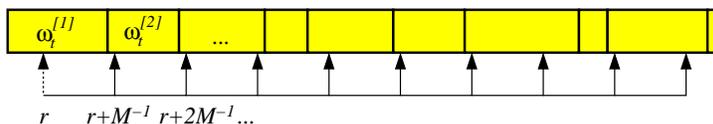
Figure 4.3 Principle of the low variance resampling procedure. We choose a random number r and then select those particles that correspond to $u = r + (m-1) \cdot M^{-1}$ where $m = 1, \ldots, M$.

resampling process (as is the case for the basic particle filter in Table 4.3), the selection involves a sequential stochastic process.

Instead of choosing M random numbers and selecting those particles that correspond to these random numbers, this algorithm computes a single random number and selects samples according to this number but still with a probability proportional to the sample weight. This is achieved by drawing a random number r in the interval $[0; M^{-1}[$, where M is the number of samples to be drawn at time t. The algorithm in Table 4.4 then selects particles by repeatedly adding the fixed amount M^{-1} to r and by choosing the particle that corresponds to the resulting number. Any number u in $[0; 1]$ points to exactly one particle, namely the particle i for which

$$ i = \operatorname*{argmin}_{j} \sum_{m=1}^{j} w_t^{[m]} \geq u \tag{4.38} $$

The while loop in Table 4.4 serves two tasks, it computes the sum in the right-hand side of this equation and additionally checks whether i is the index of the first particle such that the corresponding sum of weights exceeds u. The selection is then carried out in Line 12. This process is also illustrated in Figure 4.3.

The advantage of the low-variance sampler is threefold. First, it covers the space of samples in a more systematic fashion than the independent random sampler. This should be obvious from the fact that the dependent sampler cycles through all particles systematically, rather than choosing them independently at random. Second, if all the samples have the same importance factors, the resulting sample set $\bar{\mathcal{X}}_t$ is equivalent to $\mathcal{X}_t$ so that no samples are lost if we resample without having integrated an observation into $\mathcal{X}_t$. Third, the low-variance sampler has a complexity of $O(M)$. Achieving the same complexity for independent sampling is difficult; obvious implementations require a $O(\log M)$ search for each particle once a random number has been drawn, which results in a complexity of $O(M \log M)$ for the entire resampling process. Computation time is of essence when using particle filters, and often an efficient implementation of the resampling process can make a huge difference in the practical performance. For these

reasons, most implementations of particle filters in robotics tend to rely on mechanisms like the one just discussed.

In general, the literature on efficient sampling is huge. Another popular option is *stratified sampling*, in which particles are grouped into subsets. The number of samples in each subset can be kept the same over time, regardless of the total weight of the particles contained in each subset. Such techniques tend to perform well when a robot tracks multiple, distinct hypotheses with a single particle filter.

3. A third source of error pertains to the divergence of the proposal and target distribution. We already hinted at the problem above, when discussing importance sampling. In essence, particles are generated from a proposal distribution that does not consider the measurement (cf., Equation (4.28)). The target distribution, which is the familiar Bayes filter posterior, depends of course on the measurement. The efficiency of the particle filter relies crucially on the 'match' between the proposal and the target distribution. If, at one extreme, the sensors of the robot are highly inaccurate but its motion is very accurate, the target distribution will be similar to the proposal distribution and the particle filter will be efficient. If, on the other hand, the sensors are highly accurate but the motion is not, these distributions can deviate substantially and the resulting particle filter can become arbitrarily inefficient. An extreme example of this would be a robot with *deterministic* sensors. For most deterministic sensors, the support of the measurement probability $p(z \mid x)$ will be limited to a submanifold of the state space. For example, consider a mobile robot that performs localization with noise-free range sensors. Clearly, $p(z \mid x)$ will be zero for almost every state x, with the exceptions of those that match the range measurement z exactly. Such a situation can be fatal: the proposal distribution will practically never generate a sample x which *exactly* corresponds to the range measurement z. Thus, all importance weights will be zero with probability one, and the resampling step becomes ill-conditioned. More generally, if $p(z \mid x)$ is degenerate, meaning that its support is restricted to a manifold of a smaller dimension than the dimension of the state space, the plain particle filter algorithm is inapplicable.

There exist a range of techniques for overcoming this problem. One simple-minded technique is to simply assume more noise in perception than there actually is. For example, one might use a measurement model $p(z \mid x)$ that overestimates the actual noise in the range measurements. In many implementations, such a step improves the accuracy of the particle filter—despite the oddity of using a knowingly incorrect measurement probability. Other techniques involve modifications of the proposal distribution in ways that incorporate the measurement. Such techniques will be discussed in later chapters of this book.

4. A fourth and final disadvantage of the particle filter is known as the *particle deprivation problem*. When performing estimation in a high-dimensional space,

there may be no particles in the vicinity to the correct state. This might be because the number of particles is too small to cover all relevant regions with high likelihood. However, one might argue that this ultimately must happen in any particle filter, regardless of the particle set size M. Particle deprivation occurs as the result of random resampling; an unlucky series of random numbers can wipe out all particles near the true state. At each resampling step, the probability for this to happen is larger than zero (although it is usually exponentially small in M). Thus, we only have to run the particle filter long enough. Eventually, we will generate an estimate that is arbitrarily incorrect.

In practice, problems of this nature only tend to arise when M is small relative to the space of all states with high likelihood. A popular solution to this problem is to add a small number of randomly generated particles into the set after each resampling process, regardless of the actual sequence of motion and measurement commands. Such a methodology can reduce (but not fix) the particle deprivation problem, but at the expense of an incorrect posterior estimate. The advantage of adding random samples lies in its simplicity: The software modification necessary to add random samples in a particle filter is minimal. As a rule of thumb, adding random samples should be considered a measure of last resort, which should only be applied if all other techniques for fixing a deprivation problem have failed.

This discussion showed that the quality of the sample based representation increases with the number of samples. An important question is therefore how many samples should be used for a specific estimation problem. Unfortunately, there is no perfect answer to this question and it is often left to the user to determine the required number of samples. As a rule of thumb, the number of samples strongly depends on the dimensionality of the state space and the uncertainty of the distributions approximated by the particle filter. For example, uniform distributions require many more samples than distributions focused on a small region of the state space. A more detailed discussion on sample sizes will be given in the context of robot localization, when we consider adaptive particle filters (see Section **??**).

4.3   SUMMARY

This section introduced two nonparametric Bayes filters, histogram filters and particle filters. Nonparametric filters approximate the posterior by a finite number of values. Under mild assumptions on the system model and the shape of the posterior, both have the property that the approximation error converges uniformly to zero as the the number of values used to represent the posterior goes to infinity.

- The histogram filter decomposes the state space into finitely many convex regions. It represents the cumulative posterior probability of each region by a single numerical value.

- There exist many decomposition techniques in robotics. In particular, the granularity of a decomposition may or may not depend on the structure of the environment. When it does, the resulting algorithms are often called "topological."

- Decomposition techniques can be divided into static and dynamic. Static decompositions are made in advance, irrespective of the shape of the belief. Dynamic decompositions rely on specifics of the robot's belief when decomposing the state space, often attempting to increase spatial resolution in proportion to the posterior probability. Dynamic decompositions tend to give better results, but they are also more difficult to implement.

- An alternative nonparametric technique is known as particle filter. Particle filters represent posteriors by a random sample of states, drawn from the posterior. Such samples are called particles. Particle filter are extremely easy to implement, and they are the most versatile of all Bayes filter algorithms represented in this book.

- Specific strategies exist to reduce the error in particle filters. Among the most popular ones are techniques for reducing the variance of the estimate that arises from the randomness of the algorithm, and techniques for adapting the number of particles in accordance with the complexity of the posterior.

The filter algorithms discussed in this and the previous chapter lay the groundwork for most probabilistic robotics algorithms discussed throughout the remainder of this book. The material presented here represents many of today's most popular algorithms and representations in probabilistic robotics.

4.4   BIBLIOGRAPHICAL REMARKS

5

ROBOT MOTION

5.1 INTRODUCTION

This and the next chapter describe the two remaining components for implementing the filter algorithms described thus far: the motion and the measurement models. This chapter focuses on the motion model. It provides in-depth examples of probabilistic motion models as they are being used in actual robotics implementations. These models comprise the state transition probability $p(x_t \mid u_t, x_{t-1})$, which plays an essential role in the prediction step of the Bayes filter. The subsequent chapter will describe probabilistic models of sensor measurements $p(z_t \mid x_t)$, which are essential for the measurement update step. The material presented here will find its application in all chapters that follow.

Robot kinematics, which is the central topic of this chapter, has been studied thoroughly in past decades. However, is has almost exclusively been addressed in deterministic form. Probabilistic robotics generalizes kinematic equations to the fact that the outcome of a control is uncertain, due to control noise or unmodeled exogenous effects. Following the theme of this book, our description will be probabilistic: The outcome of a control will be described by a posterior probability. In doing so, the resulting models will be amenable to the probabilistic state estimation techniques described in the previous chapters.

Our exposition focuses entirely on mobile robot kinematics for robots operating in planar environments. In this way, it is much more specific than most contemporary treatments of kinematics. No model of manipulator kinematics will be provided, neither will we discuss models of robot dynamics. However, this restricted choice of material is by no means to be interpreted that probabilistic ideas are limited to kinematic models of mobile robots. Rather, it is descriptive of the present state of the art, as

probabilistic techniques have enjoyed their biggest successes in mobile robotics, using models of the types described in this chapter. The use of more sophisticated probabilistic models (e.g., probabilistic models of robot dynamics) remains largely unexplored in the literature. Such extensions, however, are not infeasible. As this chapter illustrates, deterministic robot actuator models are "probilified" by adding noise variables that characterize the types of uncertainty that exist in robotic actuation.

In theory, the goal of a proper probabilistic model may appear to accurately model the specific types of uncertainty that exist in robot actuation and perception. In practice, the exact shape of the model often seems to be less important than the fact that some provisions for uncertain outcomes are provided in the first place. In fact, many of the models that have proven most successful in practical applications vastly overestimate the amount of uncertainty. By doing so, the resulting algorithms are more robust to violations of the Markov assumptions (Chapter 2.4.4), such as unmodeled state and the effect of algorithmic approximations. We will point out such findings in later chapters, when discussing actual implementations of probabilistic robotic algorithms.

5.2   PRELIMINARIES

5.2.1   Kinematic Configuration

Kinematics is the calculus describing the effect of control actions on the configuration of a robot. The configuration of a rigid mobile robot is commonly described by six variables, its three-dimensional Cartesian coordinates and its three Euler angles (roll, pitch, yaw) relative to an external coordinate frame. The material presented in this book is largely restricted to mobile robots operating in planar environments, whose kinematic state, or *pose*, is summarized by three variables. This is illustrated in Figure 5.1. The robot's pose comprises its two-dimensional planar coordinates relative to an external coordinate frame, along with its angular orientation. Denoting the former as x and y (not to be confused with the state variable x_t), and the latter by θ, the pose of the robot is described by the following vector:

$$\begin{pmatrix} x \\ y \\ \theta \end{pmatrix} \tag{5.1}$$
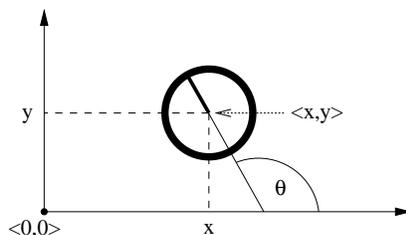
Figure 5.1 Robot pose, shown in a global coordinate system.

The orientation of a robot is often called *bearing*, or *heading direction*. As shown in Figure 5.1, we postulate that a robot with orientation $\theta = 0$ points into the direction of its x-axis. A robot with orientation $\theta = .5\pi$ points into the direction of its y-axis.

Pose without orientation will be called *location*. The concept of location will be important in the next chapter, when we discuss measures to perceive robot environments. For simplicity, locations in this book are usually described by two-dimensional vectors, which refer to the x-y coordinates of an object:

$$\begin{pmatrix} x \\ y \end{pmatrix} \tag{5.2}$$

Sometimes we will describe locations in the full 3D coordinate frame. Both the pose and the locations of objects in the environment may constitute the kinematic state x_t of the robot-environment system.

5.2.2 Probabilistic Kinematics

The probabilistic kinematic model, or *motion model* plays the role of the state transition model in mobile robotics. This model is the familiar conditional density

$$p(x_t \mid u_t, x_{t-1}) \tag{5.3}$$

Here x_t and x_{t-1} are both robot poses (and not just its x-coordinates), and u_t is a motion command. This model describes the posterior distribution over kinematics

(a)   (b)
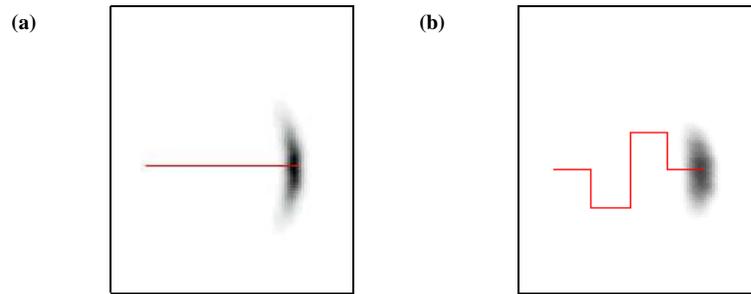
Figure 5.2 The motion model: Posterior distributions of the robot's pose upon executing the motion command illustrated by the solid line. The darker a location, the more likely it is. This plot has been projected into 2D. The original density is three-dimensional, taking the robot's heading direction θ into account.

states that a robots assumes when executing the motion command u_t when its pose is x_{t-1}. In implementations, u_t is sometimes provided by a robot's odometry. However, for conceptual reasons we will refer to u_t as control.

Figure 5.2 shows two examples that illustrate the kinematic model for a rigid mobile robot operating in a planar environment. In both cases, the robot's initial pose is x_{t-1}. The distribution $p(x_t \mid u_t, x_{t-1})$ is visualized by the grayly shaded area: The darker a pose, the more likely it is. In this figure, the posterior pose probability is projected into x-y-space, that is, the figure lacks a dimension corresponding to the robot's orientation. In Figure 5.2a, a robot moves forward some distance, during which it may accrue translational and rotational error as indicated. Figure 5.2b shows the resulting distribution of a more complicated motion command, which leads to a larger spread of uncertainty.

This chapter provides in detail two specific probabilistic motion models $p(x_t \mid u_t, x_{t-1})$, both for mobile robots operating in the plane. Both models are somewhat complimentary in the type of motion information that is being processed. The first model assumes that the motion data u_t specifies the velocity commands given to the robot's motors. Many commercial mobile robots (e.g., differential drive, synchro drive) are actuated by independent translational and rotational velocities, or are best thought of being actuated in this way. The second model assumes that one is provided with odometry information. Most commercial bases provide odometry using kinematic information (distance traveled, angle turned). The resulting probabilistic model for integrating such information is somewhat different from the velocity model.

In practice, odometry models tend to be more accurate than velocity models, for the simple reasons that most commercial robots do not execute velocity commands with the level of accuracy that can be obtained by measuring the revolution of the robot's wheels. However odometry is only available post-the-fact. Hence it cannot be used for motion planning. Planning algorithms such as collision avoidance have to predict the effects of motion. Thus, odometry models are usually applied for estimation, whereas velocity models are used for probabilistic motion planning.

5.3 VELOCITY MOTION MODEL

The velocity motion model assumes that we can control a robot through two velocities, a rotational and a translational velocity. Many commercial robots offer control interfaces where the programmer specifies velocities. Drive trains that are commonly controlled in this way include the differential drive, the Ackerman drive, the synchrodrive, and some holonomic drives (but not all). Drive systems not covered by our model are those without non-holonomic constraints, such as robots equipped with Mecanum wheels or legged robots.

We will denote the translational velocity at time t by v_t, and the rotational velocity by ω_t. Hence, we have

$$u_t = \begin{pmatrix} v_t \\ \omega_t \end{pmatrix} \tag{5.4}$$

We arbitrarily postulate that positive rotational velocities ω_t induce a counterclockwise rotation (left turns). Positive translational velocities v_t correspond to forward motion.

5.3.1 Closed Form Calculation

A possible algorithm for computing the probability $p(x_t \mid u_t, x_{t-1})$ is shown in Table 5.1. It accepts as input an initial pose $x_{t-1} = (x \ y \ \theta)^T$, a control $u_t = (v \ \omega)^T$, and a hypothesized successor pose $x_t = (x' \ y' \ \theta')^T$. It outputs the probability $p(x_t \mid u_t, x_{t-1})$ of being at x_t after executing control u_t beginning in state x_{t-1}, assuming that the control is carried out for the fixed duration Δt. The parameters α_1 to α_6 are robot-specific motion error parameters. This algorithm first calculates the controls of an error-free robot; the meaning of the individual variables in this calculation will become more apparent below, when we derive it. These parameters are given by $\hat{v}$ and $\hat{\omega}$.

(a)   (b)   (c)
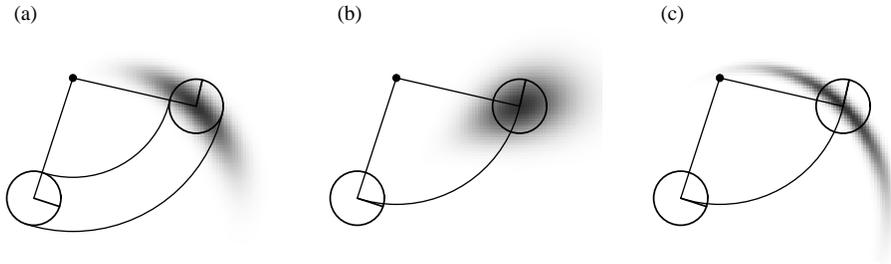
Figure 5.3 The velocity motion model, for different noise parameter settings.

The function **prob**(x, b) models the motion error. It computes the probability of its parameter x under a zero-centered random variable with variance b. Two possible implementations are shown in Table 5.2, for error variables with normal distribution and triangular distribution, respectively.

Figure 5.3 shows examples of this velocity motion model, projected into x-y-space. In all three cases, the robot sets the same translational and angular velocity. Figure 5.3a shows the resulting distribution with moderate error parameters α_1 to α_6. The distribution shown in Figure 5.3b is obtained with smaller angular error (parameters α_3 and α_4) but larger translational error (parameters α_1 and α_2). Figure 5.3c shows the distribution under large angular and small translational error.

5.3.2   Sampling Algorithm

For particle filters (cf. Section 4.2.1), it suffices to sample from the motion model $p(x_t \mid u_t, x_{t-1})$, instead of computing the posterior for arbitrary x_t, u_t and x_{t-1}. Sampling from a conditional density is different than calculating the density: In sampling, one is given u_t and x_{t-1} and seeks to generate a random x_t drawn according to the motion model $p(x_t \mid u_t, x_{t-1})$. When calculating the density, one is also given x_t generated through other means, and one seeks to compute the probability of x_t under $p(x_t \mid u_t, x_{t-1})$.

The algorithm **sample_motion_model_velocity** in Table 5.3 generates random samples from $p(x_t \mid u_t, x_{t-1})$ for a fixed control u_t and pose x_{t-1}. It accepts x_{t-1} and u_t as input and generates a random pose x_t according to the distribution $p(x_t \mid u_t, x_{t-1})$. Line 2 through 4 "perturb" the commanded control parameters by noise, drawn from the error parameters of the kinematic motion model. The noise values are then used to generate the sample's new pose, in Lines 5 through 7. Thus, the sampling pro-

1:         **Algorithm motion_model_velocity(x_t, u_t, x_{t-1}):**

2:         $\mu = \dfrac{1}{2} \dfrac{(x - x')\cos\theta + (y - y')\sin\theta}{(y - y')\cos\theta - (x - x')\sin\theta}$

3:         $x^* = \dfrac{x + x'}{2} + \mu(y - y')$

4:         $y^* = \dfrac{y + y'}{2} + \mu(x' - x)$

5:         $r^* = \sqrt{(x - x^*)^2 + (y - y^*)^2}$

6:         $\Delta\theta = \operatorname{atan2}(y' - y^*, x' - x^*) - \operatorname{atan2}(y - y^*, x - x^*)$

7:         $\hat{v} = \dfrac{\Delta\theta}{\Delta t}\, r^*$

8:         $\hat{\omega} = \dfrac{\Delta\theta}{\Delta t}$

9:         $\hat{\gamma} = \frac{\theta' - \theta}{\Delta t} - \hat{\omega}$

10:        *return* $\mathbf{prob}(v - \hat{v}, \alpha_1|v| + \alpha_2|\omega|) \;\cdot\; \mathbf{prob}(\omega - \hat{\omega}, \alpha_3|v| + \alpha_4|\omega|)$
                       $\cdot\; \mathbf{prob}(\hat{\gamma}, \alpha_5|v| + \alpha_6|\omega|)$

Table 5.1 Algorithm for computing $p(x_t \mid u_t, x_{t-1})$ based on velocity information. Here we assume x_{t-1} is represented by the vector $(x \;\; y \;\; \theta)^T$; x_t is represented by $(x' \;\; y' \;\; \theta')^T$; and u_t is represented by the velocity vector $(v \;\; \omega)^T$. The function $\mathbf{prob}(a, b)$ computes the probability of its argument a under a zero-centered distribution with variance b. It may be implemented using any of the algorithms in Table 5.2.

1:         **Algorithm prob_normal_distribution(a, b):**

2:         *return* $\frac{1}{\sqrt{2\pi b}}\, e^{-\frac{1}{2}\frac{a^2}{b}}$

3:         **Algorithm prob_triangular_distribution(a, b):**

4:         *if* $|a| > \sqrt{6b}$

5:             *return 0*

6:         *else*

7:             *return* $\frac{\sqrt{6b} - |a|}{6b}$

Table 5.2 Algorithms for computing densities of a zero-centered normal distribution and the triangular distribution with variance b.

1:       **Algorithm sample_motion_model_velocity(u_t, x_{t-1}):**

2:       $\hat{v} = v + \textbf{sample}(\alpha_1|v| + \alpha_2|\omega|)$

3:       $\hat{\omega} = \omega + \textbf{sample}(\alpha_3|v| + \alpha_4|\omega|)$

4:       $\hat{\gamma} = \textbf{sample}(\alpha_5|v| + \alpha_6|\omega|)$

5:       $x' = x - \frac{\hat{v}}{\hat{\omega}}\sin\theta + \frac{\hat{v}}{\hat{\omega}}\,sin(\theta + \hat{\omega}\Delta t)$

6:       $y' = y + \frac{\hat{v}}{\hat{\omega}}\cos\theta - \frac{\hat{v}}{\hat{\omega}}\,cos(\theta + \hat{\omega}\Delta t)$

7:       $\theta' = \theta + \hat{\omega}\Delta t + \hat{\gamma}\Delta t$

8:       *return* $x_t = (x', y', \theta')^T$

Table 5.3 Algorithm for sampling poses $x_t = (x'\ \ y'\ \ \theta')^T$ from a pose $x_{t-1} = (x\ \ y\ \ \theta)^T$ and a control $u_t = (v\ \ \omega)^T$. Note that we are perturbing the final orientation by an additional random term, $\hat{\gamma}$. The variables α_1 through α_6 are the parameters of the motion noise. The function **sample**(b) generates a random sample from a zero-centered distribution with variance b. It may, for example, be implemented using the algorithms in Table 5.4.

1:       **Algorithm sample_normal_distribution(b):**

2:       *return* $\dfrac{b}{6}\displaystyle\sum_{i=1}^{12}\textbf{rand}(-1,1)$

3:       **Algorithm sample_triangular_distribution(b):**

4:       *return* $b \cdot \textbf{rand}(-1,1) \cdot \textbf{rand}(-1,1)$

Table 5.4 Algorithm for sampling from (approximate) normal and triangular distributions with zero mean and variance b. The function **rand**(x, y) is assumed to be a pseudo random number generator with uniform distribution in $[x, y]$.
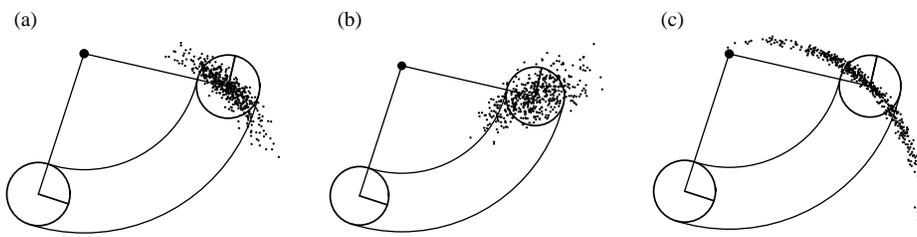
Figure 5.4 Sampling from the velocity motion model, using the same parameters as in Figure 5.3. Each diagram shows 500 samples.

cedure implements a simple physical robot motion model that incorporates control noise in its prediction, in just about the most straightforward way. Figure 5.4 illustrates the outcome of this sampling routine. It depicts 500 samples generated by **sample_motion_model_velocity**. The reader might want to compare this figure with the density depicted in in Figure 5.3.

We note that in many cases, it is easier to sample x_t than calculate the density of a given x_t. This is because samples require only a forward simulation of the physical motion model. To compute the probability of a hypothetical pose amounts to retroguessing of the error parameters, which requires to calculate the inverse of the physical motion model. The fact that particle filters rely on sampling makes them specifically attractive from an implementation point of view.

5.3.3   Mathematical Derivation

We will now derive the algorithms **motion_model_velocity** and **sample_motion_model_velocity**. As usual, the reader not interested in the mathematical details is invited to skip this section at first reading, and continue in Section 5.4 (page 107). The derivation begins with a generative model of robot motion, and then derives formulae for sampling and computing $p(x_t \mid u_t, x_{t-1})$ for arbitrary x_t, u_t, and x_{t-1}.

Exact Motion

Before turning to the probabilistic case, let us begin by stating the kinematics for an ideal, noise-free robot. Let $u_t = (v \ \omega)^T$ denote the control at time t. If both velocities are kept at a fixed value for the entire time interval $(t-1, t]$, the robot moves on a circle
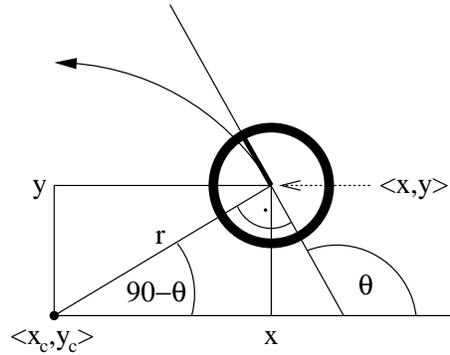
Figure 5.5 Motion carried out by a noise-free robot moving with constant velocities v and ω and starting at $(x \ y \ \theta)^T$.

with radius

$$r = \left| \frac{v}{\omega} \right| \tag{5.5}$$

This follows from the general relationship between the translational and rotational velocities v and ω for an arbitrary object moving on a circular trajectory with radius r:

$$v = \omega \cdot r. \tag{5.6}$$

Equation (5.5) encompasses the case where the robot does not turn at all (i.e., $\omega = 0$), in which case the robot moves on a straight line. A straight line corresponds to a circle with infinite radius, hence we note that r may be infinite.

Let $x_{t-1} = (x, y, \theta)^T$ be the initial pose of the robot, and suppose we keep the velocity constant at $(v \ \omega)^T$ for some time Δt. As one easily shows, the center of the circle is at

$$x_c = x - \frac{v}{\omega} \sin \theta \tag{5.7}$$

$$y_c = y + \frac{v}{\omega} \cos \theta \tag{5.8}$$

The variables $(x_c \ \ y_c)^T$ denote this coordinate. After Δt time of motion, our ideal robot will be at $x_t = (x', y', \theta')^T$ with

$$
\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} = \begin{pmatrix} x_c + \frac{v}{\omega} \ \sin(\theta + \omega \Delta t) \\ y_c - \frac{v}{\omega} \ \cos(\theta + \omega \Delta t) \\ \theta + \omega \Delta t \end{pmatrix}
$$
$$
= \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} -\frac{v}{\omega} \sin \theta + \frac{v}{\omega} \ \sin(\theta + \omega \Delta t) \\ \frac{v}{\omega} \cos \theta - \frac{v}{\omega} \cos(\theta + \omega \Delta t) \\ \omega \Delta t \end{pmatrix} \tag{5.9}
$$

The derivation of this expression follows from simple trigonometry: After Δt units of time, the noise-free robot has progressed $v \cdot \Delta t$ along the circle, which caused it's heading direction to turn by $\omega \cdot \Delta t$. At the same time, its x and y coordinate is given by the intersection of the circle about $(x_c \ \ y_c)^T$, and the ray starting at $(x_c \ \ y_c)^T$ at the angle perpendicular to $\omega \cdot \Delta t$. The second transformation simply substitutes (5.8) into the resulting motion equations.

Of course, real robots cannot jump from one velocity to another, and keep velocity constant in each time interval. To compute the kinematics with non-constant velocities, it is therefore common practice to use small values for Δt, and to approximate the actual velocity by a constant within each time interval. The (approximate) final pose is then obtained by concatenating the corresponding cyclic trajectories using the mathematical equations just stated.

Real Motion

In reality, robot motion is subject to noise. The actual velocities differ from the commanded ones (or measured ones, if the robot possesses a sensor for measuring velocity). We will model this difference by a zero-centered random variable with finite variance. More precisely, let us assume the actual velocities are given by

$$
\begin{pmatrix} \hat{v} \\ \hat{\omega} \end{pmatrix} = \begin{pmatrix} v \\ \omega \end{pmatrix} + \begin{pmatrix} \varepsilon_{\alpha_1 |v| + \alpha_2 |\omega|} \\ \varepsilon_{\alpha_3 |v| + \alpha_4 |\omega|} \end{pmatrix} \tag{5.10}
$$

Here ε_b is a zero-mean error variable with variance b. Thus, the true velocity equals the commanded velocity plus some small, additive error (noise). In our model, the variance of the error is proportional to the commanded velocity. The parameters α_1 to α_4 (with $\alpha_i \geq 0$ for $i = 1, \ldots, 4$) are robot-specific error parameters. They model the accuracy of the robot. The less accurate a robot, the larger these parameters.
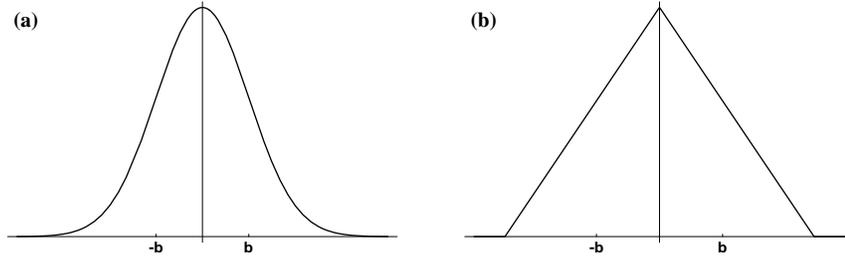
Figure 5.6 Probability density functions with variance b: (a) Normal distribution, (b) triangular distribution.

Two common choices for the error ε_b are:

- **Normal distribution.** The normal distribution with zero mean and variance b is given by the density function

$$\varepsilon_b(a) \quad = \quad \frac{1}{\sqrt{2\pi \cdot b}} \; e^{-\frac{1}{2}\frac{a^2}{b}} \tag{5.11}$$

  Figure 5.6a shows the density function of a normal distribution with variance b. Normal distributions are commonly used to model noise in continuous stochastic processes, despite the fact that their support, that is the set of points a with $p(a) > 0$, is $\Re$.

- **Triangular distribution.** The density of triangular distribution with zero mean and variance b is given by

$$\varepsilon_b(a) = \left\{ \begin{array}{ll} 0 & \text{if } |a| > \sqrt{6b} \\ \frac{\sqrt{6b}-|a|}{6b} & \text{otherwise} \end{array} \right. \tag{5.12}$$

  which is non-zero only in $(-\sqrt{6b}; \sqrt{6b})$. As Figure 5.6b suggests, the density resembles the shape of a symmetric triangle—hence the name.

A better model of the actual pose $x_t = (x' \;\; y' \;\; \theta')^T$ after executing the motion command $u_t = (v \;\; \omega)^T$ at $x_{t-1} = (x \;\; y \;\; \theta)^T$ is thus

$$\left(\begin{array}{c} x' \\ y' \\ \theta' \end{array} \right) \;\; = \;\; \left(\begin{array}{c} x \\ y \\ \theta \end{array} \right) + \left(\begin{array}{c} -\frac{\hat{v}}{\hat{\omega}}\sin\theta + \frac{\hat{v}}{\hat{\omega}}\;\sin(\theta + \hat{\omega}\Delta t) \\ \frac{\hat{v}}{\hat{\omega}}\cos\theta - \frac{\hat{v}}{\hat{\omega}}\;\cos(\theta + \hat{\omega}\Delta t) \\ \hat{\omega}\Delta t \end{array} \right) \tag{5.13}$$

This equation is simply obtained by substituting the commanded velocity $u_t = (v \ \omega)^T$ with the noisy motion $(\hat{v} \ \hat{\omega})$ in (5.9). However, this model is still not very realistic, for reasons discussed in turn.

Final Orientation

The two equations given above exactly describe the final location of the robot given that the robot actually moves on an exact circular trajectory with radius $r = \frac{\hat{v}}{\hat{\omega}}$. While the radius of this circular segment and the distance traveled is influenced by the control noise, the very fact that the trajectory is circular is not. The assumption of circular motion leads to an important degeneracy. In particular, the support of the density $p(x_t \mid u_t, x_{t-1})$ is two-dimensional, within a three-dimensional embedding pose space. The fact that all posterior poses are located on a two-dimensional manifold within the three-dimensional pose space is a direct consequence of the fact that we used only two noise variables, one for v and one for ω. Unfortunately, this degeneracy has important ramifications when applying Bayes filters for state estimation.

In reality, any meaningful posterior distribution is of course not degenerate, and poses can be found within a three-dimensional space of variations in x, y, and θ. To generalize our motion model accordingly, we will assume that the robot performs a rotation $\hat{\gamma}$ when it arrives at its final pose. Thus, instead of computing θ' according to (5.13), we model the final orientation by

$$\theta' \ = \ \theta + \hat{\omega}\Delta t + \hat{\gamma}\Delta t \tag{5.14}$$

with

$$\hat{\gamma} \ = \ \varepsilon_{\alpha_5|v|+\alpha_6|\omega|} \tag{5.15}$$

Here α_5 and α_6 are additional robot-specific parameters that determine the variance of the additional rotational noise. Thus, the resulting motion model is as follows:

$$\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} \ = \ \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} -\frac{\hat{v}}{\hat{\omega}}\sin\theta + \frac{\hat{v}}{\hat{\omega}}\sin(\theta + \hat{\omega}\Delta t) \\ \frac{\hat{v}}{\hat{\omega}}\cos\theta - \frac{\hat{v}}{\hat{\omega}}\cos(\theta + \hat{\omega}\Delta t) \\ \hat{\omega}\Delta t + \hat{\gamma}\Delta t \end{pmatrix} \tag{5.16}$$

Computation of $p(x_t \mid u_t, x_{t-1})$

The algorithm **motion_model_velocity** in Table 5.1 implements the computation of $p(x_t \mid u_t, x_{t-1})$ for given values of $x_{t-1} = (x \ \ y \ \ \theta)^T$, $u_t = (v \ \ \omega)^T$, and $x_t = (x' \ \ y' \ \ \theta')^T$. The derivation of this algorithm is somewhat involved, as it effectively implements an inverse motion model. In particular, **motion_model_velocity** determines motion parameters $\hat{u}_t = (\hat{v} \ \ \hat{\omega})^T$ from the poses x_{t-1} and x_t, along with an appropriate final rotation $\hat{\gamma}$. Our derivation makes it obvious as to why a final rotation is needed: For most values of x_{t-1}, u_t, and x_t, the motion probability would simply be zero without allowing for a final rotation.

Let us calculate the probability $p(x_t \mid u_t, x_{t-1})$ of control action $u_t = (v \ \ \omega)^T$ carrying the robot from the pose $x_{t-1} = (x \ \ y \ \ \theta)^T$ to the pose $x_t = (x' \ \ y' \ \ \theta')^T$ within Δt time units. To do so, we will first determine the control $\hat{u} = (\hat{v} \ \ \hat{\omega})^T$ required to carry the robot from x_{t-1} to position $(x' \ \ y')$, regardless of the robot's final orientation. Subsequently, we will determine the final rotation $\hat{\gamma}$ necessary for the robot to attain the orientation θ'. Based on these calculations, we can then easily calculate the desired probability $p(x_t \mid u_t, x_{t-1})$.

The reader may recall that our model assumes that the robot assumes a fixed velocity during Δt, resulting in a circular trajectory. For a robot that moved from $x_{t-1} = (x \ \ y \ \ \theta)^T$ to $x_t = (x' \ \ y')^T$, the center of the circle is defined as $(x^* \ \ y^*)^T$ and given by

$$\begin{pmatrix} x^* \\ y^* \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} -\lambda \sin\theta \\ \lambda \cos\theta \end{pmatrix} = \begin{pmatrix} \frac{x+x'}{2} + \mu(y - y') \\ \frac{y+y'}{2} + \mu(x' - x) \end{pmatrix} \quad (5.17)$$

for some unknown $\lambda, \mu \in \Re$. The first equality is the result of the fact that the circle's center is orthogonal to the initial heading direction of the robot; the second is a straightforward constraint that the center of the circle lies on a ray that lies on the half-way point between $(x \ \ y)^T$ and $(x' \ \ y')^T$ and is orthogonal to the line between these coordinates.

Usually, Equation (5.17) has a unique solution—except in the degenerate case of $\omega = 0$, in which the center of the circle lies at infinity. As the reader might want to verify, the solution is given by

$$\mu = \frac{1}{2} \frac{(x - x')\cos\theta + (y - y')\sin\theta}{(y - y')\cos\theta - (x - x')\sin\theta} \quad (5.18)$$

and hence

$$
\begin{pmatrix} x^* \\ y^* \end{pmatrix} = \begin{pmatrix} \frac{x+x'}{2} + \frac{1}{2} \frac{(x-x')\cos\theta + (y-y')\sin\theta}{(y-y')\cos\theta - (x-x')\sin\theta} (y-y') \\ \frac{y+y'}{2} + \frac{1}{2} \frac{(x-x')\cos\theta + (y-y')\sin\theta}{(y-y')\cos\theta - (x-x')\sin\theta} (x'-x) \end{pmatrix} \tag{5.19}
$$

The radius of the circle is now given by the Euclidean distance

$$
r^* = \sqrt{(x-x^*)^2 + (y-y^*)^2} = \sqrt{(x'-x^*)^2 + (y'-y^*)^2} \tag{5.20}
$$

Furthermore, we can now calculate the change of heading direction

$$
\Delta\theta = \operatorname{atan2}(y'-y^*, x'-x^*) - \operatorname{atan2}(y-y^*, x-x^*) \tag{5.21}
$$

Here $\operatorname{atan2}$ is the common extension of the arcus tangens of y/x extended to the $\Re^2$ (most programming languages provide an implementation of this function):

$$
\operatorname{atan2}(y,x) = \begin{cases} \operatorname{atan}(y/x) & \text{if } x > 0 \\ \operatorname{sign}(y)\,(\pi - \operatorname{atan}(|y/x|)) & \text{if } x < 0 \\ 0 & \text{if } x = y = 0 \\ \operatorname{sign}(y)\,\pi/2 & \text{if } x = 0, y \neq 0 \end{cases} \tag{5.22}
$$

Since we assume that the robot follows a circular trajectory, the translational distance between x_t and x_{t-1} (along this circle) is

$$
\Delta\text{dist} = r^* \cdot \Delta\theta \tag{5.23}
$$

From Δdist and $\Delta\theta$, it is now easy to compute the velocities $\hat{v}$ and $\hat{\omega}$:

$$
\hat{u}_t = \begin{pmatrix} \hat{v} \\ \hat{\omega} \end{pmatrix} = \Delta t^{-1} \begin{pmatrix} \Delta\text{dist} \\ \Delta\theta \end{pmatrix} \tag{5.24}
$$

The angle of the final rotation $\hat{\gamma}$ can be determined according to (5.14) as:

$$
\hat{\gamma} = \Delta t^{-1}(\theta' - \theta) - \hat{\omega} \tag{5.25}
$$

The *motion error* is the deviation of $\hat{u}_t$ and $\hat{\gamma}$ from the commanded velocity $u_t = (u \ \omega)^T$ and $\gamma = 0$, as defined in Equations (5.24) and (5.25).

$$
\begin{align}
v_{\mathrm{err}} &= v - \hat{v} \tag{5.26} \\
\omega_{\mathrm{err}} &= \omega - \hat{\omega} \tag{5.27} \\
\gamma_{\mathrm{err}} &= \hat{\gamma} \tag{5.28}
\end{align}
$$

Under our error model, specified in Equations (5.10), and (5.15), these errors have the following probabilities:

$$
\begin{align}
\varepsilon_{\alpha_1|v|+\alpha_2|\omega|}(v_{\mathrm{err}}) \tag{5.29} \\
\varepsilon_{\alpha_3|v|+\alpha_4|\omega|}(\omega_{\mathrm{err}}) \tag{5.30} \\
\varepsilon_{\alpha_5|v|+\alpha_6|\omega|}(\gamma_{\mathrm{err}}) \tag{5.31}
\end{align}
$$

where ε_b denotes a zero-mean error variable with variance b, as before. Since we assume independence between the different sources of error, the desired probability $p(x_t \mid u_t, x_{t-1})$ is the product of these individual errors:

$$
p(x_t \mid u_t, x_{t-1}) = \varepsilon_{\alpha_1|v|+\alpha_2|\omega|}(v_{\mathrm{err}}) \cdot \varepsilon_{\alpha_3|v|+\alpha_4|\omega|}(\omega_{\mathrm{err}}) \cdot \varepsilon_{\alpha_5|v|+\alpha_6|\omega|}(\gamma_{\mathrm{err}}) \tag{5.32}
$$

To see the correctness of the algorithm **motion_model_velocity** in Table 5.1, the reader may notice that this algorithm implements this expression. More specifically, lines 2 to 9 are equivalent to Equations (5.18), (5.19), (5.20), (5.21), (5.24), and (5.25). Line 10 implements (5.32), substituting the error terms as specified in Equations (5.29) to (5.31).

Sampling from $p(s' \mid a, s)$

The sampling algorithm **sample_motion_model_velocity** in Table 5.3 implements a forward model, as discussed earlier in this section. Lines 5 through 7 correspond to Equation (5.16). The noisy values calculated in lines 2 through 4 correspond to Equations (5.10) and (5.15).

The algorithm **sample_normal_distribution** in Table 5.4 implements a common approximation to sampling from a normal distribution. This approximation exploits the central limit theorem, which states that any average of non-degenerate random variables converges to a normal distribution. By averaging 12 uniform distributions, **sample_normal_distribution** generates values that are approximately normal
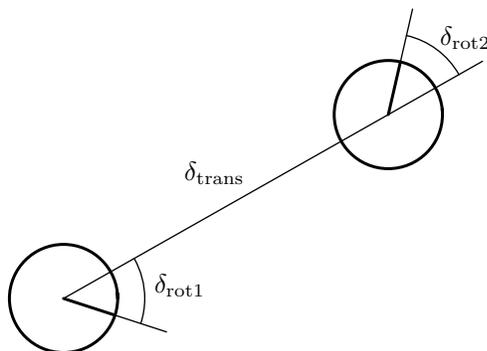
Figure 5.7 Odometry model: The robot motion in the time interval $(t-1, t]$ is approximated by a rotation δ_{rot1}, followed by a translation δ_{trans} and a second rotation δ_{rot2}. The turns and translation are noisy.

distributed; though technically the resulting values lie always in $[-2b, 2b]$. Finally, **sample_triangular_distribution** in Table 5.4 implements a sampler for triangular distributions.

5.4 ODOMETRY MOTION MODEL

The velocity motion model discussed thus far uses the robot's velocity to compute posteriors over poses. Alternatively, one might want to use the odometry measurements as the basis for calculating the robot's motion over time. Odometry is commonly obtained by integrating wheel encoders information; most commercial robots make such integrated pose estimation available in periodic time intervals (e.g., every tenth of a second). Practical experience suggests that odometry, while still erroneous, is usually more accurate than velocity. Both suffer from drift and slippage, but velocity additionally suffers from the mismatch between the actual motion controllers and its (crude) mathematical model. However, odometry is only available in retrospect, after the robot moved. This poses no problem for filter algorithms, but makes this information unusable for accurate motion planning and control.

1:          **Algorithm motion_model_odometry(x_t, u_t, x_{t-1}):**

2:                 $\delta_{\text{rot1}} = \text{atan2}(\bar{y}' - \bar{y}, \bar{x}' - \bar{x}) - \bar{\theta}$
3:                 $\delta_{\text{trans}} = \sqrt{(\bar{x} - \bar{x}')^2 + (\bar{y} - \bar{y}')^s}$
4:                 $\delta_{\text{rot2}} = \bar{\theta}' - \bar{\theta} - \delta_{\text{rot1}}$

5:                 $\hat{\delta}_{\text{rot1}} = \text{atan2}(y' - y, x' - x) - \theta$
6:                 $\hat{\delta}_{\text{trans}} = \sqrt{(x - x')^2 + (y - y')^2}$
7:                 $\hat{\delta}_{\text{rot2}} = \theta' - \theta - \hat{\delta}_{\text{rot1}}$

8:                 $p_1 = \textbf{prob}(\delta_{\text{rot1}} - \hat{\delta}_{\text{rot1}}, \alpha_1 \hat{\delta}_{\text{rot1}} + \alpha_2 \hat{\delta}_{\text{trans}})$
9:                 $p_2 = \textbf{prob}(\delta_{\text{trans}} - \hat{\delta}_{\text{trans}}, \alpha_3 \hat{\delta}_{\text{trans}} + \alpha_4(\hat{\delta}_{\text{rot1}} + \hat{\delta}_{\text{rot2}}))$
10:                $p_3 = \textbf{prob}(\delta_{\text{rot2}} - \hat{\delta}_{\text{rot2}}, \alpha_1 \hat{\delta}_{\text{rot2}} + \alpha_2 \hat{\delta}_{\text{trans}})$

11:                *return* $p_1 \cdot p_2 \cdot p_3$

Table 5.5   Algorithm for computing $p(x_t \mid u_t, x_{t-1})$ based on odometry information. Here the control u_t is

5.4.1   Closed Form Calculation

This section defines an alternative motion model that uses odometry measurements in lieu of controls. Technically, odometry are sensor measurements, not controls. To model odometry as measurements, the resulting Bayes filter would have to include the actual velocity as state variables—which increases the dimension of the state space. To keep the state space small, it is therefore common to simply consider the odometry as if it was a control signal. In this section, we will do exactly this, and treat odometry measurements as controls. The resulting model is at the core of many of today's best probabilistic robot systems.

Let us define the format of our control information. At time t, the correct pose of the robot is modeled by the random variable x_t. The robot odometry estimates this pose; however, due to drift and slippage there is no fixed coordinate transformation between the coordinates used by the robot's internal odometry and the physical world coordinates. In fact, knowing this transformation would solve the robot localization problem!

(a)   (b)   (c)
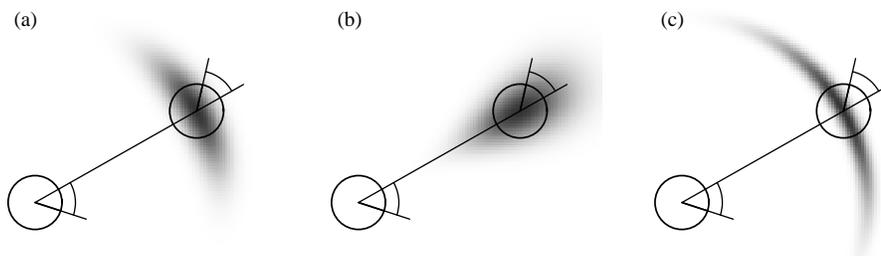
Figure 5.8   The odometry motion model, for different noise parameter settings.

The odometry model uses the *relative* information of the robot's internal odometry. More specifically, In the time interval $(t-1, t]$, the robot advances from a pose x_{t-1} to pose x_t. The odometry reports back to us a related advance from $\bar{x}_{t-1} = (\bar{x} \ \ \bar{y} \ \ \bar{\theta})$ to $\bar{x}_t = (\bar{x}' \ \ \bar{y}' \ \ \bar{\theta}')$. Here the bar indicates that these are odometry measurements, embedded in a robot-internal coordinate whose relation to the global world coordinates is unknown. The key insight for utilizing this information in state estimation is that the relative difference between $\bar{x}_{t-1}$ and $\bar{x}_t$, under an appropriate definition of the term "difference," is a good estimator for the difference of the true poses x_{t-1} and x_t. The motion information u_t is, thus, given by the pair

$$u_t \quad = \quad \begin{pmatrix} \bar{x}_{t-1} \\ \bar{x}_t \end{pmatrix} \tag{5.33}$$

To extract relative odometry, u_t is transformed into a sequence of three steps: a rotation, followed by a straight line motion (translation) and another rotation. Figure 5.7 illustrates this decomposition: the initial turn is called δ_{rot1}, the translation δ_{trans}, and the second rotation δ_{rot2}. As the reader easily verifies, each pair of positions $(\bar{s} \ \ \bar{s}')$ has a unique parameter vector $(\delta_{\mathrm{rot1}} \ \ \delta_{\mathrm{trans}} \ \ \delta_{\mathrm{rot2}})^T$, and these parameters are sufficient to reconstruct the relative motion between $\bar{s}$ and $\bar{s}'$. Thus, $\delta_{\mathrm{rot1}}, \delta_{\mathrm{trans}}, \delta_{\mathrm{rot2}}$ is a sufficient statistics of the relative motion encoded by the odometry. Our motion model assumes that these three parameters are corrupted by independent noise. The reader may note that odometry motion uses one more parameter than the velocity vector defined in the previous section, for which reason we will not face the same degeneracy that led to the definition of a "final rotation."

Before delving into mathematical detail, let us state the basic algorithm for calculating this density in closed form. Table 5.5 depicts the algorithm for computing $p(x_t \mid u_t, x_{t-1})$ from odometry. This algorithm accepts as an input an initial pose x_{t-1}, a

1:          **Algorithm sample_motion_model_odometry(u_t, x_{t-1}):**

2:              $\delta_{\text{rot1}} = \text{atan2}(\bar{y}' - \bar{y}, \bar{x}' - \bar{x}) - \bar{\theta}$

3:              $\delta_{\text{trans}} = \sqrt{(\bar{x} - \bar{x}')^2 + (\bar{y} - \bar{y}')^s}$

4:              $\delta_{\text{rot2}} = \bar{\theta}' - \bar{\theta} - \delta_{\text{rot1}}$

5:              $\hat{\delta}_{\text{rot1}} = \delta_{\text{rot1}} - \mathbf{sample}(\alpha_1 \delta_{\text{rot1}} + \alpha_2 \delta_{\text{trans}})$

6:              $\hat{\delta}_{\text{trans}} = \delta_{\text{trans}} - \mathbf{sample}(\alpha_3 \, \delta_{\text{trans}} + \alpha_4(\delta_{\text{rot1}} + \delta_{\text{rot2}}))$

7:              $\hat{\delta}_{\text{rot2}} = \delta_{\text{rot2}} - \mathbf{sample}(\alpha_1 \delta_{\text{rot2}} + \alpha_2 \delta_{\text{trans}})$

8:              $x' = x + \hat{\delta}_{\text{trans}} \, \cos(\theta + \hat{\delta}_{\text{rot1}})$

9:              $y' = y + \hat{\delta}_{\text{trans}} \, \sin(\theta + \hat{\delta}_{\text{rot1}})$

10:             $\theta' = \theta + \hat{\delta}_{\text{rot1}} + \hat{\delta}_{\text{rot2}}$

11:             *return* $x_t = (x', y', \theta')^T$

Table 5.6 Algorithm for sampling from $p(x_t \mid u_t, x_{t-1})$ based on odometry information. Here the pose at time t is represented by $x_{t-1} = (x \ y \ \theta)^T$. The control is a differentiable set of two pose estimates obtained by the robot's odometer, $u_t = (\bar{x}_{t-1} \ \bar{x}_t)^T$, with $\bar{x}_{t-1} = (\bar{x} \ \bar{y} \ \bar{\theta})$ and $\bar{x}_t = (\bar{x}' \ \bar{y}' \ \bar{\theta}')$.

pair of poses $u_t = (\bar{x}_{t-1} \ \bar{x}_t)^T$ obtained from the robot's odometry, and a hypothesized final pose x_t. It outputs the numerical probability $p(x_t \mid u_t, x_{t-1})$.

Let us dissect this algorithm. Lines 2 to 4 recover relative motion parameters $(\delta_{\text{rot1}} \quad \delta_{\text{trans}} \quad \delta_{\text{rot2}})^T$ from the odometry readings. As before, they implement an *inverse motion model*. The corresponding relative motion parameters $(\hat{\delta}_{\text{rot1}} \ \hat{\delta}_{\text{trans}} \ \hat{\delta}_{\text{rot2}})^T$ for the given poses x_{t-1} and x_t are calculated in Lines 5 through 7 of this algorithm. Lines 8 to 10 compute the error probabilities for the individual motion parameters. As above, the function $\mathbf{prob}(a, b)$ implements an error distribution over a with zero mean and variance b. Here the implementer must observe that all angular differences must lie in $[-\pi, \pi]$. Hence the outcome of $\delta_{\text{rot2}} - \bar{\delta}_{\text{rot2}}$ has to be truncated correspondingly—a common error that tends to yield occasional divergence of software based on this model. Finally, Line 11 returns the combined error probability, obtained by multiplying the individual error probabilities p_1, p_2, and p_3. This
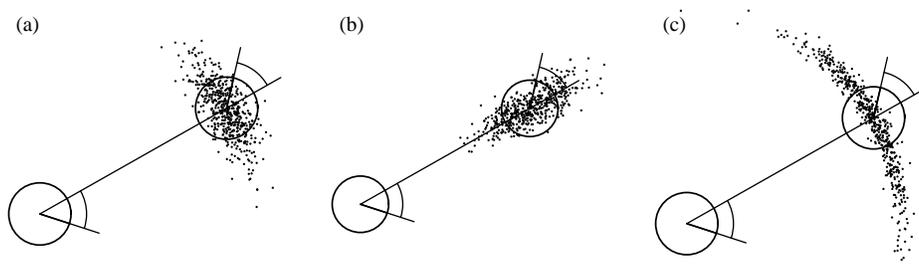
Figure 5.9   Sampling from the odometry motion model, using the same parameters as in Figure 5.8. Each diagram shows 500 samples.

last step assumes independence between the different error sources. The variables α_1 through α_4 are robot-specific parameters that specify the noise in robot motion.

Figure 5.8 shows examples of our odometry motion model for different values of the error parameters α_1 to α_4. The distribution in Figure 5.8a is a "proto-typical" one, whereas the ones shown in Figures 5.8b and 5.8c indicate unusually large translational and rotational errors, respectively. The reader may want to carefully compare these diagrams with those in Figure 5.3 on page 96. The smaller the time between to consecutive measurements, the more similar those different motion models. Thus, if the belief is updated frequently e.g., every tenth of a second for a conventional indoor robot, the difference between these motion models is not very significant. In general, the odometry model is preferable to the velocity model when applicable, since odometers are usually more accurate than velocity controls—especially if those velocity values are not sensed but instead submitted to a PID controller that sets the actual motor currents.

5.4.2   Sampling Algorithm

If particle filters are used for localization, we would also like to have an algorithm for *sampling* from $p(x_t \mid u_t, x_{t-1})$. Recall that particle filters (cf. Chapter 4.2.1) require samples of $p(x_t \mid u_t, x_{t-1})$, rather than a closed-form expression for computing $p(x_t \mid u_t, x_{t-1})$ for any x_{t-1}, u_t, and x_t. The algorithm **sample_motion_model_odometry**, shown in Table 5.6, implements the sampling approach. It accepts an initial pose x_{t-1} and an odometry reading u_t as input, and outputs a random x_t distributed according to $p(x_t \mid u_t, x_{t-1})$. It differs from the previous algorithm in that it randomly guesses a pose x_t (Lines 5-10), instead of computing the probability of a given x_t. As before, the sampling algorithm **sample_motion_model_odometry** is somewhat easier to im-

Start location

10 meters
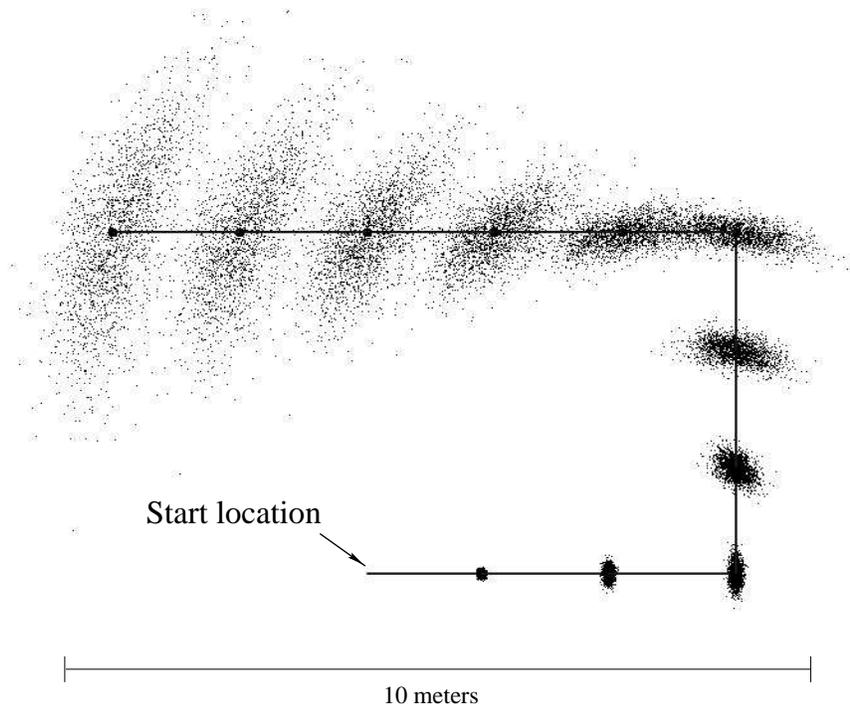
Figure 5.10 Sampling approximation of the position belief for a non-sensing robot. The solid line displays the actions, and the samples represent the robot's belief at different points in time.

plement than the closed-form algorithm **motion_model_odometry**, since it side-steps the need for an inverse model.

Figure 5.9 shows examples of sample sets generated by **sample_motion_model_odometry**, using the same parameters as in the model shown in Figure 5.8. Figure 5.10 illustrates the motion model "in action" by superimposing sample sets from multiple time steps. This data has been generated using the motion update equations of the algorithm **particle_filter** (Table 4.3), assuming the robot's odometry follows the path indicated by the solid line. The figure illustrates how the uncertainty grows as the robot moves. The samples are spread across an increasingly larger space.

5.4.3 Mathematical Derivation

The derivation of the algorithms is relatively straightforward, and once again may be skipped at first reading. To derive a probabilistic motion model using odometry, we recall that the relative difference between any two poses is represented by a concatenation of three basic motions: a rotation, a straight-line motion (translation), and another rotation. The following equations show how to calculate the values of the two rotations and the translation from the odometry reading $u_t = (\bar{x}_{t-1} \ \bar{x}_t)^T$, with $\bar{x}_{t-1} = (\bar{x} \ \bar{y} \ \bar{\theta})$ and $\bar{x}_t = (\bar{x}' \ \bar{y}' \ \bar{\theta}')$:

$$
\begin{aligned}
\delta_{\text{rot1}} &= \text{atan2}(\bar{y}' - \bar{y}, \bar{x}' - \bar{x}) - \bar{\theta} & (5.34) \\
\delta_{\text{trans}} &= \sqrt{(\bar{x} - \bar{x}')^2 + (\bar{y} - \bar{y}')^s} & (5.35) \\
\delta_{\text{rot2}} &= \bar{\theta}' - \bar{\theta} - \delta_{\text{rot1}} & (5.36)
\end{aligned}
$$

To model the motion error, we assume that the "true" values of the rotation and translation are obtained from the measured ones by subtracting independent noise ε_b with zero mean and variance b:

$$
\begin{aligned}
\hat{\delta}_{\text{rot1}} &= \delta_{\text{rot1}} - \varepsilon_{\alpha_1 |\delta_{\text{rot1}}| + \alpha_2 |\delta_{\text{trans}}|} & (5.37) \\
\hat{\delta}_{\text{trans}} &= \delta_{\text{trans}} - \varepsilon_{\alpha_3 |\delta_{\text{trans}}| + \alpha_4 |\delta_{\text{rot1}} + \delta_{\text{rot2}}|} & (5.38) \\
\hat{\delta}_{\text{rot2}} &= \delta_{\text{rot2}} - \varepsilon_{\alpha_1 |\delta_{\text{rot2}}| + \alpha_2 |\delta_{\text{trans}}|} & (5.39)
\end{aligned}
$$

As in the previous section, ε_b is a zero-mean noise variable with variance b (e.g., with normal or triangular distribution). The parameters α_1 to α_4 are robot-specific error parameters, which specify the error accrued with motion.

Consequently, the true position, x_t, is obtained from x_{t-1} by an initial rotation with angle $\hat{\delta}_{\text{rot1}}$, followed by a translation with distance $\hat{\delta}_{\text{trans}}$, followed by another rotation with angle $\hat{\delta}_{\text{rot2}}$. Thus,

$$
\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} \hat{\delta}_{\text{trans}} \ \cos(\theta + \hat{\delta}_{\text{rot1}}) \\ \hat{\delta}_{\text{trans}} \ \sin(\theta + \hat{\delta}_{\text{rot1}}) \\ \theta + \hat{\delta}_{\text{rot1}} + \hat{\delta}_{\text{rot2}} \end{pmatrix} \qquad (5.40)
$$

Notice that algorithm **sample_motion_model_odometry** implements Equations (5.34) through (5.40).

The algorithm **motion_model_odometry** is obtained by noticing that Lines 5-7 compute the motion parameters $\hat{\delta}_{\text{rot1}}$, $\hat{\delta}_{\text{trans}}$, and $\hat{\delta}_{\text{rot2}}$ for the hypothesized pose x_t, relative to the initial pose x_{t-1}. The difference of both,

$$\delta_{\text{rot1}} - \bar{\delta}_{\text{rot1}} \tag{5.41}$$

$$\delta_{\text{trans}} - \bar{\delta}_{\text{trans}} \tag{5.42}$$

$$\delta_{\text{rot2}} - \bar{\delta}_{\text{rot2}} \tag{5.43}$$

is the *error* in odometry, assuming of course that x_t is the true final pose. The error model (5.37) to (5.39) implies that the probability of these errors is given by

$$p_1 = \varepsilon_{\alpha_1 |\delta_{\text{rot1}}| + \alpha_2 |\delta_{\text{trans}}|}(\delta_{\text{rot1}} - \bar{\delta}_{\text{rot1}}) \tag{5.44}$$

$$p_2 = \varepsilon_{\alpha_3 |\delta_{\text{trans}}| + \alpha_4 |\delta_{\text{rot1}} + \delta_{\text{rot2}}|}(\delta_{\text{trans}} - \bar{\delta}_{\text{trans}}) \tag{5.45}$$

$$p_3 = \varepsilon_{\alpha_1 |\delta_{\text{rot2}}| + \alpha_2 |\delta_{\text{trans}}|}(\delta_{\text{rot2}} - \bar{\delta}_{\text{rot2}}) \tag{5.46}$$

with the distributions ε defined as above. These probabilities are computed in Lines 8-10 of our algorithm **motion_model_odometry**, and since the errors are assumed to be independent, the joint error probability is the product $p_1 \cdot p_2 \cdot p_3$ (cf., Line 11).

5.5 MOTION AND MAPS

By considering $p(x_t \mid u_t, x_{t-1})$, we defined robot motion in a vacuum. In particular, this model describes robot motion in the absence of any knowledge about the nature of the environment. In many cases, we are also given a map m, which may contain information pertaining to the places that a robot may or may not be able to navigate. For example, *occupancy maps*, which will be explained in Chapter **??**, distinguish *free* (traversable) from *occupied* terrain. The robot's pose must always be in the free space. Therefore, knowing m gives us further information about the robot pose x_t before, during, and after executing a control u_t.

This consideration calls for a motion model that takes the map m into account. We will write this model as $p(x_t \mid u_t, x_{t-1}, m)$, indicating that it considers the map m in addition to the standard variables. If m carries information relevant to pose estimation, we have

$$p(x_t \mid u_t, x_{t-1}) \quad \neq \quad p(x_t \mid u_t, x_{t-1}, m) \tag{5.47}$$

The motion model $p(x_t \mid u_t, x_{t-1}, m)$ should give better results than the map-free motion model $p(x_t \mid u_t, x_{t-1})$. We will refer to $p(x_t \mid u_t, x_{t-1}, m)$ as *map-based motion model*. The map-based motion model computes the likelihood that a robot placed in a world with map m arrives at pose x_t upon executing action u_t at pose x_{t-1}. Unfortunately, computing this motion model in closed form is difficult. This is because to compute the likelihood of being at x_t after executing action u_t, one has to incorporate the probability that an unoccupied path exists between x_{t-1} and x_t and that the robot might have followed this unoccupied path when executing the control u_t—a complex operation.

Luckily, there exists an efficient approximation for the map-based motion model, which works well if the distance between x_{t-1} and x_t is small (e.g., smaller than half a robot diameter). The approximation factorizes the map-based motion model into two components:

$$p(x_t \mid u_t, x_{t-1}, m) \quad = \quad \eta \, p(x_t \mid u_t, x_{t-1}) \, p(x_t \mid m) \tag{5.48}$$

where η is the usual normalizer. According to this factorization, we simply multiply the map-free estimate $p(x_t \mid u_t, x_{t-1})$ with a second term, $p(x_t \mid m)$, which expresses the "consistency" of pose x_t with the map m. In the case of occupancy maps, $p(x_t \mid m) = 0$ if and only if the robot "collides" with an occupied grid cell in the map; otherwise it assumes a constant value. By multiplying $p(x_t \mid m)$ and $p(x_t \mid u_t, x_{t-1})$, we obtain a distribution that assigns all probability mass to poses x_{t-1} consistent with the map, which otherwise has the same shape as $p(x_t \mid u_t, x_{t-1})$. As η can be computed by normalization, this approximation of a map-based motion model can be computed efficiently without any significant overhead compared to a map-free motion model.

Table 5.7 states the basic algorithms for computing and for sampling from the map-based motion model. Notice that the sampling algorithm returns a weighted sample, which includes an importance factor proportional to $p(x_t \mid m)$. Care has to be taken in the implementation of the sample version, to ensure termination of the inner loop. An example of the motion model is illustrated in Figure 5.11. The density in Figure 5.11a is $p(x_t \mid u_t, x_{t-1})$, computed according to the velocity motion model. Now suppose the map m possesses a long rectangular obstacle, as indicated in Figure 5.11b. The probability $p(x_t \mid m)$ is zero at all poses x_t where the robot would intersect the obstacle. Since our example robot is circular, this region is equivalent to the obstacle grown by a robot radius (this is equivalent to mapping the obstacle from *workspace* to the robot's *configuration space* [19] or *pose space*). The resulting probability $p(x_t \mid u_t, x_{t-1}, m)$, shown in Figure 5.11b, is the normalized product of $p(x_t \mid m)$ and $p(x_t \mid$

1:          **Algorithm motion_model_with_map**(x_t, u_t, x_{t-1}, m)**:**
2:               *return* $p(x_t \mid u_t, x_{t-1}) \cdot p(x_t \mid m)$

1:          **Algorithm sample_motion_model_with_map**(u_t, x_{t-1}, m)**:**
2:               *do*
3:                    $x_t = $ **sample_motion_model**(u_t, x_{t-1})
3:                    $\pi = p(x_t \mid m)$
4:               *until* $\pi > 0$
5:               *return* $\langle x_t, \pi \rangle$

Table 5.7 Algorithm for computing $p(x_t \mid u_t, x_{t-1}, m)$, which utilizes a map m of the environment. This algorithms bootstraps previous motion models (Tables 5.1, 5.3, 5.5, and 5.6) to models that take into account that robots cannot be placed in occupied space in the map m.

u_t, x_{t-1}). It is zero in the extended obstacle area, and proportional to $p(x_t \mid u_t, x_{t-1})$ everywhere else.

Figure 5.11 also illustrates a problem with our approximation. The region marked $(*)$ possesses non-zero likelihood, since both $p(x_t \mid u_t, x_{t-1})$ and $p(x_t \mid m)$ are non-zero in this region. However, for the robot to be in this particular area it must have gone through the wall, which is impossible in the real world. This error is the result of checking model consistency at the final pose x_{t-1} only, instead of verifying the consistency of the robot's path to the goal. In practice, however, such errors only occur for relatively large motions u_t, and it can be neglected for higher update frequencies.

To shed light onto the nature of the approximation, let us briefly derive it. Equation (5.48) can be obtained by applying Bayes rule:

$$p(x_t \mid u_t, x_{t-1}, m) \quad = \quad \eta \, p(m \mid x_t, u_t, x_{t-1}) \, p(x_t \mid u_t, x_{t-1}) \tag{5.49}$$

If we approximate $p(m \mid x_t, u_t, x_{t-1})$ by $p(m \mid x_t)$, we obtain the desired equation by once again applying Bayes rule:

$$p(x_t \mid u_t, x_{t-1}, m) \quad = \quad \eta \, p(m \mid x_t) \, p(x_t \mid u_t, x_{t-1})$$

(a) $p(x_t \mid u_t, x_{t-1})$ (b) $p(x_t \mid u_t, x_{t-1}, m)$

$\longleftarrow$ (∗)
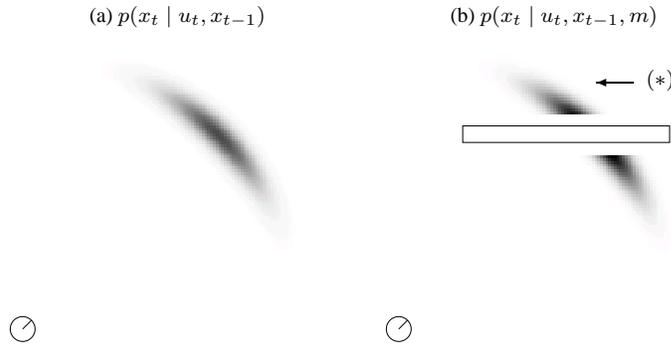
Figure 5.11 Velocity motion model (a) without a map and (b) conditioned on a map m.

$$= \ \eta \, p(x_t \mid m) \, p(x_t \mid u_t, x_{t-1}) \tag{5.50}$$

where η is the normalizer (notice that the value of η is different for the different steps in our transformation). This brief analysis shows that our map-based model is justified under the rough assumption that

$$p(m \mid x_t, u_t, x_{t-1}) \ = \ p(m \mid x_t) \tag{5.51}$$

Obviously, these expressions are not equal. When computing the conditional over m, our approximation omits two terms: u_t and x_t. By omitting these terms, we discard any information relating to the robot's path leading up to x_t. All we know is that its final pose is x_t. We already noticed the consequences of this omission in our example above, when we observed that poses behind a wall may possess non-zero likelihood. Our approximate map-based motion model may falsely assume that the robot just went through a wall, as long as the initial and final poses are in the unoccupied space. How damaging can this be? As noted above, this depends on the update interval. In fact, for sufficiently high update rates, and assuming that the noise variables in the motion model are bounded, we can guarantee that the approximation is tight and this effect will not occur.

This analysis illustrates a subtle insight pertaining to the implementation of the algorithm. In particular, one has to pay attention to the update frequency. A Bayes filter that is updated frequently might yield fundamentally different results than one that is updated occasionally only.

5.6　SUMMARY

This section derived the two principal probabilistic motion models for mobile robots operating on the plane.

- We derived an algorithm for the probabilistic motion model $p(x_t \mid u_t, x_{t-1})$ that represents control u_t by a translational and angular velocity, executed over a fixed time interval Δt. In implementing this model, we realized that two control noise parameters, one for the translational and one for the rotational velocity, are insufficient to generate a space-filling (non-generate) posterior. We therefore added a third noise parameter, expressed as a noisy "final rotation."

- We presented an alternative motion model that uses the robot's odometry as input. Odometry measurements were expressed by three parameters, an initial rotation, followed by a translation, and a final rotation. The probabilistic motion model was implemented by assuming that all three of these parameters are subject to noise. We noted that odometry readings are technically not controls; however, by using them just like controls we arrived at a simpler formulation of the estimation problem.

- For both motion models, we presented two types of implementations: One in which the probability $p(x_t \mid u_t, x_{t-1})$ is calculated in closed form, and one that enables us to generate samples from $p(x_t \mid u_t, x_{t-1})$. The closed-form expression accepts as an input x_t, u_t, and x_{t-1}, and outputs a numerical probability value. To calculate this probability, the algorithms effectively invert the motion model, to compare the *actual* with the *commanded* control parameters. The sampling model does not require such an inversion. Instead, it implements a forward model of the motion model $p(x_t \mid u_t, x_{t-1})$. It accepts as an input the values u_t and x_{t-1} and outputs a random x_t drawn according to $p(x_t \mid u_t, x_{t-1})$. Closed-form models are required for some probabilistic algorithms. Others, most notably particle filters, utilize sampling models.

- Finally we extended all motion models to incorporate a map of the environment. The resulting probability $p(x_t \mid u_t, x_{t-1}, m)$ incorporates a map m in its conditional. This extension followed the intuition that the map specifies where a robot may be; which has an effect of the ability to move from pose x_{t-1} to x_t. Our extension was approximate, in that we only checked for the validity of the final pose.

The motion models discussed here are only examples: Clearly, the field of robotic actuators is much richer than just mobile robots operating in flat terrain. Even within

the field of mobile robotics, there exist a number of devices that are not covered by the models discussed here. Examples include holonomic robots which can move sidewards. Further, our description does not consider robot dynamics, which are important for fast-moving vehicles such as cars on highways. Most of these robots can be modeled analogously: simply specify the physical laws of robot motion, and specify appropriate noise parameters. For dynamic models, this will require to extend the robot state by a velocity vector which captures the dynamic state of the vehicle. In many ways, these extensions are straightforward. Rather than cluttering this book with more motion models, it is now time to move on to the important topic of sensor measurements.

5.7   BIBLIOGRAPHICAL REMARKS

Typical drives covered by this model are the the differential drive, the Ackerman drive, or the synchro-drive [4, **?**]. Drive systems not covered by our model are those without non-holonomic constraints [19] like robots equipped with Mecanum wheels [4, **?**] or even legged robots.

6

MEASUREMENTS

6.1 INTRODUCTION

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 section, 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.

To illustrate the basic problem of mobile robots that use their sensors to perceive their environment, Figure 6.1 shows a typical 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 black and the map of the environment is shown in light grey. 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. The inability for sonar to reliably measure range to nearby objects is often paraphrased as sensor noise. Technically, this noise is quite predictable: When
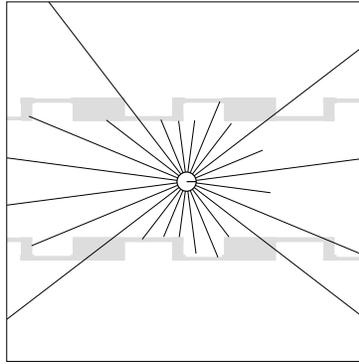
Figure 6.1   Typical ultrasound scan of a robot in its environment.

measuring smooth surfaces (such as walls) at an angle, the echo tends to travel into a direction other than the sonar sensor, as illustrated in Figure **??**. This effect is called *specular reflection* and often leads to overly large range measurements. The likelihood of specular reflection depends on a number of properties, such as the surface material and angle, the range of the surface, 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.

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 a sensor accurately, primarily for two reasons: First, developing an accurate sensor model can be extremely time-consuming, and second, an accurate model may require state variables that we might not know (such as the surface material). Probabilistic robotics accommodates the 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 write:

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

We will use z_t^k to refer to an individual measurement(e.g., one range value). We will approximate $p(z_t \mid x_t, m)$ by the product of the individual measurement likelihoods

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

Technically, this amounts to an *independence assumption* between the noise in each individual measurement value — 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 at 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:

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

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, knowns 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, $(x \ y)$.

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 which 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 suggest measurements depend on the map.

6.3   BEAM MODELS OF RANGE FINDERS

Range finders are among the most popular sensors in robotics. Our first sensor model 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

(a) Gaussian distribution p_{hit}

$p(z_t^k \mid x_t, m)$

z_t^{k*} z_{max}

(b) Exponential distribution p_{short}

$p(z_t^k \mid x_t, m)$

z_t^{k*} z_{max}

(c) Uniform distribution p_{max}

$p(z_t^k \mid x_t, m)$

z_t^{k*} z_{max}

(d) Uniform distribution p_{rand}

$p(z_t^k \mid x_t, m)$

z_t^{k*} z_{max}
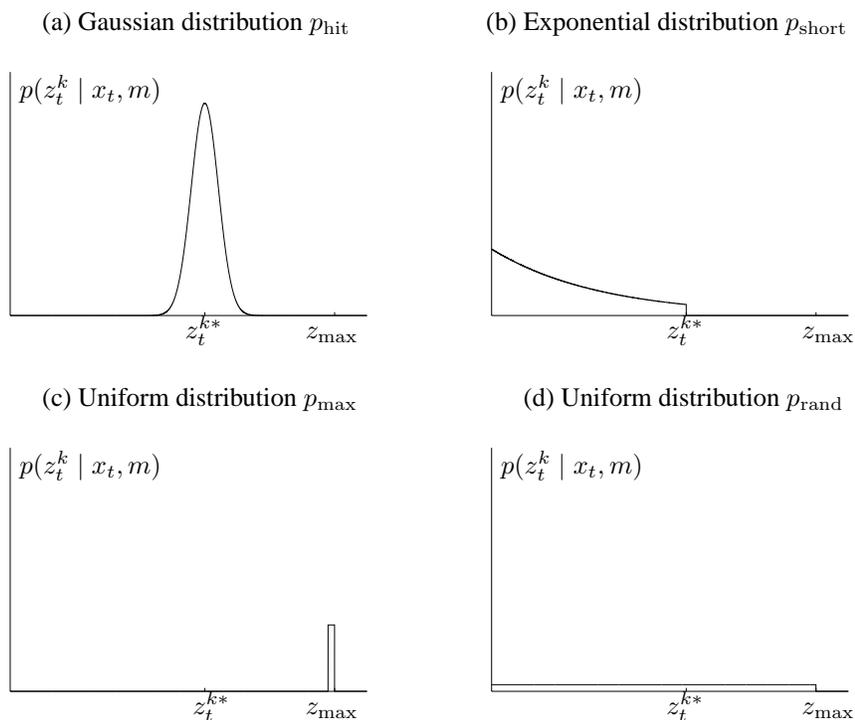
Figure 6.2   Components of the range finder sensor model. In each diagram the horizontal axis corresponds to the measurement z_t^k, the vertical to the likelihood.

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 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 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 6.2a illustrates this density p_{hit}, for a specific value of z_t^{k*}.

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

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

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

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

The normalizer η evaluates to

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

The variance 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.

More generally, 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 z_1, and the second person's range is z_2. Let us further assume that $z_1 < z_2$, without loss of generality. Then we are more likely to measure z_1 than z_2. Whenever the first person is present, our sensor measures z_1. However, for it to measure z_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 $p_{\text{short}}(z_t^k \mid x_t, m)$:

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

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

$$\int_0^{z_t^{k*}} \lambda_{\text{short}} \, e^{-\lambda_{\text{short}} z_t^k} \, dz_t^k \quad = \quad -e^{-\lambda_{\text{short}} z_t^{k*}} + e^{-\lambda_{\text{short}} 0} \qquad (6.8)$$

$$= \quad 1 - e^{-\lambda_{\text{short}} z_t^{k*}} \qquad (6.9)$$

the value of η can be derived as:

$$\eta \quad = \quad \frac{1}{1 - e^{-\lambda_{\text{short}} z_t^{k*}}} \qquad (6.10)$$

Figure 6.2b 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 with sonar sensors when measuring a surface at a steep angle. Failures also occur with laser range finders when sensing black, light-absorbing objects, or when measuring objects in bright light. A typical result of sensor failures are *max-range measurements*: 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.

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

$$p_{\text{max}}(z_t^k \mid x_t, m) \quad = \quad I(z = z_{\text{max}}) \quad = \quad \begin{cases} 1 & \text{if } z = z_{\text{max}} \\ 0 & \text{otherwise} \end{cases} \qquad (6.11)$$

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).
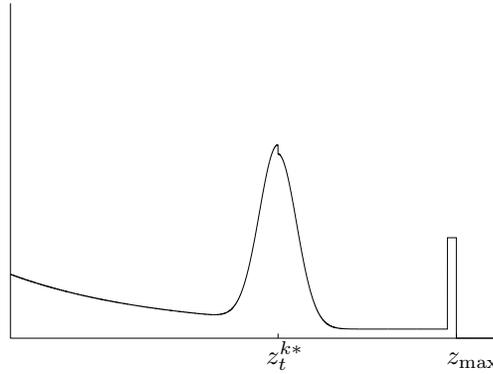
Figure 6.3 "Pseudo-density" of a typical mixture distribution $p(z_t^k \mid x_t, m)$.

4. **Random measurements.** Finally, range finders occasionally produce entirely unexplained 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 $[0; z_{\max}]$:

$$p_{\mathrm{rand}}(z_t^k \mid x_t, m) \quad = \quad \begin{cases} \frac{1}{z_{\max}} & \text{if } 0 \leq z_t^k < z_{\max} \\ 0 & \text{otherwise} \end{cases} \tag{6.12}$$

Figure 6.2d 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 $z_{\mathrm{hit}} + z_{\mathrm{short}} + z_{\max} + z_{\mathrm{rand}} = 1$.

$$p(z_t^k \mid x_t, m) \quad = \quad \begin{pmatrix} z_{\mathrm{hit}} \\ z_{\mathrm{short}} \\ z_{\max} \\ z_{\mathrm{rand}} \end{pmatrix}^T \cdot \begin{pmatrix} p_{\mathrm{hit}}(z_t^k \mid x_t, m) \\ p_{\mathrm{short}}(z_t^k \mid x_t, m) \\ p_{\max}(z_t^k \mid x_t, m) \\ p_{\mathrm{rand}}(z_t^k \mid x_t, m) \end{pmatrix} \tag{6.13}$$

A typical density resulting from this linear combination of the individual densities is shown in Figure 6.3 (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:	$p = z_{\text{hit}} \cdot p_{\text{hit}}(z_t^k \mid x_t, m) + z_{\text{short}} \cdot p_{\text{short}}(z_t^k \mid x_t, m)$
6:	$+ z_{\text{max}} \cdot p_{\text{max}}(z_t^k \mid x_t, m) + z_{\text{rand}} \cdot p_{\text{rand}}(z_t^k \mid x_t, m)$
7:	$q = q \cdot p$
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 (6.13). 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.

One way to determine the intrinsic parameters is to rely on data. Figure 6.4 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 meter (between 2.9m and 3.1m). The left plot

(a) Sonar data   (b) Laser data
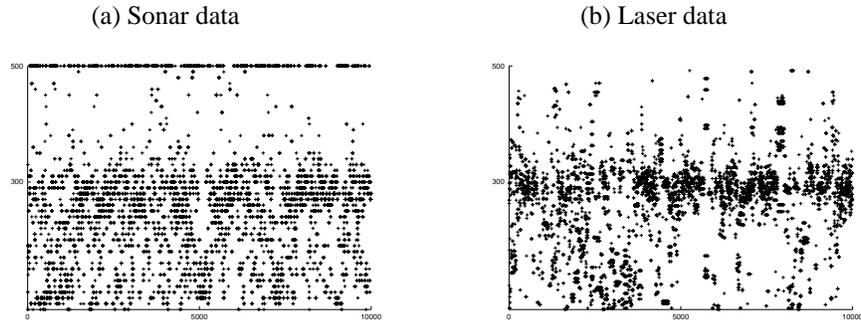
Figure 6.4   Typical data obtained with (a) a sonar sensor and (b) a laser-range sensor in
an office environment for a "true" range of 300 cm and a maximum range of 500 cm.

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

$$p(Z \mid X, m, \Theta), \tag{6.14}$$

and our goal is to identify intrinsic parameters Θ that maximize this likelihood. Algo-
rithms that maximize the likelihood of data are known as *maximum likelihood* estima-
tors, or ML estimators in short.

Table 6.2 depicts the algorithm **learn_intrinsic_parameters**, which is an algorithm
for calculating the maximum likelihood estimate for the intrinsic parameters. As we
shall see below, the algorithm is an instance of the *expectation maximization* algo-
rithm, an iterative procedure for estimating ML parameters. Initially, the algorithm

---

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

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.

learn_intrinsic_parameters 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 estimates 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.

(a) Sonar data
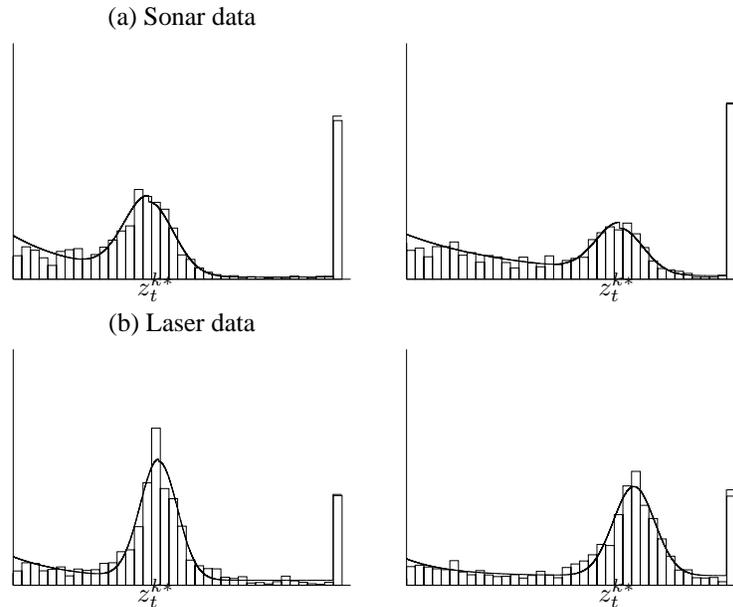
(b) Laser data

Figure 6.5 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 6.4.

Figure 6.5 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 however, this model is less susceptible to unmodeled *systematic* perturbations, such as people who block the robot's path for long periods of time.

(a) Laser scan and part of the map

(b) Likelihood for different positions
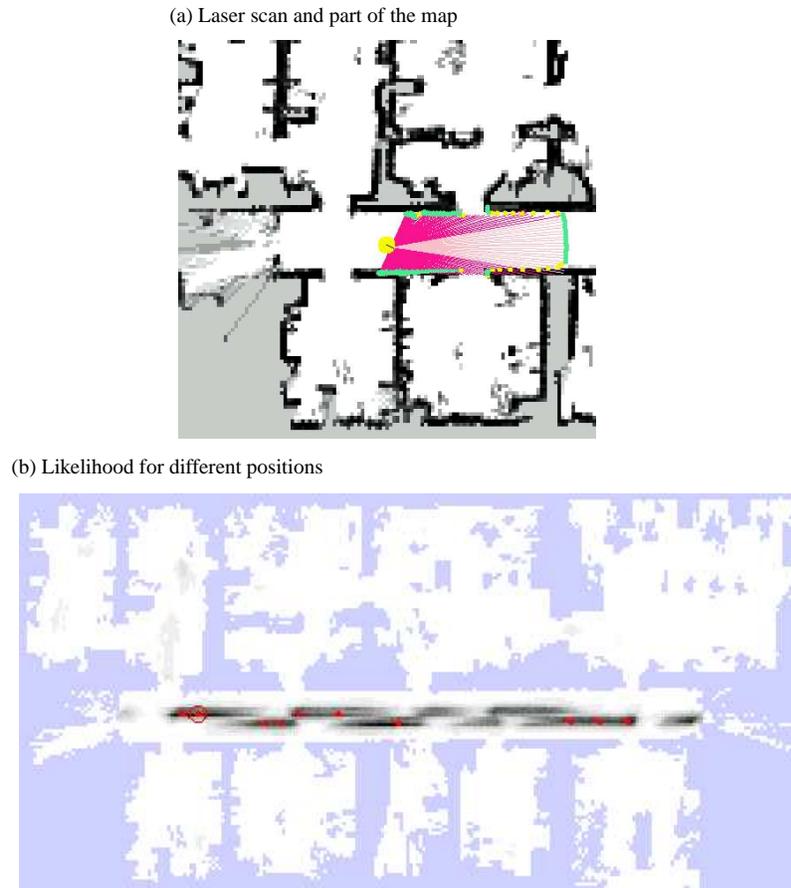
Figure 6.6 Probabilistic model of perception: (a) Laser range scan, projected into a previously acquired map m. (b) The likelihood $p(z_t \mid x_t, m)$, evaluated for all positions x_t and projected into the map (shown in gray). The darker a position, the larger $p(z_t \mid x_t, m)$.

Figure 6.6 illustrates the learned sensor model in action. Shown in Figure 6.6a is a 180 degree range scan. The robot is placed in a previously acquired occupancy grid map at its true pose. Figure 6.6b 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 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.

6.3.3   Mathematical Derivation

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, that is, 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 normalized ratios:

$$
\begin{pmatrix} z_{\text{hit}} \\ z_{\text{short}} \\ z_{\text{max}} \\ z_{\text{rand}} \end{pmatrix}
= |Z|^{-1}
\begin{pmatrix} |Z_{\text{hit}}| \\ |Z_{\text{short}}| \\ |Z_{\text{max}}| \\ |Z_{\text{rand}}| \end{pmatrix}
\tag{6.15}
$$

The remaining intrinsic parameters, σ_{hit} and λ_{short}, are obtained as follows. For the
data set Z_{hit}, we get from (6.5)

$$
\begin{aligned}
p(Z_{\text{hit}} \mid X, m, \Theta) &= \prod_{z_i \in Z_{\text{hit}}} p_{\text{hit}}(z_i \mid x_i, m, \Theta) \\
&= \prod_{z_i \in Z_{\text{hit}}} \frac{1}{\sqrt{2\pi\sigma_{\text{hit}}^2}} \, e^{-\frac{1}{2}\frac{(z_i - z_i^*)^2}{\sigma_{\text{hit}}^2}}
\end{aligned}
\tag{6.16}
$$

Here z_i^* is the "true" range, computed from the pose x_i and the map m. A clas-
sical 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

$$\log p(Z_{\text{hit}} \mid X, m, \Theta) = \sum_{z_i \in Z_{\text{hit}}} \left[-\frac{1}{2} \log 2\pi\sigma_{\text{hit}}^2 - \frac{1}{2} \frac{(z_i - z_i^*)^2}{\sigma_{\text{hit}}^2} \right], \quad (6.17)$$

which is now easily transformed as follows

$$\begin{aligned} \log p(Z_{\text{hit}} \mid X, m, \Theta) &= -\frac{1}{2} \sum_{z_i \in Z_{\text{hit}}} \left[\log 2\pi\sigma_{\text{hit}}^2 + \frac{(z_i - z_i^*)^2}{\sigma_{\text{hit}}^2} \right] \\ &= -\frac{1}{2} \left[|Z_{\text{hit}}| \log 2\pi + 2|Z_{\text{hit}}| \log \sigma_{\text{hit}} + \sum_{z_i \in Z_{\text{hit}}} \frac{(z_i - z_i^*)^2}{\sigma_{\text{hit}}^2} \right] \\ &= \text{const.} - |Z_{\text{hit}}| \log \sigma_{\text{hit}} - \frac{1}{2\sigma_{\text{hit}}^2} \sum_{z_i \in Z_{\text{hit}}} (z_i - z_i^*)^2 \quad (6.18) \end{aligned}$$

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

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

$$(6.19)$$

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.

$$\sigma_{\text{hit}} = \sqrt{\frac{1}{|Z_{\text{hit}}|} \sum_{z_i \in Z_{\text{hit}}} (z_i - z_i^*)^2} \quad (6.20)$$

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

$$\begin{aligned} p(Z_{\text{short}} \mid X, m, \Theta) &= \prod_{z_i \in Z_{\text{short}}} p_{\text{short}}(z_i \mid x_i, m) \\ &= \prod_{z_i \in Z_{\text{short}}} \lambda_{\text{short}} \, e^{-\lambda_{\text{short}} z_i} \quad (6.21) \end{aligned}$$

The logarithm is given by

$$
\begin{aligned}
\log p(Z_{\text{short}} \mid X, m, \Theta) &= \sum_{z_i \in Z_{\text{short}}} \log \lambda_{\text{short}} - \lambda_{\text{short}} z_i \\
&= |Z_{\text{short}}| \log \lambda_{\text{short}} - \lambda_{\text{short}} \sum_{z_i \in Z_{\text{short}}} z_i \qquad (6.22)
\end{aligned}
$$

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

$$
\frac{\partial \log p(Z_{\text{short}} \mid X, m, \Theta)}{\partial \lambda_{\text{short}}} = \frac{|Z_{\text{short}}|}{\lambda_{\text{short}}} - \sum_{z_i \in Z_{\text{short}}} z_i \qquad (6.23)
$$

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

$$
\lambda_{\text{short}} = \frac{|Z_{\text{short}}|}{\sum_{z_i \in Z_{\text{short}}} z_i} \qquad (6.24)
$$

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 which calculates an expectation for the c_i's and one that computes the intrinsic model parameters under these expectations. The resulting algorithm is an instance of the *expectation maximization* algorithm, or EM in short. We will encounter EM in later chapters of this book, so this might be a good opportunity for the reader to familiarize herself with the basic EM algorithm.

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

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

$$(6.25)$$

This expression can be rewritten using the variables c_i:

$$
\begin{aligned}
\log p(Z \mid X, m, \Theta) \;=\; & \sum_{z_i \in Z} I(c_i = \text{hit}) \; \log p_{\text{hit}}(z_i \mid x_i, m) \\
& + I(c_i = \text{short}) \; \log p_{\text{short}}(z_i \mid x_i, m) \\
& + I(c_i = \text{max}) \; \log p_{\text{max}}(z_i \mid x_i, m) \\
& + I(c_i = \text{rand}) \; \log p_{\text{rand}}(z_i \mid x_i, m)
\end{aligned}
\tag{6.26}
$$

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:

$$
\begin{aligned}
E[\log &\, p(Z \mid X, m, \Theta)] \\
=\; & \sum_{i} p(c_i = \text{hit}) \; \log p_{\text{hit}}(z_i \mid x_i, m) + p(c_i = \text{short}) \; \log p_{\text{short}}(z_i \mid x_i, m) \\
& + p(c_i = \text{max}) \; \log p_{\text{max}}(z_i \mid x_i, m) + p(c_i = \text{rand}) \; \log p_{\text{rand}}(z_i \mid x_i, m) \\
=:\; & \sum_{i} e_{i,\text{hit}} \; \log p_{\text{hit}}(z_i \mid x_i, m) + e_{i,\text{short}} \; \log p_{\text{short}}(z_i \mid x_i, m) \\
& + e_{i,\text{max}} \; \log p_{\text{max}}(z_i \mid x_i, m) + e_{i,\text{rand}} \; \log p_{\text{rand}}(z_i \mid x_i, m)
\end{aligned}
\tag{6.27}
$$

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.

$$
\begin{pmatrix} e_{i,\text{hit}} \\ e_{i,\text{short}} \\ e_{i,\text{max}} \\ e_{i,\text{rand}} \end{pmatrix}
:=
\begin{pmatrix} p(c_i = \text{hit}) \\ p(c_i = \text{short}) \\ p(c_i = \text{max}) \\ p(c_i = \text{rand}) \end{pmatrix}
=
\eta
\begin{pmatrix} p_{\text{hit}}(z_i \mid x_i, m) \\ p_{\text{short}}(z_i \mid x_i, m) \\ p_{\text{max}}(z_i \mid x_i, m) \\ p_{\text{rand}}(z_i \mid x_i, m) \end{pmatrix}
\tag{6.28}
$$

where the normalizer is given by

$$
\begin{aligned}
\eta \;=\; & [\, p_{\text{hit}}(z_i \mid x_i, m) + p_{\text{short}}(z_i \mid x_i, m) \\
& + p_{\text{max}}(z_i \mid x_i, m) + p_{\text{rand}}(z_i \mid x_i, m) \,]^{-1}
\end{aligned}
\tag{6.29}
$$

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

$$
\begin{pmatrix} z_{\text{hit}} \\ z_{\text{short}} \\ z_{\text{max}} \\ z_{\text{rand}} \end{pmatrix} = |Z|^{-1} \sum_i \begin{pmatrix} e_{i,\text{hit}} \\ e_{i,\text{short}} \\ e_{i,\text{max}} \\ e_{i,\text{rand}} \end{pmatrix} \tag{6.30}
$$

The ML parameters σ_{hit} and λ_{short} are then obtained analogously, by replacing the hard assignments in (6.20) and (6.24) by soft assignments weighted by the expectations.

$$
\sigma_{\text{hit}} = \sqrt{\frac{1}{\sum_{z_i \in Z} e_{i,\text{hit}}} \sum_{z_i \in Z} e_{i,\text{hit}} (z_i - z_i^*)^2} \tag{6.31}
$$

and

$$
\lambda_{\text{short}} = \frac{\sum_{z_i \in Z} e_{i,\text{short}}}{\sum_{z_i \in Z} e_{i,\text{short}} \, z_i} \tag{6.32}
$$

6.3.4 Practical Considerations

In practice, computing the densities of all sensor readings can be quite time-consuming. For example, laser range scanners often return hundreds of values per scan, at a rate of several scans per seconds. Since one has to perform a ray-tracing 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 by leaving out 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 alpha, 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.

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-computing 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 will be further 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 speedups by an order of magnitude over the plain implementation that casts rays online.

6.4 LIKELIHOOD FIELDS FOR RANGE FINDERS

6.4.1 Basic Algorithm

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

- **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 (in particular in the heading direction θ_t). Lack of smoothness has two problematic consequences. First, any approximate belief representation runs danger to miss 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.

- **Computational complexity.** 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.

We will now describe an alternative model, called *likelihood field model*, 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. In particular, the resulting posteriors are much smoother even in cluttered space, and the computation is typically 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 project an individual sensor measurement z_t^k into the global coordinate frame of the map m, we need to know where in global coordinates the robot's coordinate system is, where on the robot the sensor beam z_k originates, and where it 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 $(x_{k,sens} \ \ y_{k,\text{sens}})^T$, 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.

$$
\begin{pmatrix} x_{z_t^k} \\ y_{z_t^k} \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x_{k,\text{sens}} \\ y_{k,\text{sens}} \end{pmatrix} + z_t^k \begin{pmatrix} \cos(\theta + \theta_{k,\text{sens}}) \\ \sin(\theta + \theta_{k,\text{sens}}) \end{pmatrix}
$$
(6.33)

These coordinates are only meaningful when the sensor detects an obstacle. If the range sensor takes on its maximum value, that is, $z_t^k = z_{\max}$, 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.

(a) example environment
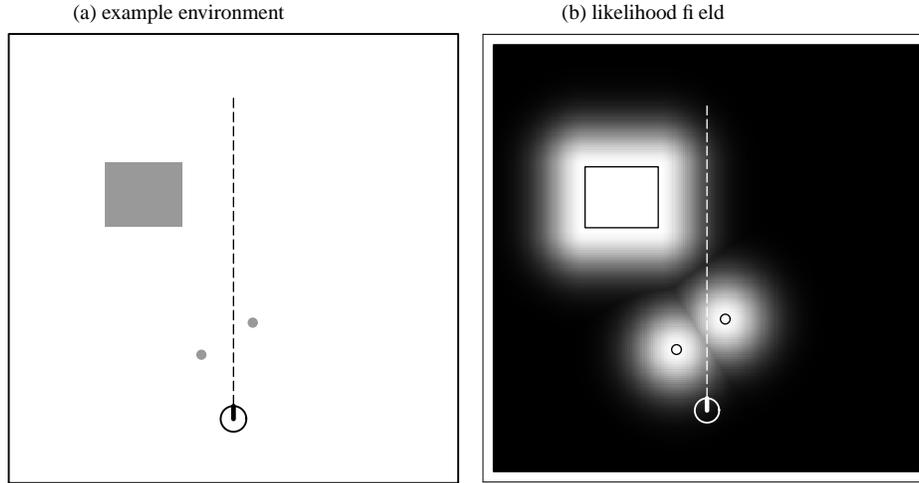
(b) likelihood field

Figure 6.7 (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 **??**.

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 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 modeling the sensor noise:

$$p_{\mathrm{hit}}(z_t^k \mid x_t, m) \quad = \quad \varepsilon_{\sigma_{\mathrm{hit}}^2}(dist^2) \tag{6.34}$$

Figure 6.7a depicts a map, and Figure 6.7b 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 6.7. The resulting function is the one shown in Figure 6.8a.

(a) $p_{\mathrm{hit}}(z_t^k \mid x_t, m)$ (b) $p(z_t^k \mid x_t, m)$

o_1 o_2 o_3 $z_{\max}$ o_1 o_2 o_3 $z_{\max}$
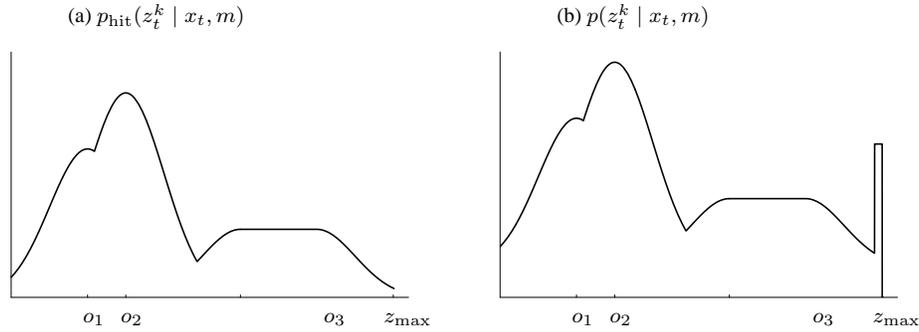
Figure 6.8   (a) Probability $p_{\mathrm{hit}}(z_t^k)$ as a function of the measurement z_t^k, for the situation depicted in Figure 6.7. Here the sensor beam passes by three obstacles, with respective nearest points o_1, o_2, and o_3. (b) Sensor probability $p(z_t^k \mid x_t, m)$, obtained for the situation depicted in Figure 6.7, obtained by adding two uniform distributions.

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. **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:

$$z_{\mathrm{hit}} \cdot p_{\mathrm{hit}} + z_{\mathrm{rand}} \cdot p_{\mathrm{rand}} + z_{\max} \cdot p_{\max} \qquad\qquad (6.35)$$

using the familiar mixing weights z_{hit}, z_{rand}, and $z_{\max}$. Figure 6.8b 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 6.8a, 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 6.7b, 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

---

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:                 $dist^2 = \min\limits_{x',y'} \left\{ (x_{z_t^k} - x')^2 + (y_{z_t^k} - y')^2 \,\middle|\, \langle x', y' \rangle \text{ occupied in } m \right\}$
8:                 $q = q \cdot \left(z_{\text{hit}} \cdot \mathbf{prob}(dist^2, \sigma_{\text{hit}}^2) + \frac{z_{\text{random}}}{z_{\max}} \right)$
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 $\mathbf{prob}(dist^2, \sigma_{\text{hit}}^2)$ computes the probability of the distance under a zero-centered Gaussian distribution with variance σ_{hit}^2.

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 (the function $\mathbf{prob}(dist^2, \sigma_{\text{hit}}^2)$ computes the probability of $dist^2$ under a zero-centered Gaussian distribution with variance σ_{hit}^2).

The most costly operation in algorithm **likelihood_field_range_finder_model** is the search for the nearest neighbor in the map (Line 7). 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 $p(z_t^k \mid$
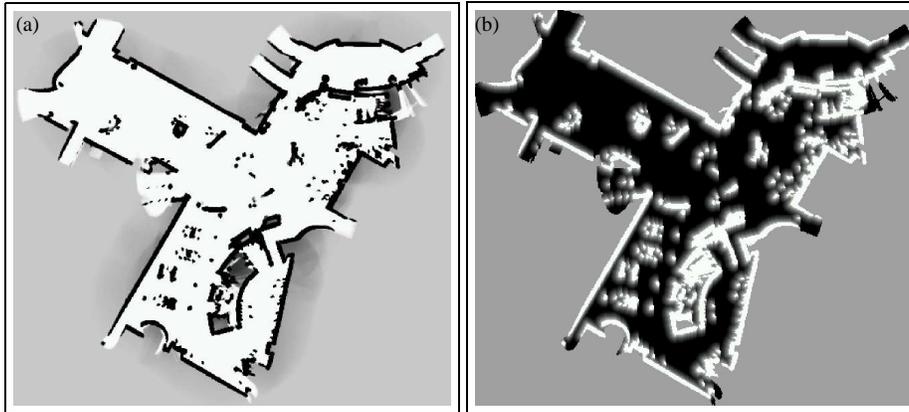
Figure 6.9 (a) Occupancy grid map of the San Jose Tech Museum, (b) pre-processed likelihood field.

x_t, m). Another key advantage is that the pre-computation takes place in 2D, instead of 3D, increasing the compactness of the pre-computed information.

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, that is, 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 that in the unexplored space, every sensor measurement is equally likely, hence $p(z_t^k \mid x_t, m)$ is modeled by a uniform distribution with density $\frac{1}{z_{\max}}$. Figure 6.9 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 precompute the nearest neighbor for a fine-grained 2-D grid.

(a) sensor scan   (b) likelihood field
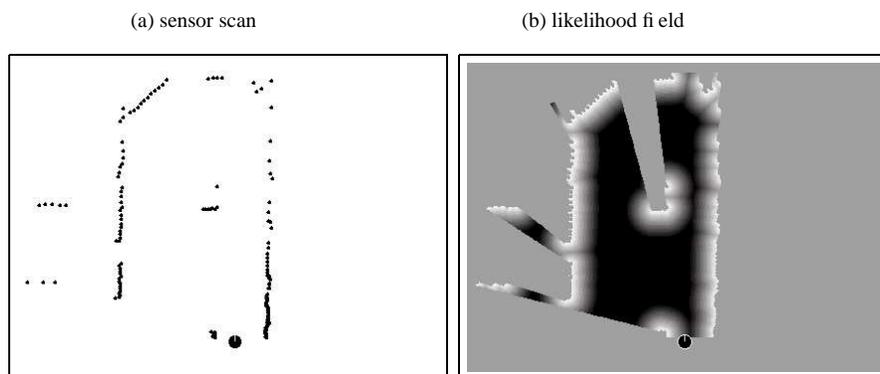
Figure 6.10 (a) Sensor scan, from a bird's eye perspective. The robot is placed at the bottom of this figure, generating a proximity scan that consists of the 180 dots in front of the robot. (b) Likelihood function generated from this sensor scan. The darker a region, the smaller the likelihood for sensing an object there. Notice that occluded regions are white, hence infer no penalty.

Likelihood fields over the visible space can also be defined for the most recent scan, which in fact defines a local map. A straightforward extension is then to match a scan and a map symmetrically: In addition to calculating the likelihood of a scan field inside the map, a symmetric extension also calculates the likelihood of each (nearby) object in the map object relative to the likelihood field of a scan. Such a symmetric routine will be of importance in future chapters on mapping, in which we seek to align scans with maps. In the interest of brevity, we omit the derivation of the resulting algorithm, which in fact is mostly a straightforward extension of the one shown in Table 6.3. However, we note that the leading implementations of the likelihood field technique rely on the extended symmetric algorithm.

6.5 CORRELATION-BASED SENSOR MODELS

There exists 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 6.11 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
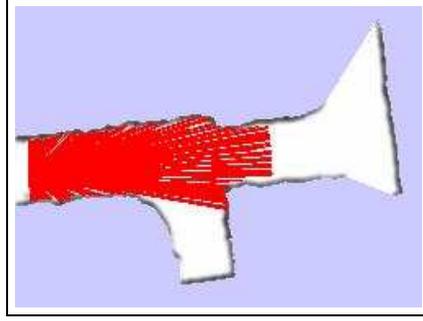
Figure 6.11　Example of a local map generated from 10 range scans, one of which is shown.

that the more similar m and m_{local}, the larger $p(m_{\mathrm{local}} \mid x_t, m)$. 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 (6.33) of sensor measurements used in the likelihood field model. If the robot is at location x_t, we denote by $m_{x,y,\mathrm{local}}(x_t)$ the grid cell in the local map that 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:

$$\rho_{m,m_{\mathrm{local}},x_t} \;\; = \;\; \frac{\sum_{x,y}(m_{x,y} - \bar{m}) \cdot (m_{x,y,\mathrm{local}}(x_t) - \bar{m})}{\sqrt{\sum_{x,y}(m_{x,y} - \bar{m})^2 \sum_{x,y}(m_{x,y,\mathrm{local}}(x_t) - \bar{m})^2}} \tag{6.36}$$

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

$$\bar{m} \;\; = \;\; \frac{1}{2N}\sum_{x,y}(m_{x,y} + m_{x,y,\mathrm{local}}), \tag{6.37}$$

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

$$p(m_{\mathrm{local}} \mid x_t, m) \;\; = \;\; \max\{\rho_{m,m_{\mathrm{local}},x_t}, 0\} \tag{6.38}$$

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 SENSOR MODELS

6.6.1 Feature Extraction

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, 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 window stills; 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.

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. This 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 multi-dimensional vector characterizing a landmark (e.g., height and color).

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

$$f(z_t) \;\; = \;\; \{f_t^1, f_t^2, \ldots\} \;\; = \;\; \{ \begin{pmatrix} r_t^1 \\ \phi_t^1 \\ s_t^1 \end{pmatrix}, \begin{pmatrix} r_t^2 \\ \phi_t^1 \\ s_t^2 \end{pmatrix}, \ldots \} \tag{6.39}$$

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

$$p(f(z_t) \mid x_t, m) \;\; = \;\; \prod_i p(r_t^i, \phi_t^i, s_t^i \mid x_t, m) \tag{6.40}$$

Conditional independence applies if the noise in each individual measurement $(r_t^i \ \phi_t^i \ s_t^i)^T$ is independent of the noise in other measurements $(r_t^j \ \phi_t^j \ s_t^j)^T$ (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 list 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, an 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 $x_t = (x \ y \ \theta)^T$.

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

(6.41)

Here $\varepsilon_{\sigma_r^2}$, $\varepsilon_{\sigma_\phi^2}$, and $\varepsilon_{\sigma_s^2}$ are zero-mean Gaussian error variables with variances σ_r^2, σ_ϕ^2, and σ_s^2, respectively.

6.6.3 Sensor Model With Known Correspondence

To implement this measurement model, we need to define a variable that establishes correspondence 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

---

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:              $q = \textbf{prob}(r_t^i - \hat{r}, \sigma_r^2) \cdot \textbf{prob}(\phi_t^i - \hat{\phi}, \sigma_\phi^2) \cdot \textbf{prob}(s_t^i - s_j, \sigma_s^2)$

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. It's output is the numerical probability $p(f_t^i \mid c_t^i, m, x_t)$.

landmarks; it is also of great relevance for the topic of robotic mapping, in which the robot regularly encounters previously unobserved landmarks.

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 (6.41).

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

---

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:　　　　　$\hat{r} = r_t^i + \textbf{sample}(\sigma_r^2)$

5:　　　　　$\hat{\phi} = \phi_t^i + \textbf{sample}(\sigma_\phi^2)$

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.

prior is uniform (it generally is not!). Bayes rule then suggests that

$$
\begin{aligned}
p(x_t \mid f_t^i, c_t^i, m) &= \eta \, p(f_t^i \mid c_t^i, x_t, m) \, p(x_t \mid c_t^i, m) \\
&= \eta \, p(f_t^i \mid c_t^i, x_t, m)
\end{aligned}
\tag{6.42}
$$

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 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 6.12 illustrates the pose distribution $p(x_t \mid f_t^i, c_t^i, m)$ (left diagram) and also shows　　　　　　　　　　　　　a　　　　　　　　　　　　　sample
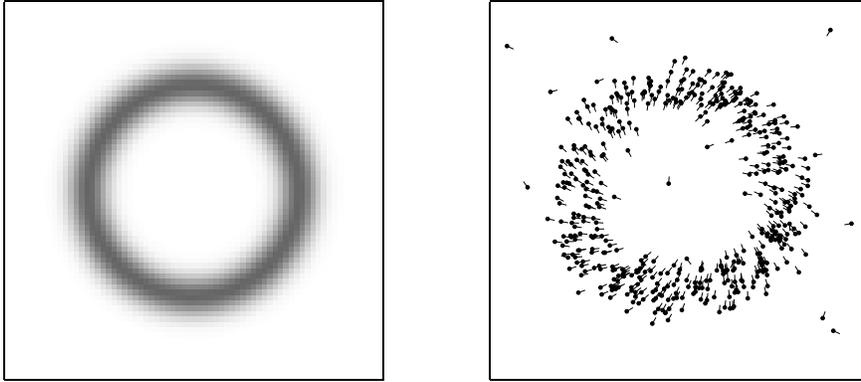
Figure 6.12 Landmark detection model: (a) Posterior distribution of the robot's pose given that it detected a landmark in 5m distance and 30deg relative bearing (projected onto 2D). (b) Sample robot poses generated from such a detection. The lines indicate the orientation of the poses.

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

We also introduced a signature value for landmarks. Most published algorithms do not make the use of apearance features explicit. When the signature is not provided, all landmarks look equal, and the data association problem of estimating the correspondence variables is even harder. We have included the signature in our model because it is a valuable source of information that can often be easily extracted form 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 underly the feature 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 a sufficient statistic of the measurement vector z_t, that is

$$p(f(z_t) \mid x_t, m) \quad \approx \quad p(z_t \mid x_t, m) \tag{6.43}$$

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 window stills, 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 of great importance. They enable us to introduce the basic concepts in probabilistic robotics, and with proper treatement of problems such as the correspondence problem they can be brought to bear even in cases where maps are cmposes 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 Section 6.3.4, is a much better way, in that it does not introduce additional variance in the outcome of a probabilistic algorithm.

6.8   SUMMARY

This section described probabilistic measurement models.

- Starting with models for range finders—and lasers in particular—we also 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 measurement distinct features. In doing so, they reduce the dimensionality of the sensor measurement 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.

7

MOBILE ROBOT LOCALIZATION

7.1 INTRODUCTION

This chapter presents a number of probabilistic algorithms for mobile robot localization. Mobile robot localization is the problem of determining the pose of a robot relative to a given map of the environment. It is often called *position estimation* or *position tracking*. Mobile robot localization is an instance of the general localization problem, which is the most basic perceptual problem in robotics. This is because nearly all robotics tasks require knowledge of the location of the robots and the objects that are being manipulated (although not necessarily within a global map).

Localization can be seen as a problem of coordinate transformation. Maps are described in a global coordinate system, which is independent of a robot's pose. Localization is the process of establishing correspondence between the map coordinate system and the robot's local coordinate system. Knowing this coordinate transformation enables the robot to express the location of objects of interests within its own coordinate frame—a necessary prerequisite for robot navigation. As the reader easily verifies, knowing the pose $x_t = (x \ \ y \ \ \theta)^T$ of the robot is sufficient to determine this coordinate transformation, assuming that the pose is expressed in the same coordinate frame as the map.

Unfortunately—and herein lies the problem of mobile robot localization—the pose can usually *not* be sensed directly. Put differently, most robots do not possess a (noise-free!) sensor for measuring pose. The pose has therefore to be inferred from data. A key difficulty arises from the fact that a single sensor measurement is usually insufficient to determine the pose. Instead, the robot has to integrate data over time to determine its pose. To see why this is necessary, just picture a robot located inside a

157
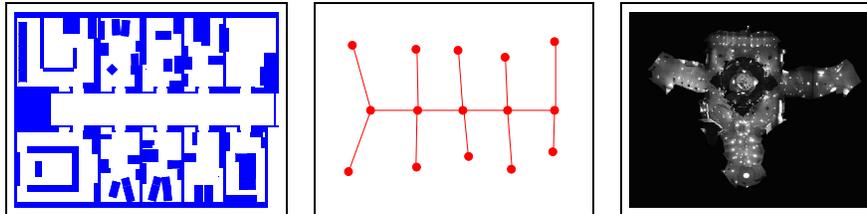
Figure 7.1   Example maps used for robot localization: a 2D metric layout, a graph-like topological map, and an image mosaic of a ceiling

building where many corridors look alike. Here a single sensor measurement (e.g., a range scan) is usually insufficient to disambiguate the identity of the corridor.

Localization has been applied in conjunction with a broad set of map representations. We already discussed two types of maps in the previous chapter: feature-based and location-based. An example of the latter were occupancy grid maps, which were informally discussed and are subject to a later chapter in this book. Instances of such maps are shown in Figure 7.1. This figure shows a hand-drawn metric 2-D map, a graph-like topological map, and an image mosaic of a ceiling (which can also be used as a map). The space of map representations used in today's research is huge. A number of later chapters will investigate specific map types and discuss algorithms for acquiring maps from data. Localization assumes that an accurate map is available.

In this and the subsequent chapter, we will present some basic probabilistic algorithms for mobile localization. All of these algorithms are variants of the basic Bayes filter described in Chapter 2. We will discuss the advantages and shortcomings of each representation and associated algorithm. The chapter also goes through a series of extensions that address different localization problems, as defined through the following taxonomy of robot localization problems.

7.2   A TAXONOMY OF LOCALIZATION PROBLEMS

Not every localization problem is equally hard. To understand the difficulty of a localization problem, we will now discuss a brief taxonomy of localization problems. This taxonomy will divide localization problems along a number of important dimensions

pertaining to the nature of the environment and the initial knowledge that a robot may possess relative to the localization problem.

Local Versus Global Localization

Localization problems are characterized by the type of knowledge that is available initially and at run-time. We distinguish three types of localization problems with an increasing degree of difficulty.

- **Position tracking.** Position tracking assumes that the *initial* robot pose is known. Localizing the robot can be achieved by accommodating the noise in robot motion. The effect of such noise is usually small. Hence, methods for position tracking often rely on the assumption that the pose error is small. The pose uncertainty is often approximated by a unimodal distribution (e.g., a Gaussian). The position tracking problem is a *local* problem, since the uncertainty is local and confined to region near the robot's true pose.

- **Global localization.** Here the initial pose of the robot is unknown. The robot is initially placed somewhere in its environment, but it lacks knowledge of where it is. Approaches to global localization cannot assume boundedness of the pose error. As we shall see later in this chapter, unimodal probability distributions are usually inappropriate. Global localization is more difficult than position tracking; in fact, it subsumes the position tracking problem.

- **Kidnapped robot problem.** This problem is a variant of the global localization problem, but one that is even more difficult. During operation, the robot can get kidnapped and teleported to some other location. The kidnapped robot problem is more difficult than the global localization problem, in that the robot might believe it knows where it is while it does not. In global localization, there robots knows that it doesn't know where it is. One might argue that robots are rarely kidnapped in practice. The practical importance of this problem, however, arises from the observation that most state-of-the-art localization algorithms cannot be guaranteed never to fail. The ability to recover from failures is essential for truly autonomous robots. Testing a localization algorithm by kidnapping it measures its ability to recover from global localization failures.

Static Versus Dynamic Environments

A second dimension that has a substantial impact on the difficulty of localization is the environment. Environments can be static or dynamic.

- **Static environments.** Static environments are environments where the only variable quantity (state) is the robot's pose. Put differently, only the robot moves in static environment. All other objects in the environments remain at the same location forever. Static environments have some nice mathematical properties that make them amenable to efficient probabilistic estimation.

- **Dynamic environments.** Dynamic environments possess objects other than the robot whose location or configuration changes over time. Of particular interest are changes that persist over time, and that impact more than a single sensor reading. Changes that are not measurable are of course of no relevance to localization, and those that affect only a single measurement are best treated as noise (*cf.* Chapter 2.4.4). Examples of more persistent changes are: people, daylight (for robots equipped with cameras), movable furniture, or doors. Clearly, most real environment are dynamic, with state changes occurring at a range of different speeds.

Obviously, localization in dynamic environments is more difficult than localization in static ones. There are two principal approaches for accommodating dynamics: First, dynamic entities might be included in the state vector. As a result, the Markov assumption might now be justified, but such an approach carries the burden of additional computational and modeling complexity; in fact, the resulting algorithm becomes effectively a mapping algorithm. Second, in certain situations sensor data can be filtered so as to eliminate the damaging effect of unmodeled dynamics. Such an approach will be described further below in Section 8.4.

Passive Versus Active Approaches

A third dimension that characterizes different localization problems pertains to the fact whether or not the localization algorithm controls the motion of the robot. We distinguish two cases:

- **Passive localization.** In passive approaches, the localization module only *observes* the robot operating. The robot is controlled through some other means, and the robot's motion is not aimed at facilitating localization. For example, the robot might move randomly or perform its everyday's tasks.

- **Active localization.** Active localization algorithms control the robot so as to minimize the localization error and/or the costs arising from moving a poorly localized robot into a hazardous place.
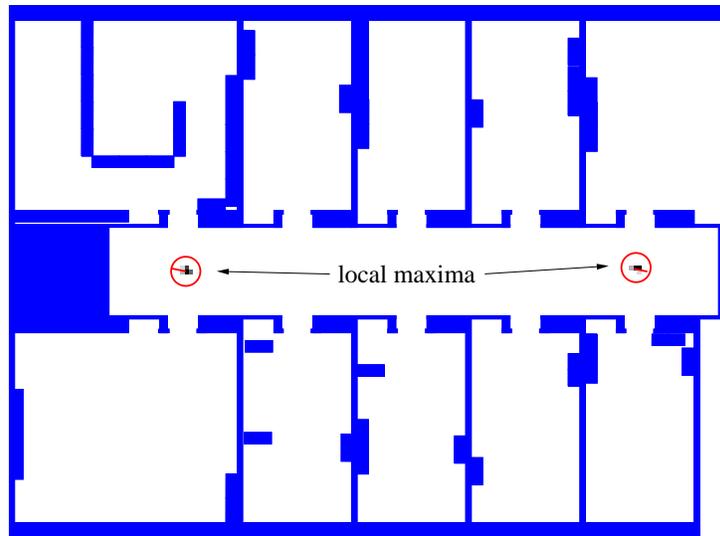
Figure 7.2 Example situation that shows a typical belief state during global localization in a locally symmetric environment. The robot has to move into one of the rooms to determine its location.

Active approaches to localization typically yield better localization results than passive ones. We already discussed an examples in the introduction to this book: coastal navigation. A second example situation is shown in Figure 7.2. Here the robot is located in a symmetric corridor, and its belief after navigating the corridor for a while is centered at two (symmetric) poses. The local symmetry of the environment makes it impossible to localize the robot while in the corridor. Only if it moves into a room will it be able to eliminate the ambiguity and to determine its pose. It is situations like these where active localization gives much better results: Instead of merely waiting until the robot incidentally moves into a room, active localization can recognize the impasse and send it there directly.

However, a key limitation of active approaches is that they require control over the robot. Thus, in practice, an active localization technique alone tends to be insufficient: The robot has to be able to localize itself even when carrying out some other task than localization. Some active localization techniques are built on top of a passive technique. Others combine tasks performance goals with localization goals when controlling a robot.

This chapter exclusively considers passive localization algorithms. We will return to the issue of active localization in Chapter **??** of this book, where we will present probabilistic algorithms for robot control.

Single-Robot Versus Multi-Robot

A fourth dimension of the localization problem is related to the number of robots involved.

- **Single-robot localization.** The most commonly studied approach to localization deals with a single robot only. Single robot localization offers the convenience that all data is collected at a single robot platform, and there is no communication issue.

- **Multi-robot localization.** The localization problem naturally arises to teams of robots. At first glance, each robot could localize itself individually, hence the multi-robot localization problem can be solved through single-robot localization. If robots are able to detect each other, however, there is the opportunity to do better. This is because one robot's belief can be used to bias another robot's belief if knowledge of the relative location of both robots is available. The issue of multi-robot localization raises interesting, non-trivial issues on the representation of beliefs and the nature of the communication between them.

These four dimensions capture the four most important characteristics of the mobile robot localization problem. There exist a number of other characterizations that impact the hardness of the problem, such as the information provided by robot measurements and the information lost through motion. Also, symmetric environments are more difficult than asymmetric ones, due to the higher degree of ambiguity. We will now look at specific algorithms and discuss their applicability to the different localization problems as defined thus far.

7.3 MARKOV LOCALIZATION

Probabilistic localization algorithms are variants of the Bayes filter. The straightforward application of Bayes filters to the localization problem is called *Markov localization*. Table 7.1 depicts the basic algorithm. This algorithm is derived from the algorithm **Bayes_filter** (Table 2.1 on page 24). Notice that **Markov_localization** also requires a map m as input. The map plays a role in the measurement model $p(z_t \mid x_t, m)$ (Line 4). It often, but not always, is incorporated in the motion model

---

1: **Algorithm Markov_localization**$(bel(x_{t-1}), u_t, z_t, m)$**:**
2:      *for all* x_t *do*
3:          $\overline{bel}(x_t) = \int p(x_t \mid u_t, x_{t-1}, m)\, bel(x_{t-1})\, dx$
4:          $bel(x_t) = \eta\, p(z_t \mid x_t, m)\, \overline{bel}(x_t)$
5:      *endfor*
6:      *return* $bel(x_t)$

Table 7.1   Markov localization.

$p(x_t \mid u_t, x_{t-1}, m)$ as well (Line 3). Just like the Bayes filter, Markov localization transforms a probabilistic belief at time $t-1$ into a belief at time t. Markov localization addresses the global localization problem, the position tracking problem, and the kidnapped robot problem in static environments.

The initial belief, $bel(x_0)$, reflects the initial knowledge of the robot's pose. It is set differently depending on the type of localization problem.

- **Position tracking.** If the initial pose is known, $bel(x_0)$ is initialized by a point-mass distribution. Let $\bar{x}_0$ denote the (known) initial pose. Then

$$bel(x_0) \quad = \quad \left\{ \begin{array}{ll} 1 & \text{if } x_0 = \bar{x}_0 \\ 0 & \text{otherwise} \end{array} \right. \tag{7.1}$$

  Point-mass distributions are discrete and therefore do not possess a density.

  In practice the initial pose is often just known in approximation. The belief $bel(x_0)$ is then usually initialized by a narrow Gaussian distribution centered around $\bar{x}_0$. Gaussians were defined in Equation (2.4) on page 11.

$$\begin{aligned} bel(x_0) \quad &= \quad \det(2\pi\Sigma)^{-\frac{1}{2}} \exp\left\{-\tfrac{1}{2}(x_0 - \bar{x}_0)^T \Sigma^{-1}(x_0 - \bar{x}_0)\right\} \\ &\sim \quad \mathcal{N}(x_0; \bar{x}_0, \Sigma) \end{aligned} \tag{7.2}$$

  Σ is the covariance of the initial pose uncertainty.

- **Global localization**. If the initial pose is unknown, $bel(x_0)$ is initialized by a uniform distribution over the space of all legal poses in the map:

$$bel(x_0) \quad = \quad \frac{1}{|X|} \tag{7.3}$$
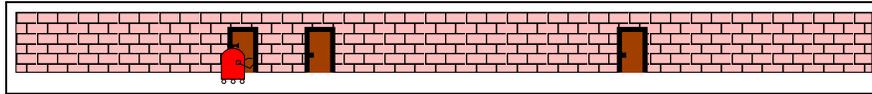
Figure 7.3 Example environment used to illustrate mobile robot localization: One-dimensional hallway environment with three indistinguishable doors. Initially the robot does not know its location except for its heading direction. Its goal is to find out where it is.

where $|X|$ stands for the volume (Lebesgue measure) of the space of all poses within the map.

- **Other.** Partial knowledge of the robot's position can usually easily be transformed into an appropriate initial distribution. For example, if the robot is known to start next to a door, one might initialize $bel(x_0)$ using a density that is zero except for places near doors, where it may be uniform. If it is known to be located in a specific corridor, one might initialize $bel(x_0)$ by a uniform distribution in the area of the corridor and zero anywhere else.

7.4   ILLUSTRATION OF MARKOV LOCALIZATION

We have already discussed Markov localization in the introduction to this book, as a motivating example for probabilistic robotics. Now we can back up this example using a concrete mathematical framework. Figure 7.3 depicts our one-dimensional hallway with three identically looking doors. The initial belief $bel(x_0)$ is uniform over all poses, as illustrated by the uniform density in Figure 7.4a. As the robot queries its sensors and notices that it is adjacent to one of the doors, it multiplies its belief $bel(x_0)$ by $p(z_t \mid x_t, m)$, as stated in Line 4 of our algorithm. The upper density in Figure 7.4b visualizes $p(z_t \mid x_t, m)$ for the hallway example. The lower density is the result of multiplying this density into the robot's uniform prior belief. Again, the resulting belief is multi-modal, reflecting the residual uncertainty of the robot at this point.

As the robot moves to the right, indicated in Figure 7.4c, Line 3 of the Markov locations algorithm convolves its belief with the motion model $p(x_t \mid u_t, x_{t-1})$. The motion model $p(x_t \mid u_t, x_{t-1})$ is not focused on a single pose but on a whole continuum of poses centered around the expected outcome of a noise-free motion. The effect is visualized in Figure 7.4c, which shows a shifted belief that is also flattened out, as a result of the convolution.
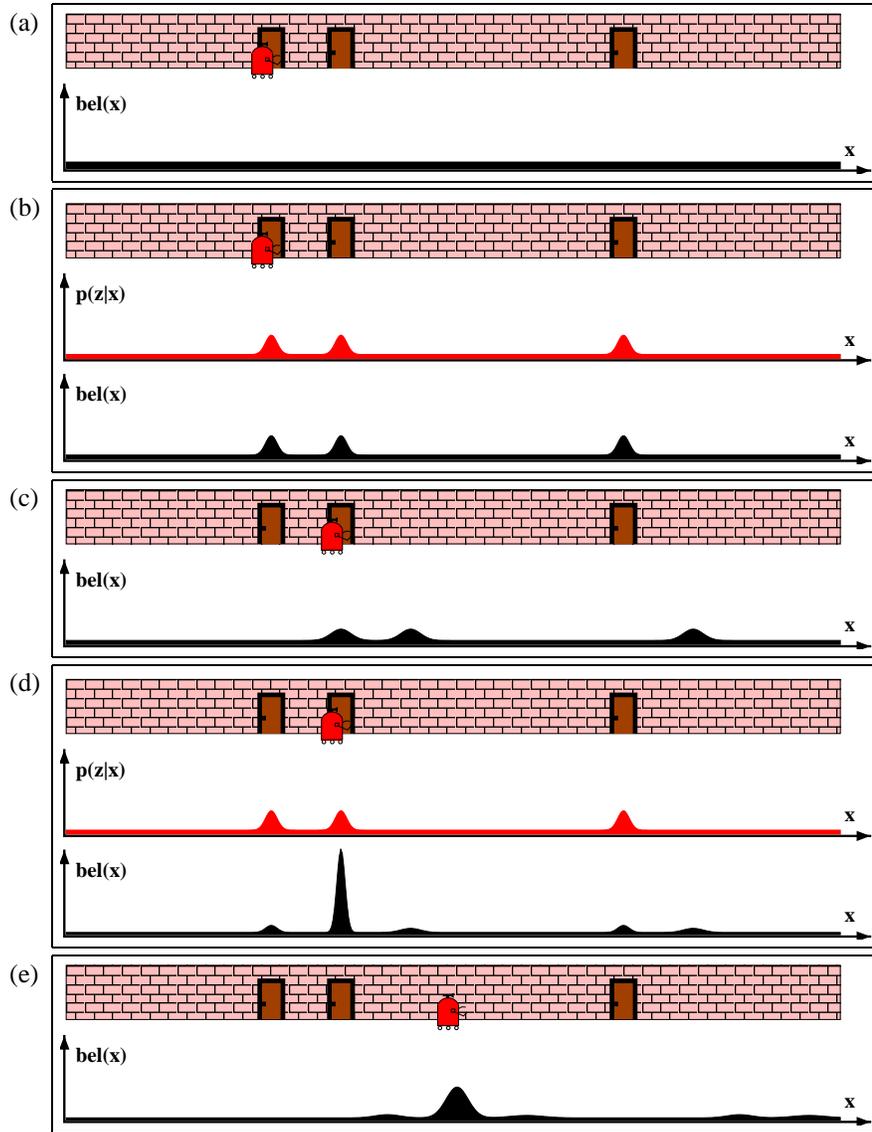
Figure 7.4 Illustration of the Markov localization algorithm. Each picture depicts the position of the robot in the hallway and its current belief $bel(x)$. (b) and (d) additionally depict the observation model $p(z_t \mid x_t)$, which describes the probability of observing a door at the different locations in the hallway.

The final measurement is illustrated in Figure 7.4d. Here the Markov localization algorithm multiplies the current belief with the perceptual probability $p(z_t \mid x_t)$. At this point, most of the probability mass is focused on the correct pose, and the robot is quite confident of having localized itself. Figure 7.4e illustrates the robot's belief after having moved further down the hallway.

We already noted that Markov localization is independent of the underlying representation of the state space. In fact, Markov localization can be implemented using any of the representations discussed in Chapter 2. We will now consider three different representations and devise practical algorithms that can localize mobile robots in real time. We begin with Kalman filters, which represent beliefs by their first and second moment. We then continue with discrete, grid representations and finally introduce algorithms using particle filters.

7.5 EKF LOCALIZATION

The extended Kalman filter localization algorithm, or EKF localization, is a special case of Markov localization. EKF localization represent beliefs $bel(x_t)$ by their their first and second moment, that is, the mean μ_t and the covariance Σ_t. The basic EKF algorithm was stated in Table 3.3 in Chapter 3.3. EKF localization shall be our first concrete implementation of an EKF in the context of an actual robotics problem.

Our EKF localization algorithm assumes that the map is represented by a collection of features. Thus, at any point in time t, the robot gets to observe a vector of ranges and bearings to nearby features: $z_t = \{z_t^1, z_t^2, \ldots\}$. We begin with a localization algorithm in which all features are uniquely identifiable. The existence of uniquely identifiable features may not be a bad assumption: For example, the Eiffel Tour in Paris is a landmark that is rarely confused with other landmarks, and it is widely visible throughout Paris. The identity of a feature is expressed by set of *correspondence variables*, denoted c_t^i, one for each feature vector z_t^i. Correspondence variables were already discussed in Chapter 6.6. In our first algorithm, the correspondence is assumed to be known. We then progress to a more general version which allows for ambiguity among features. Instead of the correspondence, the robot observes a feature signature, s_t^i. The second, more general version applied a maximum likelihood estimator to estimate the value of the latent correspondence variable, and uses the result of this estimation as ground truth.
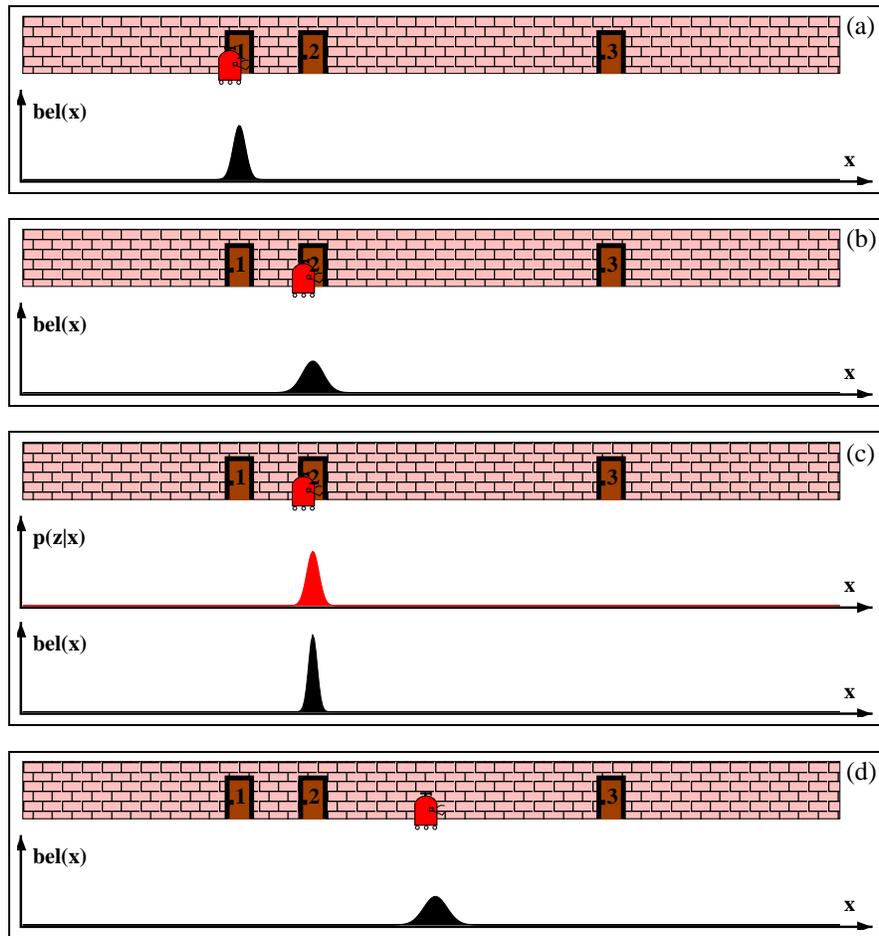
Figure 7.5 Application of the Kalman filter algorithm to mobile robot localization. All densities are represented by uni-modal Gaussians.

7.5.1 Illustration

Figure 7.5 illustrates the EKF localization algorithm using our example of mobile robot localization in the one-dimensional corridor environment (*cf.* Figure 7.3). To accommodate the unimodal shape of the belief in EKFs, we make two convenient assumptions: First, we assume that the correspondences are known: we will attach unique labels to each door (1, 2, and 3), and we will denote the measurement model by $p(z_t \mid x_t, c_t)$ where $c_t \in \{1, 2, 3\}$ is the identity of the door observed at time

t. Second, we assume that the initial pose is relatively well known. A typical initial belief is represented by the Gaussian distribution shown in Figure 7.5a, centered on the area near Door 1 and with a Gaussian uncertainty as indicated in that figure. As the robot moves to the right, its belief is convolved with the Gaussian motion model. The resulting belief is a shifted Gaussian of increased width, as shown in Figure 7.5b.

Now suppose the robot detects that it is in front of door $c_t = 2$. The upper density in Figure 7.5c visualizes $p(z_t \mid x_t, m, c_t)$ for this observation—again a Gaussian. Folding this measurement probability into the robot's belief yields the posterior shown in Figure 7.5c. Note that the variance of the resulting belief is smaller than the variances of both the robot's previous belief and the observation density. This is natural, since integrating two independent estimates should make the robot more certain than each estimate in isolation. After moving down the hallway, the robot's uncertainty in its position increases again, since the EKF continues to incorporate motion uncertainty into the robot's belief. Figure 7.5d shows one of these beliefs. This example illustrates the EKF in our limited setting.

7.5.2   The EKF Localization Algorithm

The discussion thus far has been fairly abstract: We have silently assumed the availability of an appropriate motion and measurement model, and have left unspecified a number of key variables in the EKF update. We will now discuss a concrete implementation of the EKF, for feature-based maps. Our feature-based maps consist of point landmarks, as already discussed in Chapter 6.2. For such point landmarks, we will use the common measurement model discussed in Chapter 6.6. We will also adopt the velocity motion model defined in Chapter 5.3. The reader may take a moment to briefly reacquire the basic measurement and motion equations discussed in these chapters before reading on.

Table 7.2 describes **EKF_localization_known_correspondences**, the EKF algorithm for localization with known correspondences. This algorithm is derived from the EKF in Table 3.3 in Chapter 3. It requires as its input a Gaussian estimate of the robot pose at time $t - 1$, with mean μ_{t-1} and covariance Σ_{t-1}. Further, it requires a control u_t, a map m, and a set of features $z_t = \{z_t^1, z_t^2, \ldots\}$ measured at time t, along with the correspondence variables $c_t = \{c_t^1, c_t^2, \ldots\}$. It output is a new, revised estimate μ_t, Σ_t.

The individual calculations in this algorithm will be explained further below. Lines 2 through 4 implement the familiar motion update, using a linearized motion model. The predicted pose after motion is calculated as $\bar{\mu}_t$ in Line 2, and Line 4 computes the

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

2:
$$\bar{\mu}_t = \mu_{t-1} + \begin{pmatrix} -\frac{v_t}{\omega_t}\sin\mu_{t-1,\theta} + \frac{v_t}{\omega_t}\sin(\mu_{t-1,\theta} + \omega_t\Delta t) \\ \frac{v_t}{\omega_t}\cos\mu_{t-1,\theta} - \frac{v_t}{\omega_t}\cos(\mu_{t-1,\theta} + \omega_t\Delta t) \\ \omega_t\Delta t \end{pmatrix}$$

3:
$$G_t = \begin{pmatrix} 1 & 0 & \frac{v_t}{\omega_t}\cos\mu_{t-1,\theta} - \frac{v_t}{\omega_t}\cos(\mu_{t-1,\theta} + \omega_t\Delta t) \\ 0 & 1 & \frac{v_t}{\omega_t}\sin\mu_{t-1,\theta} - \frac{v_t}{\omega_t}\sin(\mu_{t-1,\theta} + \omega_t\Delta t) \\ 0 & 0 & 1 \end{pmatrix}$$

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

5:
$$Q_t = \begin{pmatrix} \sigma_r & 0 & 0 \\ 0 & \sigma_\phi & 0 \\ 0 & 0 & \sigma_s \end{pmatrix}$$

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

7: $\quad j = c_t^i$

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

9: $\quad q = \delta^T\delta$

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

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

12: $\quad K_t^i = \bar{\Sigma}_t\,H_t^{i,T}(H_t^i\,\bar{\Sigma}_t\,H_t^{i,T} + Q_t)^{-1}$

13: *endfor*

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

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

16: *return μ_t, Σ_t*

Table 7.2 The extended Kalman filter (EKF) localization algorithm, formulated here for a feature-based map and a robot equipped with sensors for measuring range and bearing. This version assumes knowledge of the exact correspondences.
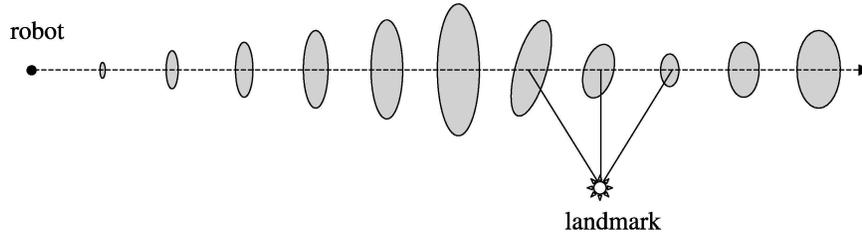
Figure 7.6 Example of localization using the extended Kalman filter. The robot moves on a straight line. As it progresses, its uncertainty increases gradually, as illustrated by the error ellipses. When it observes a landmark with known position, the uncertainty is reduced.

corresponding uncertainty ellipse. Lines 5 to 15 implement the measurement update. The core of this update is a loop through all possible features i observed at time t. In Line 7, the algorithm assigns to j the correspondence of the i-th feature in the measurement vector. It then calculates a predicted measurement $\hat{z}_t^i$ and the Jacobian H_t^i of the measurement model. The Kalman gain K_t^i is then calculated in Line 12 for each observed feature. The sum of all updates is then applied to obtain the new pose estimate, as stated in Lines 14 and 15. Notice that the last row of H_t^i is all zero. This is because the signature does not depend on the robot pose. The effect of this degeneracy is that the observed signature s_t^i has no effect on the result of the EKF update. This should come at no surprise: knowledge of the correct correspondence z_t^i renders the observed signature entirely uninformative.

Figure 7.6 illustrates the EKF localization algorithm in a synthetic environment with a single landmark. The robot starts out on the left and accrues uncertainty as it moves. Upon seeing the landmark, its uncertainty is gradually reduced as indicated.

7.5.3   Mathematical Derivation

To understand the motion update, let us briefly restate the motion model that was defined in Equation (5.13):

$$
\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} -\frac{v_t}{\omega_t}\sin\theta + \frac{v_t}{\omega_t}\sin(\theta + \omega_t\Delta t) \\ \frac{v_t}{\omega_t}\cos\theta - \frac{v_t}{\omega_t}\cos(\theta + \omega_t\Delta t) \\ \omega_t\Delta t + \gamma_t\Delta t \end{pmatrix} \tag{7.4}
$$

Here $x_t = (x \ y \ \theta)^T$ and $x_t = (x' \ y' \ \theta')^T$ are the state vectors at time $t - 1$ and t, respectively. As before, the control is a vector of two independent velocities:

$$u_t \;=\; \begin{pmatrix} v_t \\ \omega_t \end{pmatrix}, \tag{7.5}$$

where v_t is a translational velocity, and ω_t is a rotational velocity, and Δt is the time window over which the robot motion is executed. The variable $\frac{v_t}{\omega_t}$ in (7.4) stands for the quotient $\frac{v_t}{\omega_t}$.

We already know from Chapter 3 that EKF localization maintains its local posterior estimate of the state, represented by the mean μ_{t-1} and covariance Σ_{t-1}. We also recall that the "trick" of the EKF lies in linearizing the motion and measurement model. For that, we decompose the motion model into a noise-free part and a random noise component with (approximately) zero mean.

$$\underbrace{\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix}}_{x_t} \;=\; \underbrace{\begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} -\frac{v_t}{\omega_t}\sin\theta + \frac{v_t}{\omega_t}\sin(\theta + \omega_t \Delta t) \\ \frac{v_t}{\omega_t}\cos\theta - \frac{v_t}{\omega_t}\cos(\theta + \omega_t \Delta t) \\ \omega\Delta t \end{pmatrix}}_{g(u_t, x_{t-1})} + \mathcal{N}(0, R_t) \tag{7.6}$$

This is of the form (3.50) defined in Chapter 3. We recall from that chapter that EKF linearization approximates g through a Taylor expansion:

$$g(u_t, x_{t-1}) \;\approx\; g(u_t, \mu_{t-1}) \;+\; G_t \,(x_{t-1} - \mu_{t-1}) \tag{7.7}$$

The function $g(u_t, \mu_{t-1})$ is simply obtained by replacing the exact state x_{t-1}—which we do not know—by our expectation μ_{t-1}—which we know. The Jacobian G_t is the derivative of the function g with respect to μ_{t-1}, and evaluated at u_t and μ_{t-1}:

$$G_t \;=\; g'(u_t, \mu_{t-1}) \;=\; \begin{pmatrix} 1 & 0 & \frac{v_t}{\omega_t}\cos\mu_{t-1,\theta} - \frac{v_t}{\omega_t}\cos(\mu_{t-1,\theta} + \omega_t\Delta t) \\ 0 & 1 & \frac{v_t}{\omega_t}\sin\mu_{t-1,\theta} - \frac{v_t}{\omega_t}\sin(\mu_{t-1,\theta} + \omega_t\Delta t) \\ 0 & 0 & 1 \end{pmatrix} \tag{7.8}$$

Here $\mu_{t-1,\theta}$ denotes the third component of the mean vector μ_{t-1}, which is the estimate of the rotation θ_t.

The reader may have noticed that we silently ignored the fact that the amount of noise in motion Σ_u depends on the magnitude of robot motion u_t. When implementing EKF localization, the covariance R_t is a function of the velocities v_t and ω_t and the mean μ_{t-1}. This issue was already discussed in more depth in Chapter 5, and for the sake of brevity we leave the design of an appropriate Σ_u to the reader as an exercise.

EKF localization also requires a linearized measurement model with additive Gaussian noise, as discussed back in Chapter 3. The measurement model for our feature-based maps shall be a variant of Equation (6.41) in Chapter 6.6 which presupposes knowledge of the landmark identity via the correspondence variables. x Let us denote by j the identity of the landmark that corresponds to the i-th component in the measurement vector. Then we have

$$
\begin{aligned}
z_t^i &= \left(\begin{array}{c} r_t^i \\ \phi_t^i \\ s_t^i \end{array} \right) \\
&= \left(\begin{array}{c} \sqrt{(m_{j,x} - x)^2 + (m_{j,y} - y)^2} \\ \mathrm{atan2}(m_{j,y} - y, m_{j,x} - x) - \theta \\ m_{j,s} \end{array} \right) + \left(\begin{array}{c} \mathcal{N}(0, \sigma_r) \\ \mathcal{N}(0, \sigma_\phi) \\ \mathcal{N}(0, \sigma_s) \end{array} \right)
\end{aligned} \tag{7.9}
$$

Here $(m_{j,x} \;\; m_{j,y})^T$ are the coordinates of the landmark observed at time t, and $m_{j,s}$ is its (correct) signature. The variables r_t^i and ϕ_t^i denote the range and bearing to this landmark, and s_t^i denotes the signature as perceived by the robot. the variances of the measurement noise is given by σ_r, σ_ϕ, and σ_s, respectively. To bring this into a form familiar from Chapter'3, we rewrite this model as follows:

$$
z_t^i = h(x_t, j, m) + \mathcal{N}(0, Q_t) \tag{7.10}
$$

with $x_t = (x \;\; y \;\; \theta)^T$ and

$$
h(x_t, j, m) = \left(\begin{array}{c} \sqrt{(m_{j,x} - x)^2 + (m_{j,y} - y)^2} \\ \mathrm{atan2}(m_{j,y} - y, m_{j,x} - x) - \theta \\ m_{j,s} \end{array} \right) \tag{7.11}
$$

and

$$
Q_t = \left(\begin{array}{ccc} \sigma_r & 0 & 0 \\ 0 & \sigma_\phi & 0 \\ 0 & 0 & \sigma_s \end{array} \right) \tag{7.12}
$$

The Taylor approximation follows now directly from Equation (3.52):

$$h(x_t, j, m) \approx h(\bar{\mu}_t, j, m) + H_t^i \, (x_t - \bar{\mu}_t) \tag{7.13}$$

Here H_t is the Jacobian of h at $\bar{\mu}_t$, calculated for the i-the landmark and the map m. This approximation substitutes the exact robot pose x_t in $h(x_t, j, m)$ by the estimate $\bar{\mu}_t = (\bar{\mu}_{t,x} \;\; \bar{\mu}_{t,y} \;\; \bar{\mu}_{t,\theta})^T$. The Jacobian of h is given by the following matrix

$$H_t^t = h'(\bar{\mu}_t, j, m) = \begin{pmatrix} \frac{\partial r_t^i}{\partial \bar{\mu}_{t,x}} & \frac{\partial r_t^i}{\partial \bar{\mu}_{t,y}} & \frac{\partial r_t^i}{\partial \bar{\mu}_{t,\theta}} \\ \frac{\partial \phi_t^i}{\partial \bar{\mu}_{t,x}} & \frac{\partial \phi_t^i}{\partial \bar{\mu}_{t,y}} & \frac{\partial \phi_t^i}{\partial \bar{\mu}_{t,\theta}} \\ \frac{\partial s_t^i}{\partial \bar{\mu}_{t,x}} & \frac{\partial s_t^i}{\partial \bar{\mu}_{t,y}} & \frac{\partial s_t^i}{\partial \bar{\mu}_{t,\theta}} \end{pmatrix} \tag{7.14}$$

in which $\bar{\mu}_t = (\bar{\mu}_{t,x} \;\; \bar{\mu}_{t,y} \;\; \bar{\mu}_{t,\theta})^T$ denotes the estimate $\bar{\mu}_t$ factored into its individual three values. Calculating the desired derivatives from Equation (7.11) gives us the following matrix:

$$H_t^i = \begin{pmatrix} \dfrac{m_{j,x} - \bar{\mu}_{t,x}}{\sqrt{q_t}} & \dfrac{y_t - \bar{\mu}_{t,y}}{\sqrt{q_t}} & 0 \\ \dfrac{\bar{\mu}_{t,y} - y_t}{q_t} & \dfrac{m_{j,x} - \bar{\mu}_{t,x}}{q_t} & -1 \\ 0 & 0 & 0 \end{pmatrix} \tag{7.15}$$

with $q_t = (m_{j,x} - \bar{\mu}_{t,x})^2 + (m_{j,y} - \bar{\mu}_{t,y})^2$, and $j = c_t^i$ if the landmark that corresponds to the measurement z_t^i. This linear approximation, plugged into the standard EKF equations, leads to lines 7 through 11 in the EKF localization algorithm shown in Table 7.2.

Finally, we note that our feature-based localizer processes multiple measurements at-a-time, whereas the EKF discussed in Chapter 3.2 only processed a single sensor item. Our algorithm relies on an implicit conditional independence assumption, which we briefly discussed in Chapter 6.6, Equation (6.40). Essentially, we assume that all feature measurement probabilities are independent given the pose x_t and the map m:

$$p(z_t \mid x_t, m) = \prod_i p(z_t^i \mid x_t, m) \tag{7.16}$$

This is usually a good assumption, especially if the world is static. It enables us to add the information from multiple features into our filter, as specified in lines 14 and 15 in Table 7.2. This slightly non-obvious addition in Table 7.2 stems from the fact that multiple features are integrated via Bayes rule (which is a product). The addition takes place in information space, in which probabilities are represented logarithmically (c.f., Chapter 3.4). The logarithm of a product is a sum, hence the additive form of this integration step.

7.6   ESTIMATING CORRESPONDENCES

7.6.1   EKF Localization with Unknown Correspondences

The EKF localization discussed thus is only applicable when landmark correspondences can be determined with absolute certainty. In practice, this is rarely the case. Most implementations therefore determine the identity of the landmark during localization. Throughout this book, we will encounter a number of strategies to cope with the correspondence problem. The most simple of all is known as *maximum likelihood* correspondence, in which one first determines the most likely value of the correspondence variable, and then takes this value for granted.

Maximum likelihood techniques are brittle if there are many equally likely hypotheses for the correspondence variable. However, one can often design the system for this to be not the case. To reduce the danger of asserting a false data association, there exist essentially two techniques: First, select landmarks that are sufficiently unique and sufficiently far apart from each other that confusing them with each other is unlikely. Second, make sure that the robot's pose uncertainty remains small. Unfortunately, these two strategies are somewhat counter to each other, and finding the right granularity of landmarks in the environment can be somewhat of an art.

Nevertheless, the maximum likelihood technique is of great practical importance. Table 7.3 depicts the EKF localization algorithm with a maximum likelihood estimator for the correspondence. The motion update in Lines 2 through 4 is identical to the one in Table 7.2. The key difference is in the measurement update: Here we first calculate for all landmarks k in the map a number of quantities that enables us to determine the most likely correspondence (Lines 6 through 12). The correspondence variable is then chosen in Line 14, by minimizing a quadratic Mahalanobis distance function defined over the measured feature vector z_t^i and the expected measurement $\hat{z}_t^k$, for any

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

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

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

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

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

6:       *for all landmarks* k *in the map* m *do*

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

8:           $q_k = \delta_k^T \delta_k$

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

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

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

12:       *endfor*

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

14:          $j(i) = \operatorname*{argmin}_{k} (z_t^i - \hat{z}_t^k)^T \, \Psi_k^{-1} \, (z_t^i - \hat{z}_t^k)$

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

16:       *endfor*

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

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

19:       *return* μ_t, Σ_t

Table 7.3 The extended Kalman filter (EKF) localization algorithm with unknown correspondences. The correspondences $j(i)$ are estimated via a maximum likelihood estimator.

possible landmark m_k in the map. The covariance in this expression is composed of the measurement uncertainty, calculated in Line 5, and the robot uncertainty projected into the measurement space (Line 11). The final EKF update in Lines 17 and 18 only incorporates the most likely correspondences.

The algorithm in Table 7.2 is inefficient. It can be improved through a more thoughtful selection of landmarks in Lines 6 through 12. In most settings, the robot only sees a small number of landmarks at a time in its immediate vicinity; and simple tests can reject a large number of unlikely landmarks in the map.

Further, the algorithm can be modified to accommodate outliers. The standard approach is to only accept landmarks for which the Mahalanobis distance in Line 14, or the associated probability, passes a threshold test. This is generally a good idea: Gaussians fall of exponentially, and a single outlier can have a huge effect on the pose estimate. In practice, thresholding adds an important layer of robustness to the algorithm without which EKF localization tends to be brittle.

7.6.2   Mathematical Derivation

The maximum likelihood estimator determines the correspondence that maximizes the data likelihood.

$$\hat{c}_t \quad = \quad \underset{c_t}{\operatorname{argmax}}\, p(z_t \mid c_{1:t}, m, z_{1:t-1}, u_{1:t}) \tag{7.17}$$

Here c_t is the correspondence vector at time t. The vector $z_t = \{z_t^1, z_t^2, \ldots\}$ is the measurement vector which contains the list of features z_t^i observed at time t. As before, each feature vector now contains three elements, the range, the bearing, and the signature:

$$z_t^i \quad = \quad (r_t^i \ \phi_t^i \ s_t^i)^T \tag{7.18}$$

The argmax in (7.17) selects the correspondence vector $\hat{c}_t$ that maximized the likelihood of the measurement. Note that this expression is conditioned on prior correspondences $c_{1,t-1}$. While those have been estimated in previous update steps, the maximum likelihood approach treats those as if they are always correct. This has two important ramifications: It makes it possible to update the filter incrementally. But it also introduces brittleness in the filter, which tends to diverge when correspondence estimates is erroneous.

The likelihood $p(z_t \mid c_{1:t}, m, z_{1:t-1}, u_{1:t})$ in Equation (7.17) is now easily computed from the belief $\overline{bel}(x_t)$ by integrating over the pose x_t, and omitting irrelevant conditioning variables:

$$
\begin{aligned}
p(z_t \mid & c_{1:t}, m, z_{1:t-1}, u_{1:t}) \\
&= \int p(z_t \mid c_{1:t}, x_t, m, z_{1:t-1}, u_{1:t}) \, p(x_t \mid c_{1:t}, m, z_{1:t-1}, u_{1:t}) \, dx_t \\
&= \int p(z_t \mid c_t, x_t, m) \, p(x_t \mid c_{1,t-1}, m, z_{1:t-1}, u_{1:t}) \, dx_t \\
&= \int p(z_t \mid c_t, x_t, m) \, \overline{bel}(x_t) \, dx_t
\end{aligned}
\tag{7.19}
$$

This yields the following maximization

$$
\hat{c}_t = \operatorname*{argmax}_{c_t} \int p(z_t \mid c_t, x_t, m) \, \overline{bel}(x_t) \, dx_t
\tag{7.20}
$$

This maximization is carried out over the entire correspondence vector c_t.

Unfortunately, there are exponentially many terms in this maximization. When the number of features per measurement is large, the number of possible feature vectors may grow too large for practical implementations. The most common technique to avoid such an exponential complexity performs the maximization separately for each individual feature z_t^i in the measurement vector $z_t = \{z_t^1, z_t^2, \ldots\}$.

$$
\hat{c}_t^i = \operatorname*{argmax}_{c_t^i} \int p(z_t^i \mid c_t^i, x_t, m) \, \overline{bel}(x_t) \, dx_t
\tag{7.21}
$$

The implications of this component-wise optimization will be discussed below; we note that it is "justified" only when we happen to know that individual feature vectors are conditionally independent—an assumption that is usually adopted for convenience, and that we will discuss further below.

$$
p(z_t \mid c_t, x_t, m) = \prod_i p(z_t^i \mid c_t, x_t, m)
\tag{7.22}
$$

This assumption is analogous to the one stated in Equation (6.40). Under this assumption, the term that is being maximized in (7.20) becomes a product of terms with

disjoint optimization parameters, for which the maximum is attained when each individual factor is maximal.

The measurement probability for each z_t^i is given by the following exponential function:

$$p(z_t^i \mid c_t^i, x_t, m) \tag{7.23}$$
$$= \quad \eta \, \exp\left\{-\tfrac{1}{2}(z_t^i - h(x_t, c_t^i, m))^T \, Q_t^{-1} \, (z_t^i - h(x_t, c_t^i, m))\right\}$$

where h is defined as in (7.11). Applying once again our Taylor expansion as in (3.52) gives us

$$p(z_t^i \mid c_t^i, x_t, m) \quad \approx \quad \eta \, \exp\left\{-\tfrac{1}{2}(z_t^i - h(\bar{\mu}_t, c_t^i, m) \; - \; H_t \, (x_t - \bar{\mu}_t))^T \, Q_t^{-1}\right.$$
$$\left.(z_t^i - h(\bar{\mu}_t, c_t^i, m) \; - \; H_t \, (x_t - \bar{\mu}_t))\right\} \tag{7.24}$$

Thus, the integral (7.21) can be written as

$$\int \eta \, \exp\{-L_t(z_t^i, c_t^i, x_t, m)\} \, dx_t \tag{7.25}$$

with

$$L_t(z_t^i, c_t^i, x_t, m) \quad = \quad \tfrac{1}{2}(z_t^i - h(\bar{\mu}_t, c_t^i, m) \; - \; H_t \, (x_t - \bar{\mu}_t))^T \, Q_t^{-1}$$
$$(z_t^i - h(\bar{\mu}_t, c_t^i, m) \; - \; H_t \, (x_t - \bar{\mu}_t))$$
$$+ \, \tfrac{1}{2} \, (x_t - \bar{\mu}_t)^T \, \bar{\Sigma}_t^{-1} \, (x_t - \bar{\mu}_t) \tag{7.26}$$

We already encountered integrals of this form in Chapter 3.2, where we derived the motion update of the Kalman filter and the EKF. The closed-form solution to this integral is derived completely analogously to those derivations. In particular, the Gaussian defined by (7.25) has mean $h(\bar{\mu}_t, c_t^i, m)$ and covariance $H_t \, \bar{\Sigma}_t \, H_t^T + Q_t$. Thus, we have under our linear approximation the following closed form expression for the measurement likelihood:

$$\int p(z_t^i \mid c_t^i, x_t, m) \, \overline{bel}(x_t) \, dx_t \quad \sim \quad \mathcal{N}(h(\bar{\mu}_t, c_t^i, m), H_t \, \bar{\Sigma}_t \, H_t^T + Q_t) \tag{7.27}$$

and thus

$$
p(z_t^i \mid c_{1:t}, m, z_{1:t-1}, u_{1:t}) \tag{7.28}
$$
$$
= \eta \, \exp\left\{-\tfrac{1}{2} \, (z_t^i - h(\bar{\mu}_t, c_t^i, m))^T \, [H_t \, \bar{\Sigma}_t \, H_t^T + Q_t]^{-1} \, (z_t^i - h(\bar{\mu}_t, c_t^i, m))\right\}
$$

Since we only seek the value for c_t^i that maximizes this function, it suffices to maximize the quadratic term in the exponential:

$$
\hat{c}_t^i = \operatorname*{argmax}_{c_t^i} \, (z_t^i - h(\bar{\mu}_t, c_t^i, m))^T \, [H_t \, \bar{\Sigma}_t \, H_t^T + Q_t]^{-1} \, (z_t^i - h(\bar{\mu}_t, c_t^i, m))
$$
$$
\tag{7.29}
$$

This calculation is implemented in Line 15 in Table 7.3.

This distribution is remarkably similar to the probability calculated by the algorithm **landmark_model_known_correspondence** in Table 6.4, the only difference arising from the covariance term. In Table 6.4, the robot pose is assumed to be known, hence the covariance reduces to the measurement covariance Q_t. Here we only have a probabilistic estimate of the pose, hence have to use our best estimate for the pose, and fold in the uncertainty in our estimate. Our state estimate is given by $\bar{\mu}_t$. As the derivation above shows, the covariance is adjusted by the additional uncertainty $H_t \, \bar{\Sigma}_t \, H_t^T$, which is the projection of the pose uncertainty under the linearized approximation of the measurement function h. This shows the correctness of the calculation in lines 12 and 13 in our EKF algorithm in Table 7.3: π_k is indeed the desired likelihood. The correctness of the algorithm follows from our assumption that feature measurements are conditionally independent, as stated in Equation (7.22). Of course, this independence is usually violated, which makes our algorithm approximate. A further approximation is introduced by the Taylor expansion.

7.7 MULTI-HYPOTHESIS TRACKING

There exist a number of extensions of the basic EKF to accommodate situations where the correct data association cannot be determined with sufficient reliability. Several of those techniques will be discussed later in this book, hence our exposition at this point will be brief.

A classical technique that overcomes difficulties in data association is the *Multi-hypothesis Tracking Algorithm (MHT)*. The MHT can represent a belief by multiple

Gaussians, that is, the posterior is represented by the mixture

$$bel(x_t) \quad = \quad \frac{1}{\sum_l \psi_{t,l}} \sum_l \psi_{t,l} \det \left(2\pi\Sigma_{t,l}\right)^{-\frac{1}{2}} \exp\left\{-\tfrac{1}{2}(x_t - \mu_{t,l})^T \Sigma_{t,l}^{-1}(x_t - \mu_{t,l})\right\}$$

(7.30)

Here l is the index of the mixture component. Each such component, or "track" in MHT slang, is itself a Gaussian with mean $\mu_{t,l}$ and covariance $\Sigma_{t,l}$. The scalar $\psi_{t,l} \geq 0$ is a *mixture weight*. It determines the weight of the l-th mixture component in the posterior. Since the posterior is normalized by $\sum_l \psi_{t,l}$, each $\psi_{t,l}$ is a relative weight, and the contribution of the l-th mixture component depends on the magnitude of all other mixture weights.

As we shall see below when we describe the MHT algorithm, each mixture component relies on a unique sequence of data association decisions. Hence, it makes sense to write $c_{t,l}$ for the data association vector associated with the l-th track, and $c_{1:t,l}$ for all past and present data associations associated with the l-th mixture component. With this notation, we can now think of mixture components as contributing local belief functions conditioned on a unique sequence of data associations:

$$bel_l(x_t) \quad = \quad p(x_t \mid z_{1:t}, u_{1:t}, c_{1:t,l})$$

(7.31)

Here $c_{1:t,l} = \{c_{1,l}, c_{2,l}, \ldots, c_{t,l}\}$ denotes the sequence of correspondence vectors associated with the l-th track.

Before describing the MHT, it makes sense to discuss a completely intractable algorithm from which the MHT is derived. This algorithm is the full Bayesian implementation of the EKF under unknown data association. It is amazingly simple: Instead of selecting the most likely data association vector, our fictitious algorithm maintains them all. More specifically, at time t each mixture is split into many new mixtures, each conditioned on a unique correspondence vector c_t. Let m be the index of one of the new Gaussians, and l be the index from which this new Gaussian is derived, for the correspondence $c_{t,l}$. The weight of this new mixture is then set to

$$\psi_{t,m} \quad = \quad \psi_{t,l} \, p(z_t \mid c_{1:t-1,l}, c_{t,m}, z_{1:t-1}, u_{1:t})$$

(7.32)

This is the product of the mixture weight $\psi_{t,l}$ from which the new component was derived, times the likelihood of the measurement z_t under the specific correspondence

vector that led to the new mixture component. In other words, we treat correspondences as latent variable and calculate the posterior likelihood that a mixture component is correct. A nice aspect of this approach is that we already know how to compute the measurement likelihood $p(z_t \mid c_{1:t-1,l}, c_{t,m}, z_{1:t-1}, u_{1:t})$ in Equation (7.32): It is simply the product $\prod \pi_k$ of the individual feature likelihoods computed in line 13 of our localization algorithm in Table 7.3. Thus, we can incrementally calculate the mixture weights for each new component. The only downside of this algorithm is the fact that the number of mixture components, or tracks, grow exponentially over time.

The MHT algorithm approximates this algorithm by keeping the number of mixture components small. In essence, it terminates every component whose relative mixture weight

$$\frac{\psi_{t,l}}{\sum_m \psi_{t,m}} \tag{7.33}$$

is smaller than a threshold $\psi_{\min}$. It is easy to see that the number of mixture components is always at most $\psi_{\min}^{-1}$. Thus, the MHT maintains a compact posterior that can be updated efficiently. It is approximate in that it maintains a very small number of Gaussians, but in practice the number of plausible robot locations is usually very small.

We will omit a formal description of the MHT algorithm at this point, and instead refer the reader to a large number of related algorithms in this book. However, when implementing the MHT, it is useful to devise strategies for identifying low-likelihood tracks before instantiating them.

7.8 PRACTICAL CONSIDERATIONS

The EKF localization algorithm and its close relative, MHT localization, are popular techniques for position tracking. There exist a large number of variations of these algorithm that enhance their efficiency and robustness.

- **Efficient search.** First, it is often impractical to loop through all landmarks k in the map. Often, there exist simple tests to identify plausible candidate landmarks (e.g., by simply projecting the measurement into x-y-space), enabling one to rule out all but a constant number of candidates. Such algorithms can be orders of magnitude faster that our naive implementations.

- **Mutual exclusion.** A key limitation of our implementations arises from our assumed independence of feature noise in the EKF (and, by inheritance, the MHT). The reader may recall condition (7.22), which enabled us to process individual features sequentially, thereby avoiding a potential exponential search through the space of all correspondence vectors. Unfortunately, such an approach allows for assigning multiple observed features, say z_t^i and z_t^j with $i \neq j$, to be assigned to the same landmark in the map: $\hat{c}_t^i = \hat{c}_t^i$. For many sensors, such a correspondence assignment is wrong by default. For example, if the feature vector is extracted from a single camera image, we know by default that two different regions in the image space must correspond to different locations in the physical world. Put differently, we usually know that $i \neq j \longrightarrow \hat{c}_t^i \neq \hat{c}_t^i$. This (hard!) constraint is called *mutual exclusion principle in data association*. It reduces the space of all possible correspondence vectors. Advanced implementations consider this constraint. For example, one might first search for each correspondence separately—as in our version of the EKF localizer—followed by a "repair" phase in which violations of the mutual exclusion principle are resolved by changing correspondence values accordingly.

- **Outliers.** Further, our implementation does not address the issue of outliers. The reader may recall from Chapter 6.6 that we allow for a correspondence $c = N + 1$, with N being the number of landmarks in the map. Such an outlier test is quite easily added to our algorithms. In particular, if we set π_{N+1} to be the a prior probability of an outlier, the argmax-step in line 15 EKF localization (Table 7.3) will default to $N + 1$ if an outlier is the most likely explanation of the measurement vector. Clearly, an outlier does not provide any information on the robot's pose; hence, the corresponding terms are simply omitted in lines 18 and 19 in Table 7.3.

Both the EKF and the MHT localization are only applicable to position tracking problems. In particular, linearized Gaussian techniques tend to work well only if the position uncertainty is small. There are three complimentary reasons for this observation:

- A uni-modal Gaussian is usually a good representation of uncertainty in tracking whereas it is not in more general global localization problems. Both the EKF and the MHT start with a single unimodal Gaussian, although the MHT can potentially branch into multiple local Gaussian.

- A narrow Gaussian reduces the danger of erroneous correspondence decisions. This is important particularly for the EKF, since a single false correspondence can derail the tracker by inducing an entire stream localization and correspondence errors. The MHT is more robust to this problem, though it can fail equally

when the correct correspondence is not among those maintained in the Gaussian mixture.

- The Taylor expansion is usually only good in a close proximity to the linearization point. As a rule of thumb, if the standard deviation for the orientation θ is larger then ± 20 degrees, linearization effects are likely to make the algorithm fail. This problem applies equally to the EKF and the MHT and explains why starting the MHT with a very wide initial Gaussian does not turn it into a global localization algorithm.

For all those reasons, the Gaussian localization algorithms discussed in this chapter are inapplicable to global localization problems or the kidnapped robot problem.

The design of the appropriate features for EKF localization is a bit of an art. This is because multiple competing objectives have to be met. On the one hand, one wants sufficiently many features in the environment, so that the uncertainty in the robot's pose estimate can be kept small. Small uncertainty is absolutely vital for reasons already discussed. On the other hand, one wants to minimize chances that landmarks are confused with each other, or that the landmark detector detects spurious features. Many environments do not possess too many point landmarks that can be detected with high reliability, hence many implementation rely on relatively sparsely distributed landmarks. Here the MHT has a clear advantage, in that it is more robust to data association errors. As a rule of thumb, large numbers of landmarks tend to work better than small numbers even for the EKF. When landmarks are dense, however, it is critical to apply the mutual exclusion principle in data association.

Finally, we note that EKF localization processes only a subset of all information in the sensor measurement. By going from raw measurements to features, the amount of information that is being processed is already drastically reduced. Further, EKF localization is unable to process *negative* information. Negative information pertains to the absence of a feature. Clearly, not seeing a feature when one expects to see it carries relevant information. For example, not seeing the Eiffel Tour in Paris implies that it is unlikely that we are right next to it. The problem with negative information is that it induces non-Gaussian beliefs, which cannot be represented by the mean and variance. For this reason, EKF implementations simply ignore the issue of negative information, and instead integrate only information from observed features. The standard MHT also avoids negative information. However, it is possible to fold negative information into the mixture weight, by decaying mixture components that failed to observe a landmark.

With all these limitations, does this mean that Gaussian techniques are generally brittle and inapplicable to more general localization techniques? The answer is no. In fact,

the key to successful localization lies in the approach for data association. Later in this book, we will encounter more sophisticated techniques for handling correspondences than the ones discussed thus far. Many of these techniques are applicable (and will be applied!) to Gaussian representations, and the resulting algorithms are often among the best ones known.

7.9   SUMMARY

In this chapter, we introduced the mobile robot localization problem and devised a first practical algorithm for solving it.

- The localization problem is the problem of estimating a robot's pose relative to a known map of its environment.

- Position tracking addresses the problem of accommodating the local uncertainty of a robot whose initial pose is known; global localization is the more general problem of localizing a robot from scratch. Kidnapping is a localization problem in which a well-localized robot is secretly teleported somewhere else without being told—it is the hardest of the three localization problems.

- The hardness of the localization problem is also a function of the degree to which the environment changes over time. All algorithms discussed thus far assume a static environment.

- Passive localization approaches are filters: they process data acquired by the robot but do not control the robot. Active techniques control the robot. In this and the next chapter, we study passive approaches; active approaches will be discussed in Chapter **??** of this book.

- Markov localization is just a different name for the Bayes filter applied to the mobile robot localization problem.

- EKF localization applies the extended Kalman filter to the localization problem. EKF localization is primarily applied to feature-based maps.

- The most common technique for dealing with correspondence problems is the maximum likelihood technique. This approach simply assumes that at each point in time, the most likely correspondence is correct.

- The multi hypothesis tracking algorithm (MHT) pursues multiple correspondences, using a Gaussian mixture to represent the posterior. mixture components

are created dynamically, and terminated if their total likelihood sinks below a user-specified threshold.

- The MHT is more robust to data association problems than the EKF, at an increased computational cost.

- Both EKF and MHT localization are well-suited for local position tracking problems with limited uncertainty and in environments with distinct features. They are less applicable to global localization or in environments where most objects look alike.

- Selecting features for EKFs and MHTs requires skill! The performance of both algorithms can be improved by a number of measures, such as enforcing mutual exclusion in data association.

In the next chapter, we will discuss probabilistic localization techniques that can cope with more general localization problems, such as the global localization problem or the problem of localizing in dynamic environments.

8

GRID AND MONTE CARLO LOCALIZATION

8.1 INTRODUCTION

This chapter describes two localization algorithms that are capable of solving global localization problems. These algorithms possess a number of differences to the Gaussian techniques discussed in the previous chapter.

- They can process raw sensor measurements. There is no need to extract features from sensor values. As a direct implication, they can also process negative information.

- They are non-parametric. In particular, they are not bound to a uni-modal distribution as was the case with the EKF localizer.

- They can solve global localization and—in some instances—kidnapped robot problems. Neither the EKF nor the MHT are able to solve such problems—although the MHT can be modified so as to solve global localization problems.

The techniques presented here have exhibited excellent performance in a number of fielded robotic systems.

The first approach is called *grid localization*. It uses a histogram filter to represent the posterior belief. A number of issues arise when implementing grid localization: with a fine-grained grid, the computation required for a naive implementation may make the algorithm unreasonably slow. With a coarse grid, the additional information loss through the discretization negatively affects the filter and—if not properly treated—may even prevent the filter from working.

1:          **Algorithm Grid_localization($\{p_{k,t-1}\}, u_t, z_t, m$):**
2:               for all k do
3:                    $\bar{p}_{k,t} = \sum_{i} p_{i,t-1}$ **motion_model**$(\text{mean}(\mathbf{x}_k), u_t, \text{mean}(\mathbf{x}_i))$
4:                    $p_{k,t} = \eta$ **measurement_model**$(z_t, \text{mean}(\mathbf{x}_k), m)$
5:               endfor
6:               return $\{p_{k,t}\}$

Table 8.1 Grid localization, a variant of the discrete Bayes filter. The function **motion_model** implements one of the motion models, and **measurement_model** a sensor model. The function 'mean' returns the center of gravity of a grid cell $\mathbf{x}_k$.

The second approach is the Monte Carlo localization (MCL) algorithm, arguably the most popular approach to date. It uses particle filters to estimate posteriors over robot poses. A number of shortcomings of the MCL are discussed, and techniques for applying it to the kidnapped robot problem and to dynamic environments are presented.

The material in this chapter covers some of the most successful methods to date. Appropriately implemented, these techniques are able to localize robots globally, and to recover from localization failure. These abilities make the algorithms presented here the method of choice in many applications that require reliable robot operation.

8.2   GRID LOCALIZATION

8.2.1   Basic Algorithm

Grid localization approximates the posterior using a histogram filter over a grid decomposition of the pose space. The discrete Bayes filter was already extensively discussed in Chapter 4.1 and is depicted in Table 4.1. It maintains as posterior a collection of discrete probability values

$$bel(x_t) \quad = \quad \{p_{k,t}\} \tag{8.1}$$

where each probability $p_{k,t}$ is defined over a grid cell $\mathbf{x}_k$. The set of all grid cells forms a partition of the space of all legitimate poses:

$$\mathrm{range}(X_t) \quad = \quad \mathbf{x}_{1,t} \cup \mathbf{x}_{2,t} \cup \ldots \mathbf{x}_{K,t} \tag{8.2}$$

In the most basic version of grid localization, the partitioning of the space of all poses is time-invariant, and each grid cell is of the same size. A common granularity used in many of the indoor environments is 15 centimeters for the x- and y-dimensions, and 5 degrees for the rotational dimension. A finer representation yields better results, but at the expense of increased computational requirements.

Grid localization is largely identical to the basic binary Bayes filter from which it is derived. Table 8.1 provides pseudo-code for the most basic implementation. It requires as input the discrete probability values $\{p_{t-1,k}\}$, along with the most recent measurement, control, and the map. Its inner loop iterates through all grid cells. Line 3 implements the motion model update, and line 4 the measurement update. The final probabilities are normalized, as indicated by the normalizer η in line 4. The functions **motion_model**, and **measurement_model**, may be implemented by any of the motion models in Chapter 5, and measurement models in Chapter 6, respectively. The algorithm in Table 8.1 assumes that each cell possesses the same volume.

Figure 8.1 illustrates grid localization in our one-dimensional hallway example. This diagram is equivalent to that of the general Bayes filter, except for the discrete nature of the representation. As before, the robot starts out with global uncertainty, represented by a uniform histogram. As it senses, the corresponding grid cells raise its probability value. The example highlights the ability to represent multi-modal distributions with grid localization.

8.2.2 Grid Resolutions

A key variable of the grid localizer is the resolution of the grid. On the surface, this might appear to be a minor detail; however, the type sensor model that is applicable, the computation involved in updating the belief, and the type results to expect all depend on the grid resolution.

At the extreme end are two types of representations, both of which have been brought to bear successfully in fielded robotics systems.
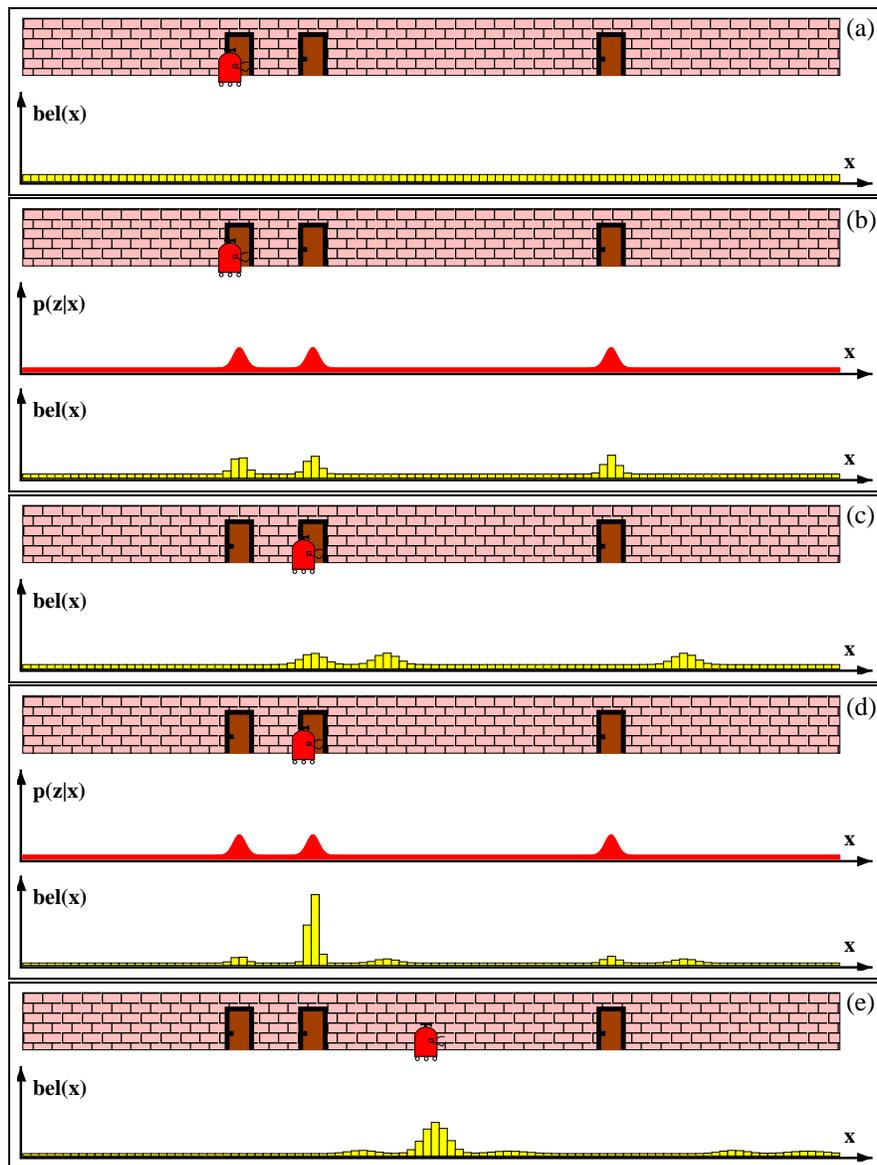
Figure 8.1 Grid localization using a fine-grained metric decomposition. Each picture depicts the position of the robot in the hallway along with its belief $bel(x_t)$, represented by a histogram over a grid.
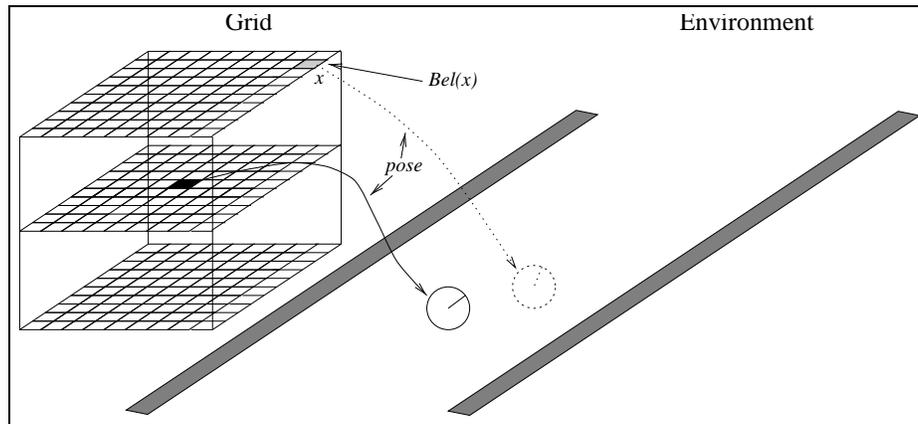
Figure 8.2 Example of a fixed-resolution grid over the robot pose variables x, y, and θ. Each grid cell represents a robot pose in the environment. Different orientations of the robot correspond to different planes in the grid (shown are only three orientations).

- **Coarse, variable-resolution grids.** Some implementation decompose the space of all poses into regions that correspond to "significant" places in the environment. Such places may be defined by the presence (or absence) of specific landmarks, such as doors and windows. In hallway environments, places may correspond to intersections, T-junctions, dead ends, and so on. In such representations, the resolution of the decomposition depends on the structure of the environment, and they tend to be course. Figure 8.5 shows such a coarse representation for the one-dimensional hallway example. Course representation like these are commonly associated with topological representations of space.

- **Fine fixed-resolution grids.** Other methods decompose the state space using equally spaced grids. The resolution of such decompositions is usually much higher than that of variable-resolution grids. For example, some of the examples in Chapter 7 use grid decompositions with cell sizes of 15 centimeters or less. Hence, they are more accurate, but at the expense of increased computational costs. Figure 8.2 illustrates such a fixed-resolution grid. Fine resolution like these are commonly associated with metric representation of space.

When implementing grid localization for coarse resolutions, it is important to compensate for the coarseness in the resolution in the sensor and motion models. In particular, for a high-resolution sensor like a laser range finder, the value of the measurement model $p(z_t \mid x_t)$ may vary drastically inside each grid cell $\mathbf{x}_{k,t}$. If this is the case, just evaluating it at the center-of-gravity will generally yield a poor result. Similarly, pre-
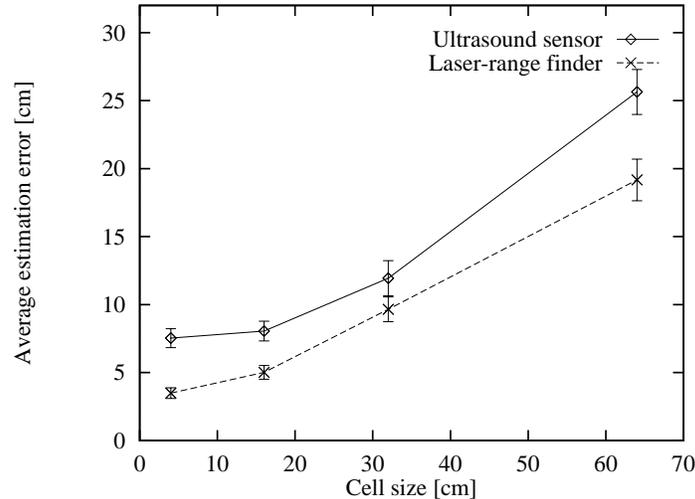
Figure 8.3 Average localization error as a function of grid cell size, for ultrasound sensors and laser range-finders.

dicting robot motion from the center-or-gravity may yield poor results: If the motion is updated in 1-second intervals for a robot moving at 10cm/sec, and the grid resolution is 1 meter, the naive implementation will never result in a state transition. This is because any location that is approximately 10cm away form the center-or-gravity of a grid cell still falls into the same grid cell.

A common way to compensate this effect is to modify both the measurement and the motion model by inflating the amount of noise. For example, the variance of a range finder model's main Gaussian cone may be enlarged by half the diameter of the grid cell. In doing so, the new model is much smoother, and its interpretation will be less susceptible to the exact location of the sample point relative to the correct robot location. However, this modified measurement model reduces the information intake, thereby reducing the localization accuracy. Similarly, a motion model may predict a random transition to a nearby cell with a probability that it proportional to the length of the motion arc, divided by the diameter of a cell. The result of such an inflated motion model is that the robot can indeed move from one cell to another, even if its motion between consecutive updates is small relative to the size of a grid cell. However, the resulting posteriors are wrong in that an unreasonably large probability will be placed on the hypothesis that the robot changes cell at each motion update—and hence moves much much faster than commanded.
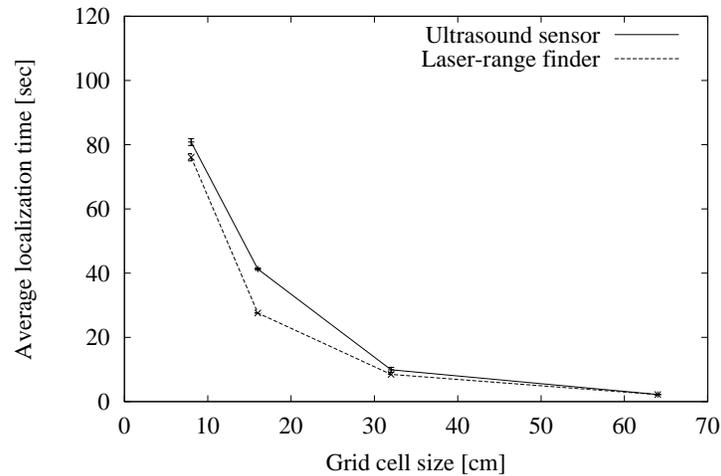
Figure 8.4 Average CPU-time needed for global localization as a function of grid resolution, shown for both ultrasound sensors and laser range-finders.

Figures 8.3 and 8.4 plot the performance of grid localization as a function of the resolution, for two different types of range sensors. As to be expected, the localization error increases as the resolution decreases. The total time necessary to localize a robot decreases as the grid becomes coarser, as shown in Figure 8.4.

8.2.3 Computational Considerations

When using a fine-grained grid such as some of the metric grids described in the previous section, the basic algorithm cannot be executed in real-time. At fault are both the motion and the measurement update. The motion update requires a convolution, which for a 3-D grid is a 6-D operation. The measurement update is a 3-D operation, but calculating the likelihood of a full scan is a costly operation.

There exist a number of techniques to reduce the computational complexity of grid localization.

■ **Pre-caching.** The measurement model is costly since it may involve ray casting. As motivated in Chapter 6.3.4, a common strategy is to calculate for each grid cell essential statistics that facilitate the measurement update. In particular,
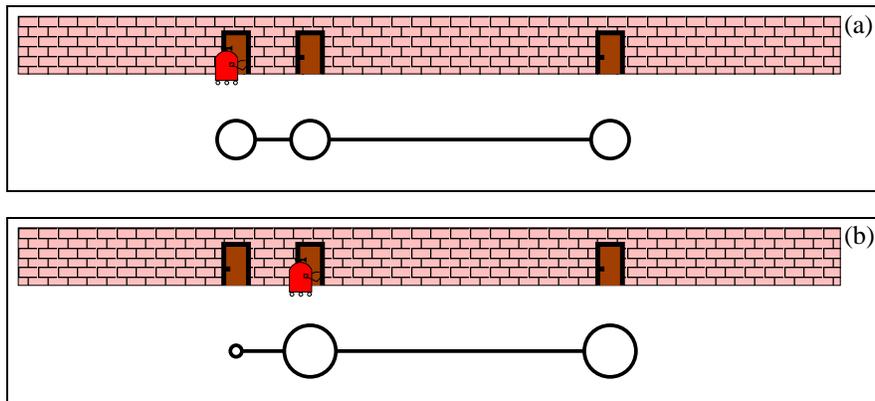
Figure 8.5 Application of a coarse-grained, topological representation to mobile robot localization. Each state corresponds to a distinctive place in the environment (a door in this case). The robot's belief $bel(x_t)$ of being in a state is represented by the size of the circles. (a) The initial belief is uniform over all poses. (b)b shows the belief after the robot made one state transition and detected a door. At this point, it is unlikely that the robot is still in the left position.

when using a beam model, it is common to cache away the correct range for each center-of-gravity point in each grid cell. Further, the sensor model can be precalculated for a fine-grained array of possible ranges. The calculation of the measurement model reduces then to two table lookups, which is much faster.

- **Sensor subsampling.** Further speed-ups can be achieved by evaluating the measurement model only for a subset of all ranges. In some of our systems, we use only 8 of our 360 laser range measurement and still achieve excellent results. Subsampling can take place spatially and in time.

- **Delayed motion updates.** Instead of applying the motion update every time a control is issued or an odometry reading is obtained, it is possible to reduce the frequency of motion update. This is achieved by geometrically integrating the controls or odometry readings over short time period. This can speed up the algorithm by an order of magnitude.

- **Selective updating**. This technique was already described in Chapter 4.1.3. When updating the grid, selective techniques update a fraction of all grid cells only. A common implementation of this idea updates only those grid cells whose posterior probability exceeds a user-specified threshold. Selective updating techniques can reduce the computational effort involved in updating beliefs by many orders of magnitude. Special care has to be taken to reactivate low-likelihood grid cells when one seeks to apply this approach to the kidnapped robot problem.

With these modifications, grid localization can in fact become quite efficient; Even 10 years ago, low-end PC were fast enough to generate the results shown in this chapter. However, our modifications place an additional burden on the programmer, and make a final implementation more complex than the short algorithm in Table 8.1 suggests.

8.2.4   Illustration

Figure 8.6 shows an example of Markov localization with metric grids, at a spatial resolution of 15 centimeter and an angular resolution of 5 degrees. Shown there is a global localization run where a mobile robot equipped with two laser range-finders localizes itself from scratch. The probabilistic model of the range-finders is computed by the beam model described in Section 6.3 and depicted in Table 8.1.

Initially, the robot's belief is uniformly distributed over the pose space. Figure 8.6a depicts a scan of the laser range-finders taken at the start position of the robot. Here, max range measurements are omitted and the relevant part of the map is shaded in grey. After incorporating this sensor scan, the robot's location is focused on just a few regions in the (highly asymmetric) space, as shown by the gray-scale in Figure 8.6b. Notice that beliefs are projected into x-y space; the true belief is defined over a third dimension, the robot's orientation θ, which is omitted in this and the following diagrams. Figure 8.6d shows the belief after the robot moved 2m, and incorporated the second range scan shown in Figure 8.6c. The certainty in the position estimation increases and the global maximum of the belief already corresponds to the true location of the robot. After integrating another scan into the belief the robot finally perceives the sensor scan shown in Figure 8.6e. Virtually all probability mass is now centered at the actual robot pose (see Figure 8.6f). Intuitively, we say that the robot successfully localized itself. This example illustrates that grid localization is capable to globally localize a robot very efficiently. A second example is shown in Figure 8.7.

Of course, global localization usually requires more than just a few sensor scans to succeed. This is particularly the case in symmetric environments, and if the sensors are less accurate than laser sensors. Figures 8.8 to 8.10 illustrate global localization using a mobile robot equipped with sonar sensors only, and in an environment that possesses many corridors of approximately the same width. An occupancy grid map is shown in Figure 8.8. Figure 8.9a shows the data set, obtained by moving along one of the corridors and then turning into another. Each of the "beams" in Figure 8.9a corresponds to a sonar measurement. In this particular environment, the walls are smooth and a large fraction of sonar readings are corrupted. Again, the probabilistic model of the sensor readings is the beam-based model described in described in Section 6.3. Figure 8.9 additionally shows the belief for three different points in time, marked "A,"
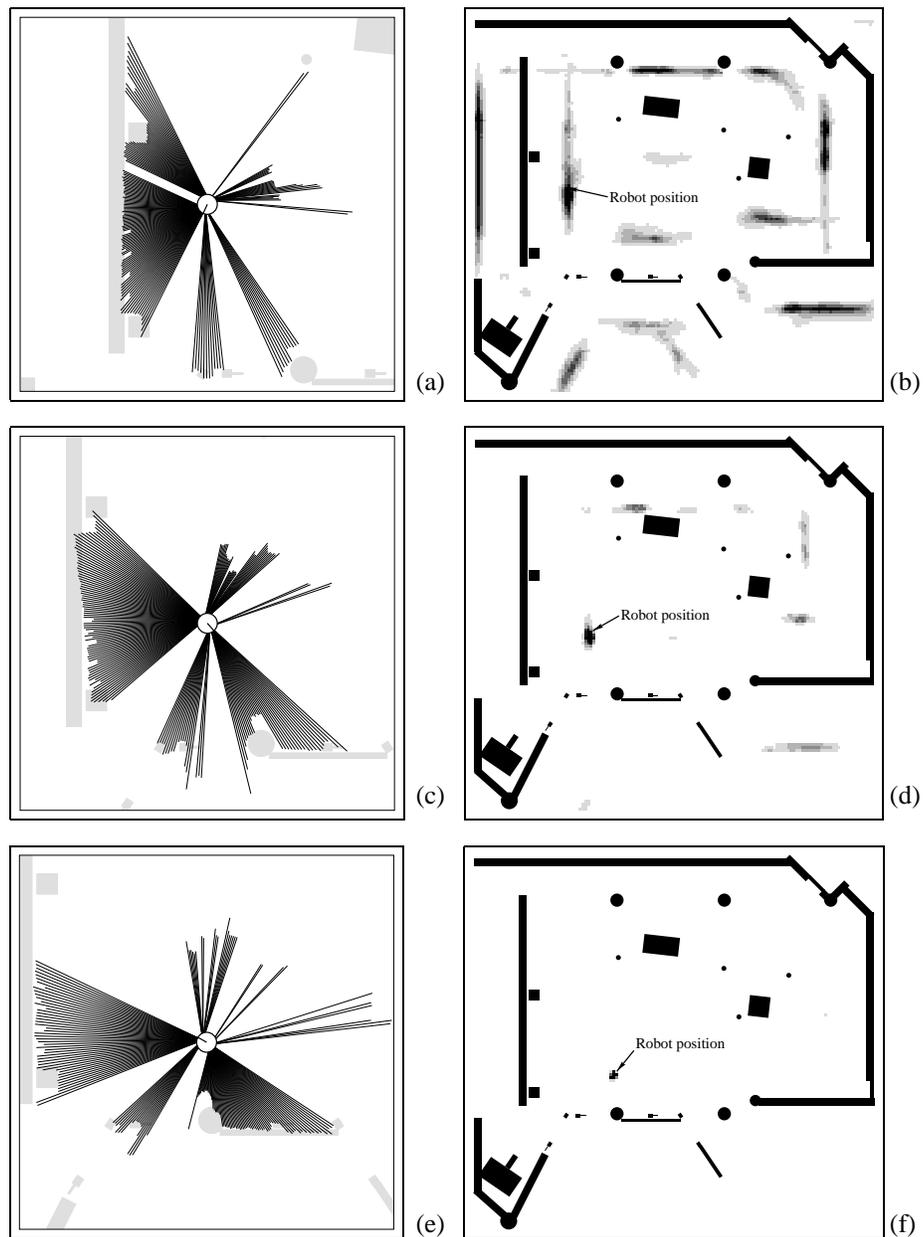
Figure 8.6 Global localization in a map using laser range-finder data. (a) Scan of the laser range-finders taken at the start position of the robot (max range readings are omitted). Figure (b) shows the situation after incorporating this laser scan, starting with the uniform distribution. (c) Second scan and (d) resulting belief. After integrating the final scan shown in (e), the robot's belief is centered at its actual location (see (f)).
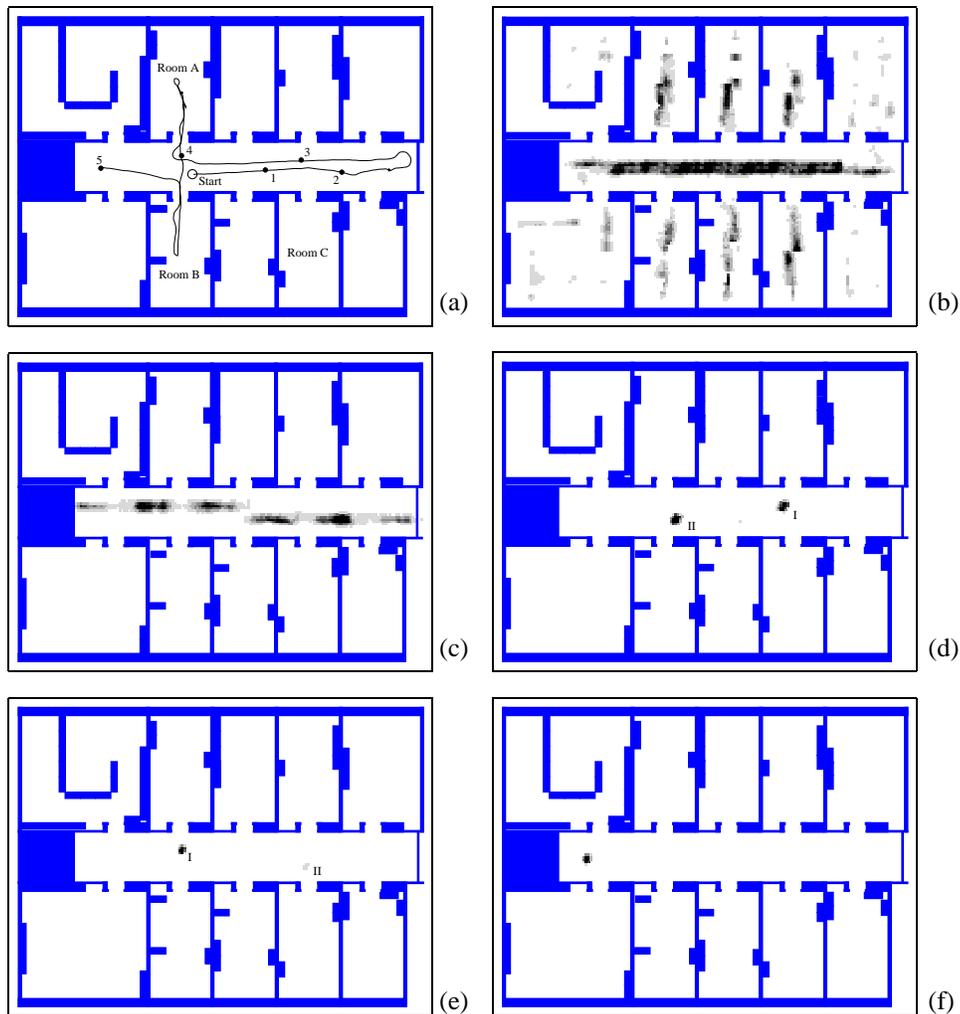
Figure 8.7 Global localization in an office environment using sonar data. (a) Path of the robot. (b) Belief as the robot passes position 1. (c) After some meters of robot motion, the robot knows that it is in the corridor. (d) As the robot reaches position 3 it has scanned the end of the corridor with its sonar sensors and hence the distribution is concentrated on two local maxima. While the maximum labeled I represents the true location of the robot, the second maximum arises due to the symmetry of the corridor (position II is rotated by 180° relative to position I). (e) After moving through Room A, the probability of being at the correct position I is now higher than the probability of being at position II. (f) Finally the robot's belief is centered on the correct pose.
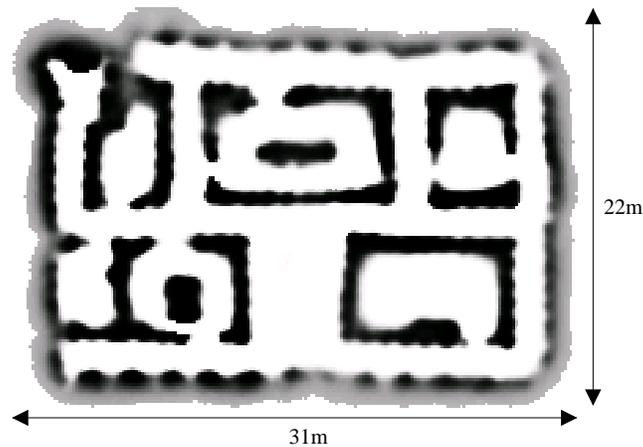
Figure 8.8   Occupancy grid map of the 1994 AAAI mobile robot competition arena.

"B," and "C" in Figure 8.9a. After moving approximately three meters, during which
the robot incorporates 5 sonar scans, the belief is spread almost uniformly along all
corridors of approximately equal size, as shown in Figure 8.9b. A few seconds later,
the belief is now focused on a few distinct hypotheses, as depicted in Figure 8.9c.
Finally, as the robot turns around the corner and reaches the point marked "C," the
sensor data is now sufficient to uniquely determine the robot's position. The belief
shown in Figure 8.9d is now closely centered around the actual robot pose. This ex-
ample illustrates that the grid representation works well for high-noise sonar data and
in symmetric environments, where multiple hypotheses have to be maintained during
global localization.

Figure 8.10 illustrates the ability of the grid approach to correct accumulated dead-
reckoning errors by matching sonar data with occupancy grid maps. Figure 8.10a
shows the raw odometry data of a 240m long trajectory. Obviously, the rotational
error of the odometry quickly increases. After traveling only 40m, the accumulated
error in the orientation (raw odometry) is about 50 degrees. Figure 8.10b shows the
path of the robot estimated by the grid localizer.

Obviously, the resolution of the discrete representation is a key parameter for grid
Markov localization. Given sufficient computing and memory resources, fine-grained
approaches are generally preferable over coarse-grained ones. In particular, fine-
grained approaches are superior to coarse-grained approaches, assuming that sufficient
computing time and memory is available. As we already discussed in Chapter 2.4.4,
the histogram representation causes systematic error that may violate the Markov as-
sumption in Bayes filters. The finer the resolution, the less error is introduced, and the
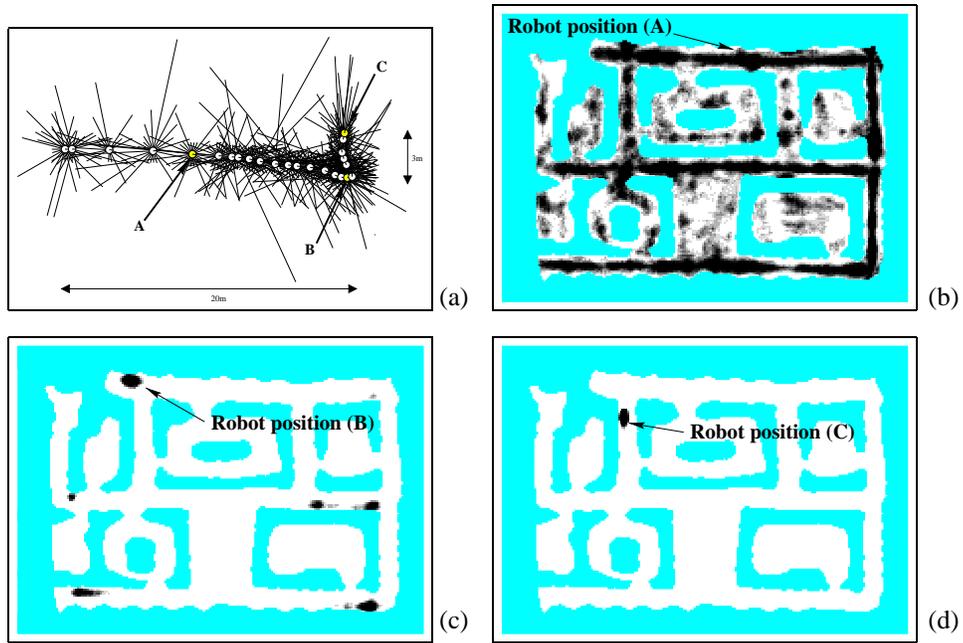
Figure 8.9 (a) Data set (odometry and sonar range scans) collected in the environment shown in Figure 8.8. This data set is sufficient for global localization using the grid localization. The beliefs at the points marked "A," "B" and "C" are shown in (b) - (d).
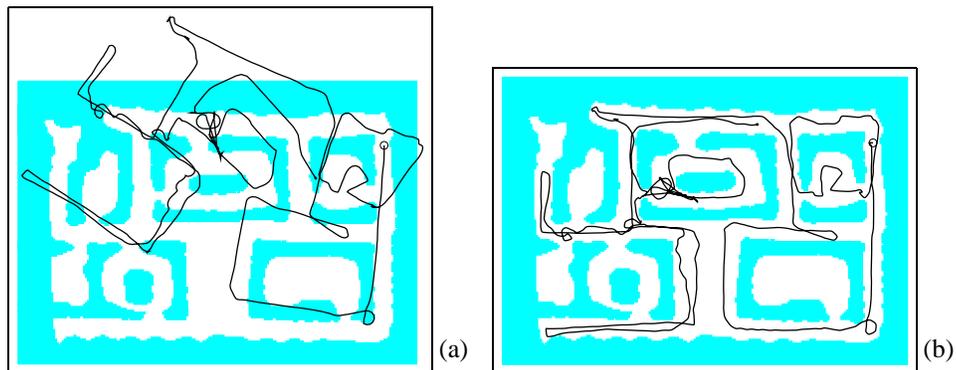
Figure 8.10 (a) Odometry information and (b) corrected path of the robot.

```
1:        Algorithm MCL($\mathcal{X}_{t-1}, u_t, z_t, m$):
2:            $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
3:            for $m = 1$ to $M$ do
4:                $x_t^{[m]} = \mathbf{sample\_motion\_model}(u_t, x_{t-1}^{[m]})$
5:                $w_t^{[m]} = \mathbf{measurement\_model}(z_t, x_t^{[m]}, m)$
6:                $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
7:            endfor
8:            for $m = 1$ to $M$ do
9:                draw $i$ with probability $\propto w_t^{[i]}$
10:               add $x_t^{[i]}$ to $\mathcal{X}_t$
11:           endfor
12:           return $\mathcal{X}_t$
```

Table 8.2   MCL, or Monte Carlo Localization, a localization algorithm based on particle filters.

better the results. Fine-grained approximations also tend to suffer less from "catastrophic" failures where the robot's belief differs significantly from its actual position.

8.3   MONTE CARLO LOCALIZATION

8.3.1   The MCL Algorithm

We will now turn our attention to a popular localization algorithm which represents the belief $bel(x_t)$ by particles. The algorithm is called *Monte Carlo Localization*, or *MCL*. Like grid-based Markov localization, MCL is applicable to both local and global localization problems. Despite its relatively short existence, MCL has already become one of the most popular localization algorithms in robotics. It is easy to implement, and tends to work well across a broad range of localization problems.

Table 8.2 shows the basic MCL algorithm, which is obtained by substituting the appropriate probabilistic motion and perceptual models into the algorithm **particle_filters** (Table 4.3 on page 78). The basic MCL algorithm represents the belief $bel(x_t)$ by a set of M particles $\mathcal{X}_t = \{x_t^{[1]}, x_t^{[2]}, \ldots, x_t^{[M]}\}$. Lines 4 in our algorithm (Table 8.2) sam-

ples from the motion model, using particles from present belief as starting points. The beam measurement model is then applied in line 5 to determine the importance weight of that particle. The initial belief $bel(x_0)$ is obtained by randomly generating M such particles from the prior distribution $p(x_0)$, and assigning the uniform importance factor M^{-1} to each particle. As in grid localization, the functions **motion_model**, and **measurement_model**, may be implemented by any of the motion models in Chapter 5, and measurement models in Chapter 6, respectively.

Figure 8.11 illustrates MCL using the one-dimensional hallway example. The initial global uncertainty is achieved through a set of pose particles drawn at random and uniformly over the entire pose space, as shown in Figure 8.11a. As the robot senses the door, line 5 of in the algorithm **MCL** assigns importance factors to each particle. The resulting particle set is shown in Figure 8.11b. The height of each particle in this figure shows its importance weight. It is important to notice that this set of particles is identical to the one in Figure 8.11a—the only thing modified by the measurement update are the importance weights.

Figure 8.11c shows the particle set after resampling (line 8-11 in the algorithm **MCL**) and after incorporating the robot motion (line 4). This leads to a new particle set with uniform importance weights, but with an increased number of particles near the three likely places. The new measurement assigns non-uniform importance weights to the particle set, as shown in Figure 8.11d. At this point, most of the cumulative probability mass is centered on the second door, which is also the most likely location. Further motion leads to another resampling step, and a step in which a new particle set is generated according to the motion model (Figure 8.11e). As should be obvious from this example, the particle sets approximate the correct posterior, as would be calculated by an exact Bayes filter.

Figure 8.12 shows the result of applying MCL in an actual office environment, for a robot equipped with an array of sonar range finders. The figure depicts particle sets after 5, 28, and 55, meters of robot motion, respectively. Each particle set is defined over the 3-dimensional pose space, although only the x- and y-coordinates of each particle are shown. A second illustration is provided in Figure 8.13, here using a camera pointed towards the ceiling, and a measurement model that relates the brightness in the center of the image to a previously acquired ceiling map.

8.3.2 Properties of MCL

MCL can approximate almost any distribution of practical importance. It is not bound to a limited parametric subset of distributions, as was the case for EKF localization.
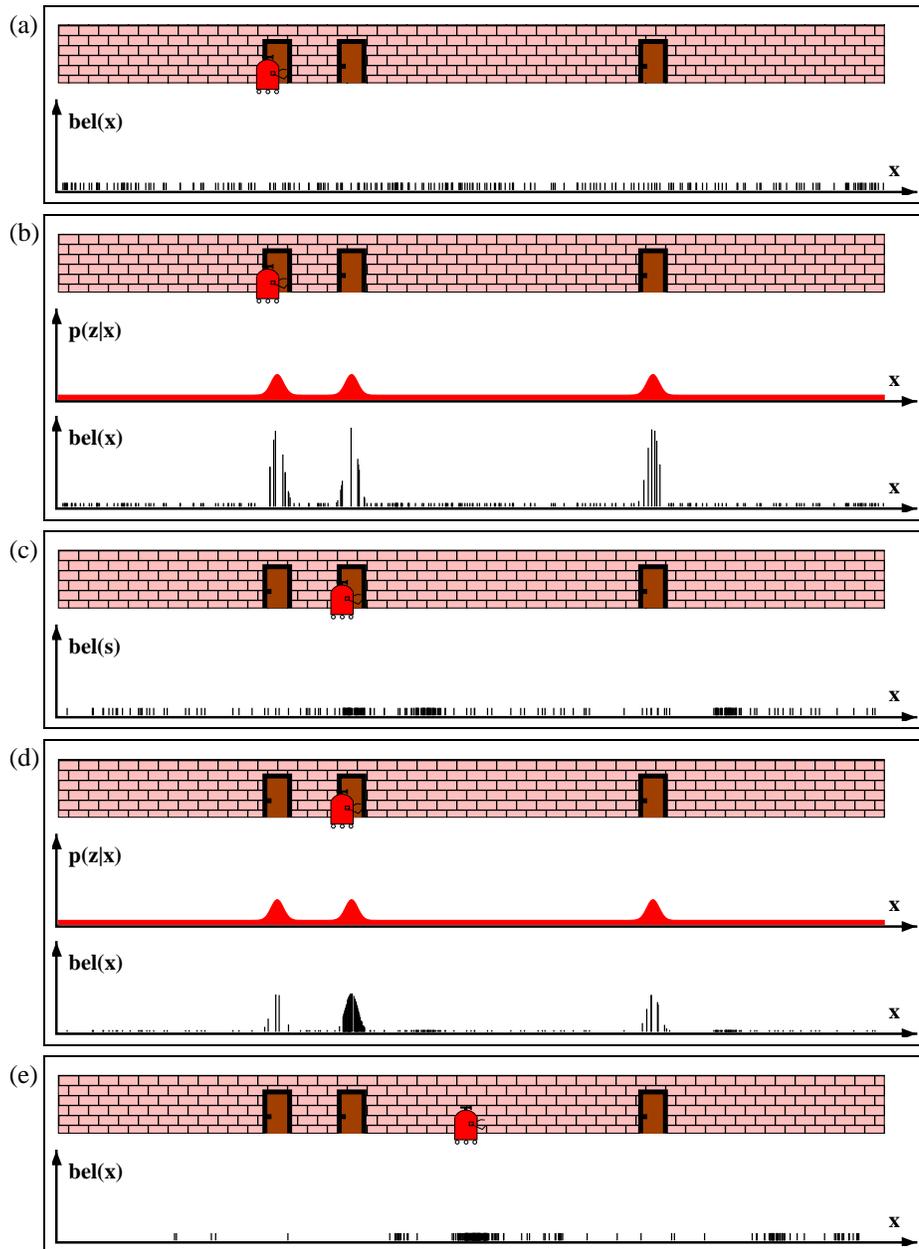
Figure 8.11   Monte Carlo Localization, a particle filter applied to mobile robot localization.
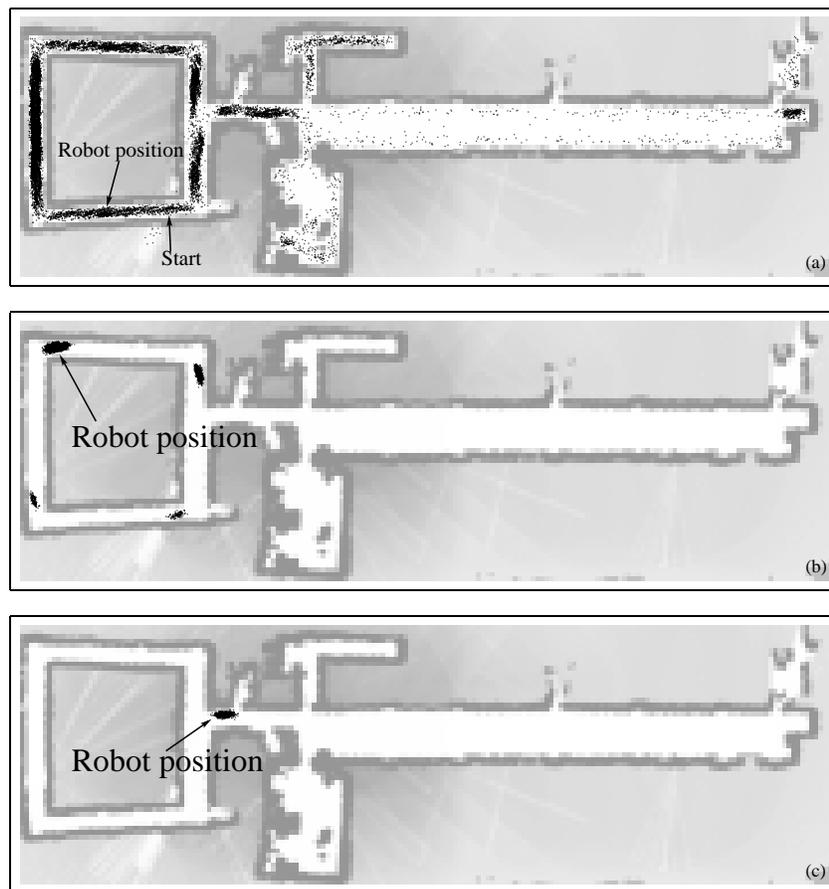
Figure 8.12 Illustration of Monte Carlo localization: Shown here is a robot operating in an office environment of size 54m × 18m. (a) After moving 5m, the robot is still highly uncertain about its position and the particles are spread trough major parts of the free-space. (b) Even as the robot reaches the upper left corner of the map, its belief is still concentrated around four possible locations. (c) Finally, after moving approximately 55m, the ambiguity is resolved and the robot knows where it is. All computation is carried out in real-time on a low-end PC.
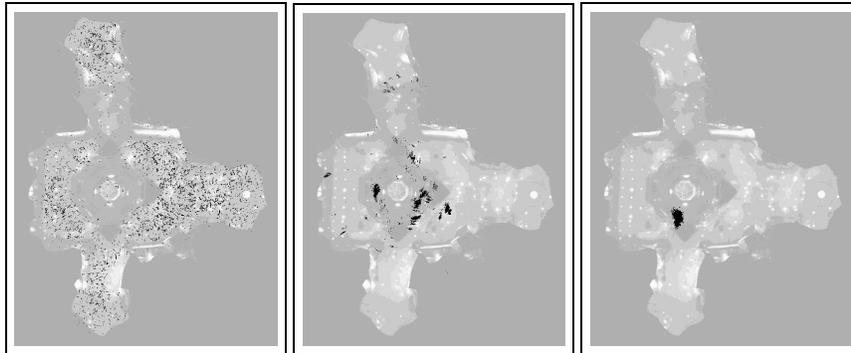
Figure 8.13   Global localization using a camera pointed at the ceiling.

The accuracy of the approximation is easily determined by the size of the particle set M. Increasing the total number of particles increases the accuracy of the approximation. The number of particles M is a parameter that enables the user to trade off the accuracy of the computation and the computational resources necessary to run MCL. A common strategy for setting M is to keep sampling until the next pair u_t and z_t has arrived. In this way, the implementation is adaptive with regards to the computational resources: the faster the underlying processor, the better the localization algorithm. Such a resource-adaptivity is difficult to achieve for grid localization and Gaussian techniques.

A final advantage of MCL pertains to the non-parametric nature of the approximation. As our illustrative results suggest, MCL can represent complex multi-modal probability distributions, and blend them seamlessly with focused Gaussian-style distributions. This provides MCL with the ability to solve global localization problems with high-accuracy position tracking.

8.3.3   Random Particle MCL: Recovery from Failures

MCL, in its present form, solves the global localization problem but cannot recover from robot kidnapping, or global localization failures. This quite obvious from the results in Figure 8.12: As the position is acquired, particles at places other than the

most likely pose gradually disappear. At some point, particles only "survive" near a single pose, and the algorithm is unable to recover if this pose happens to be incorrect.

This problem is significant. In practice, any stochastic algorithm such as MCL may accidentally discard all particles near the correct pose during the resampling step. This problem is particularly paramount when the number of particles is small (e.g., $M = 50$), and when the particles are spread over a large volume (e.g., during global localization).

Fortunately, this problem can be solved by a rather simple heuristic. The idea of this heuristic is to add random particles to the particle sets. Such an "injection" of random particles can be justified mathematically by assuming that the robot might get kidnapped with a small probability, thereby generating a fraction of random states in the motion model. Even if the robot does not get kidnapped, however, the random particles add an additional level of robustness.

The approach of adding particles raises two questions. First, how many particles should be added at each iteration of the algorithm and, second, from which distribution should we generate these particles? One might add a fixed number of random particles at each iteration. A better idea is to add particles based on some estimate of the localization a accuracy. One way to implement this idea is to monitor the probability of sensor measurements

$$p(z_t \mid z_{t-1}, u_t, m) \tag{8.3}$$

and relate it to the average measurement probability (which is easily learned from data). In particle filters, an approximation to this quantity is easily obtained from the importance factor, since, by definition, an importance weight is a stochastic estimate of $p(z_t \mid z_{t-1}, u_t, m)$. The mean

$$p(z_t \mid z_{t-1}, u_t, m) \quad \approx \quad \frac{1}{M} \sum_{m=1}^{M} w_t^{[m]} \tag{8.4}$$

thus approximates the desired quantity. It is usually a good idea to smooth this estimate by averaging it over multiple time steps. There exist multiple reasons why this mean may be poor in addition to poor position tracking: the amount of sensor noise might be unnatural high, or the particles may still be spread out during a global localization phase. For these reasons, it is a good idea to maintain a short-term average of the measurement likelihood, and relate it to the long-term average when determining the number of random samples.

1:         **Algorithm Augmented_MCL($\mathcal{X}_{t-1}, u_t, z_t, m$):**
2:             static $w_{\text{slow}},\ w_{\text{fast}}$
3:             $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
4:             for $m = 1$ to M do
5:                 $x_t^{[m]} = \textbf{sample\_motion\_model}(u_t, x_{t-1}^{[m]})$
6:                 $w_t^{[m]} = \textbf{measurement\_model}(z_t, x_t^{[m]}, m)$
7:                 $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
8:                 $w_{\text{avg}} = w_{\text{avg}} + \frac{1}{M}\, w_t^{[m]}$
9:             endfor
10:            $w_{\text{slow}} = w_{\text{slow}} + \alpha_{\text{slow}}(w_{\text{avg}} - w_{\text{slow}})$
11:            $w_{\text{fast}} = w_{\text{fast}} + \alpha_{\text{fast}}(w_{\text{avg}} - w_{\text{fast}})$
12:            for $m = 1$ to M do
13:                with probability $\max(0.0,\ 1.0 - w_{\text{fast}}/w_{\text{slow}})$ do
14:                    add random pose to $\mathcal{X}_t$
15:                else
16:                    draw $i \in \{1, \dots, N\}$ with probability $\propto w_t^{[i]}$
17:                    add $x_t^{[i]}$ to $\mathcal{X}_t$
18:                endwith
19:            endfor
20:            return $\mathcal{X}_t$

Table 8.3 An adaptive variant of MCL that adds random samples. The number of random samples is determined by comparing the short-term with the long-term likelihood of sensor measurements.

The second problem of determining which sample distribution to use, can be addressed in two ways. One can draw particles according to a uniform distribution over the pose space, and then weight them with the current observation. For some sensor models, however, it is impractical to generate particles directly in accordance to the measurement distribution. One example of such a sensor model is the landmark detection model discussed in Chapter 6.6. In this case the additional particles can be placed directly at locations distributed according to the observation likelihood (see Table 6.5).

Table 8.3 shows a variant of the MCL algorithm that adds random particles. This algorithm is adaptive, in that it tracks the short-term and the long-term average of the likelihood $p(z_t \mid z_{t-1}, u_t, m)$. Its first part is identical to the algorithm **MCL** in Table 8.2: New poses are samples from old particles using the motion model (line 5), and

their importance weight is set in accordance to the measurement model (line 6). However, **Augmented_MCL** calculates the empirical measurement likelihood in line 8, and maintains short-term and long-term averages of this likelihood in lines 10 and 11. The algorithm requires that $0 \leq \alpha_{\text{slow}} \ll \alpha_{\text{fast}}$. The parameters α_{slow}, and α_{fast}, are decay rates for the exponential filters that estimate the long-term, and short-term, averages, respectively. The crux of this algorithm can be found in line 13: During the resampling process, a random sample is added with probability $\max(0.0, \ 1.0 - w_{\text{fast}}/w_{\text{slow}})$; otherwise, resampling proceeds in the familiar way. The probability of adding a random sample takes into consideration the divergence between the short- and the long-term average of the measurement likelihood. If the short-term likelihood is better or equal to the long-term likelihood, no random sample is added. However, if the short-term likelihood is worse than the long-term one, random samples are added in proportion to the quotient of these values. In this way, a sudden decay in measurement likelihood induces an increased number of random samples. The exponential smoothing counteracts the danger of mistaking momentary sensor noise for a poor localization result.

Figure 8.15 illustrates our augmented MCL algorithm in practice. Shown there is a sequence of particle sets during global localization and relocalization of a legged robot equipped with a color camera, and operating on a 3×2m field as it is used in RoboCup soccer competitions. Sensor measurements correspond to the detection and relative localization of six visual markers placed around the field. The algorithm described in Table 6.4 is used to determine the likelihood of detections. Step 14 in Figure 8.3 is replaced by an algorithm for sampling according to the most recent sensor measurement, which is easily implemented for our feature-based sensor model.

Figures 8.15a–d illustrate global localization. At the first marker detection, virtually all particles are drawn according to this detection (Figure 8.15b). This step corresponds to situation in which the short-term average of the measurement probability is much worse than its long-term correspondent. After several more detections, the particles are clustered around the true robot position (Figure 8.15d), and both the short- and long-term average of the measurement likelihood increases. At this stage of localization, the robot is merely tracking its position, the observation likelihoods are rather high, and only a very small number random particles are occasionally added.

As the robot is physically carried to a different location—a common even in robotic soccer tournaments—the measurement probability drops. The first marker detection at this new location does not yet trigger any additional particles, since the smoothed estimate w_{fast} is still high (see Figure 8.15e). After several marker detections observed at the new location, w_{fast} decreases much faster than w_{slow} and more random particles are added (Figure 8.15f,g). Finally, the robot successfully relocalizes itself as shown
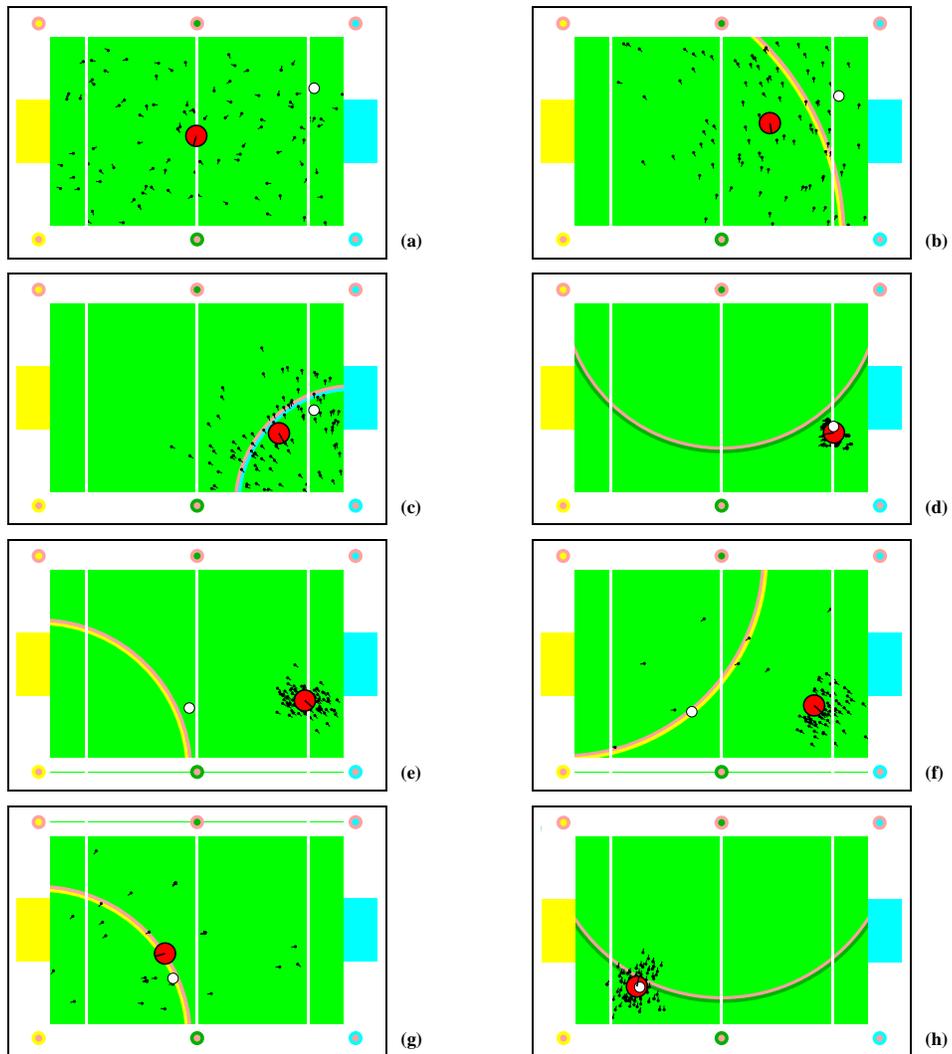
Figure 8.15 Mont Carlo localization with random particles. Each picture shows a particle set representing the robot's position estimate (small lines indicate the orientation of the particles). The large circle depicts the mean of the particles, and the true robot position is indicated by the small, white circle. Marker detections are illustrated by arcs centered at the detected marker. The pictures illustrate global localization (a)–(d) and relocalization (e)–(h).
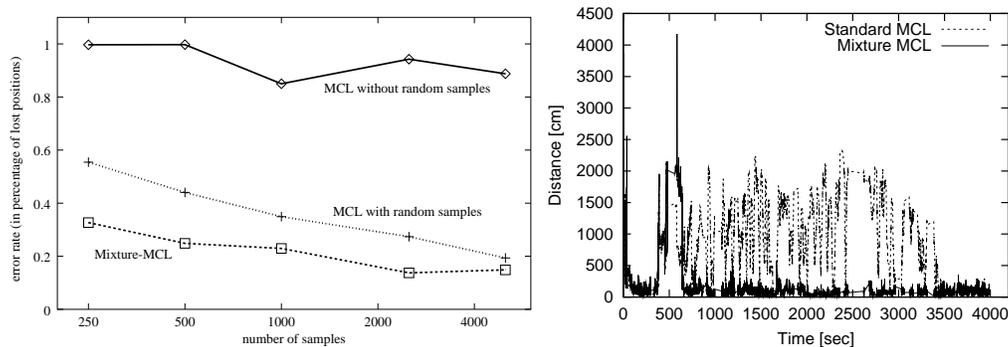
Figure 8.16 Left diagram: plain MCL (top curve), MCL with random samples (center curve), and MCL with mixture proposal distribution (bottom curve). The error rate is measured in percentage of time during which the robot lost track of its position, for a data set acquired by a robot operating in a crowded museum. Right: Error as a function of time for standard MCL and mixture MCL, using a ceiling map for localization.

in Figure 8.15h, demonstrating that our augmented MCL algorithm is indeed capable of "surviving" kidnapping.

8.3.4   Modifying the Proposal Distribution

A related limitation of MCL arises from its proposal mechanism; in fact, it inherits this deficiency from the vanilla particle filter. As discussed in Chapter 4.2.4, the particle filter uses the motion model as proposal distribution, but it seeks to approximate a product of this distribution and the perceptual likelihood. The larger the difference between the proposal and the target distribution, the more samples are needed.

In MCL, this induces a surprising failure mode: If we were to acquire a perfect sensor which—without any noise—always informs the robot of its correct pose, MCL would fail. This is even true for noise-free sensors that do not carry sufficient information for localization. An example of the latter would be a 1-D noise-free range sensor: When receiving such a range measurement, the space of valid pose hypotheses will be a 2-D subspace of the 3-D pose space. We already discussed in length in Section 4.2.4 that chances to sample into this 2-D submanifold are zero when sampling from the robot motion model. Thus, we face the strange situation that under certain circumstances, a less accurate sensor would be preferable to a more accurate sensor when using MCL for localization. This is *not* the case for EKF localization, since the EKF update takes

the measurements into account when calculating the new mean—instead of generating mean(s) from the motion model alone.

Luckily, a simple trick provides remedy: Simply use a measurement model that artificially inflates the amount of noise in the sensor. One can think of this inflation as accommodating not just the measurement uncertainty, but also the uncertainty induced by the approximate nature of the particle filter algorithm.

An alternative, more sound solution involves a modification of the sampling process which we already discussed briefly in Section 4.2.4. The idea is that for a small fraction of all particles, the role of the motion model and the measurement model are reversed: Particles are generated according to the measurement model

$$x_t^{[m]} \quad \sim \quad p(z_t \mid x_t) \tag{8.5}$$

and the importance weight is calculated in proportion to

$$w_t^{[m]} \quad = \quad \int p(x_t^{[m]} \mid u_t, x_{t-1} \ bel(x_{t-1}) \ dx_{t-1} \tag{8.6}$$

This new sampling process is a legitimate alternative to the plain particle filter. It alone will be inefficient since it entire ignores the history when generating particles. However, it is equally legitimate to generate a fraction of the particles with either of those two mechanisms and merge the two particle sets. The resulting algorithm is called *MCL with mixture distribution*, or mixture MCL. In practice, it tends to suffice to generate a small fraction of particles (e.g., 5%) through the new process.

Unfortunately, our ides does not come without challenges. The two main steps—sampling from $p(z_t \mid x_t)$ and calculating the importance weights $w_t^{[m]}$—can be difficult to realize. Sampling from the measurement model is only easy if its inverse possesses a closed form solution from which it is easy to sample. This is usually not the case: imagine sampling from the space of all poses that fit a given laser range scan! Calculating the importance weights is complicated by the integral in (8.6), and by the fact that $bel(x_{t-1})$ is itself represented by a set of particles.

Without delving into too much detail, we note that both steps can be implemented, but only with additional approximations. Figure 8.16 shows comparative results for MCL, MCL augmented with random samples, and mixture MCL for two real-world data sets. In both cases, $p(z_t \mid x_t)$ was itself learned from data and represented by

a density tree—an elaborate procedure whose description is beyond the scope of this book. For calculating the importance weights, the integral was replaced by a stochastic integration, and the prior belief was continued into a space-filling density by convolving each particle with a narrow Gaussian. Details aside, these results illustrate that the mixture idea yields superior results, but it can be challenging to implement.

We also note that the mixture MCL provides a sound solution to the kidnapped robot problem. By seed-starting particles using the most recent measurement only, we constantly generate particles at locations that are plausible given the momentary sensor input, regardless of past measurements and controls. There exist ample evidence in the literature that such approaches can cope well with total localization failure (Figure 8.16b happens to show one such failure for regular MCL), hence provides improved robustness in practical implementations.

8.4 LOCALIZATION IN DYNAMIC ENVIRONMENTS

A key limitation of all localization algorithms discussed thus far arises from the static world assumption, or Markov assumption. Most interesting environments are populated by people, and hence exhibit dynamics not modeled by the state x_t. To some extent, probabilistic approaches are robust to such unmodeled dynamics, due to their ability to accommodate sensor noise. However, as previously noted, the type sensor noise accommodated in the probabilistic filtering framework must be independent at each time step, whereas unmodeled dynamics induce effects on the sensor measurements over multiple time steps. When such effects are paramount, probabilistic localization algorithms that rely on the static world assumption may fail.

A good example of such a failure situation is shown in Figure 8.17. This example involves a mobile tour-guide robot, navigating in museums full of people. The people—their locations, velocities, intentions etc.—are hidden state relative to the localization algorithm which is not captured in the algorithms discussed thus far. Why is this problematic? Imagine people lining up in a way that suggests the robot is facing a wall. As a result, with each single sensor measurement the robot increases its belief of being next to a wall. Since information is treated as independent, the robot will ultimately assign high likelihood to poses near walls. Such an effect is possible with independent sensor noise, but its likelihood is vanishingly small.

There exist two fundamental techniques for dealing with dynamic environments. The first technique includes the hidden state into the state estimated by the filter; the
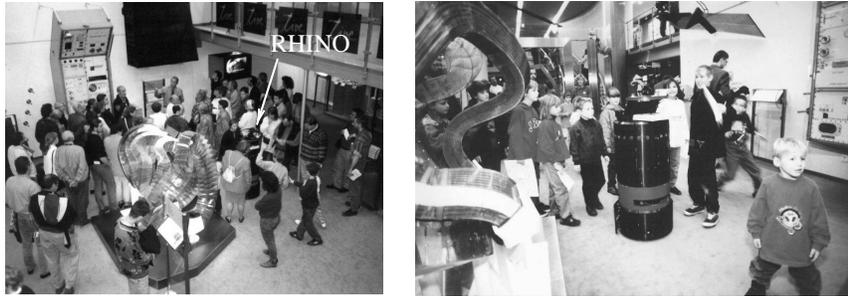
Figure 8.17   Scenes from the Deutsches Museum Bonn, where the mobile robot 'Rhino' was frequently surrounded by dozens of people.

other preprocesses sensor measurements to eliminate measurements affected by hidden state. The former methodology is mathematically the more general one: Instead of just estimating the robot's pose, one can define filter that also estimates people's positions, their velocities, etc. In fact, we will later on discuss such an approach, as an extension to a mobile robot mapping algorithm.

The principle disadvantage of estimating the hidden state variables lies in its computational complexity: Instead of estimating 3 variables, the robot must now calculate posteriors over a much larger number of variables. In fact, the number of variables itself is a variable, as the number of people may vary over time. Thus, the resulting algorithm will be substantially more involved than the localization algorithms discussed thus far.

The alternative—preprocessing sensor data—works well in certain limited situations, which includes situations where people's presence may affect range finders or (to a lesser extent) camera images. Here we develop it for the beam-based range finder model from Chapter 6.3.

The idea is to investigate the *cause* of a sensor measurement, and to reject those likely to be affected by unmodeled environment dynamics. The sensor models discussed thus far all address different, alternative ways by which a measurement can come into existence. If we manage to associate specific ways with the presence of unwanted dynamic effects—such as people—all we have to do it to discard those measurements that are with high likelihood caused by such an unmodeled entity. This idea is surprisingly general. In Equation(6.13), Chapter 6.3, we defined the beam-based measurement
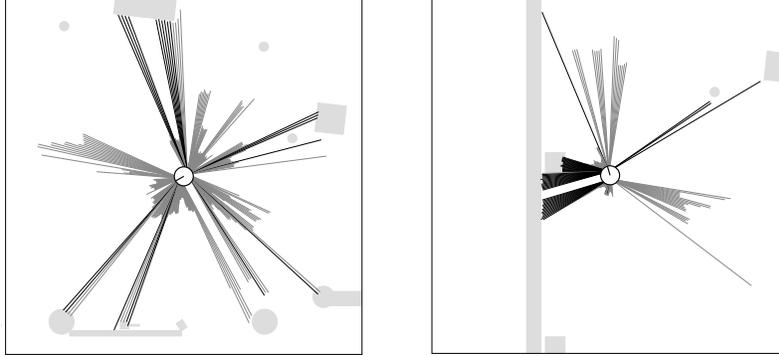
Figure 8.18 Laser range scans are often heavily corrupted when people surround the robot. How can a robot maintain accurate localization under such circumstances?

model for range finders as a mixture of four terms:

$$
p(z_t^k \mid x_t, m) \;\;=\;\;
\begin{pmatrix} z_{\text{hit}} \\ z_{\text{short}} \\ z_{\text{max}} \\ z_{\text{rand}} \end{pmatrix}^T
\cdot
\begin{pmatrix} p_{\text{hit}}(z_t^k \mid x_t, m) \\ p_{\text{short}}(z_t^k \mid x_t, m) \\ p_{\text{max}}(z_t^k \mid x_t, m) \\ p_{\text{rand}}(z_t^k \mid x_t, m) \end{pmatrix}
\tag{8.7}
$$

As our derivation of the model clearly states, One of those terms, the one involving z_{short} and p_{short}, corresponds to unexpected objects. To calculate the probability that a measurement z_t^k corresponds to an unexpected object, we have to introduce a new correspondence variable, $\bar{c}_t^k$ which can take on one of the four values $\{\text{hit}, \text{short}, \text{max}, \text{rand}\}$.

The posterior probability that the range measurement z_t^k corresponds to a "short" reading—our mnemonic from Chapter 6.3 for unexpected obstacle—is then obtained by applying Bayes rule and subsequently dropping irrelevant conditioning variables:

$$
\begin{aligned}
& p(\bar{c}_t^k = \text{short} \mid z_t^k, z_{1:t-1}, u_{1:t}, m) \\
&= \frac{p(z_t^k \mid \bar{c}_t^k = \text{short}, z_{1:t-1}, u_{1:t}, m)\, p(\bar{c}_t^k = \text{short} \mid z_{1:t-1}, u_{1:t}, m)}{\displaystyle\sum_c p(z_t^k \mid \bar{c}_t^k = c, z_{1:t-1}, u_{1:t}, m)\, p(\bar{c}_t^k = c \mid z_{1:t-1}, u_{1:t}, m)}
\end{aligned}
$$

$$= \quad \frac{p(z_t^k \mid \bar{c}_t^k = \text{short}, z_{1:t-1}, u_{1:t}, m) \; p(\bar{c}_t^k = \text{short})}{\sum_c p(z_t^k \mid \bar{c}_t^k = c, z_{1:t-1}, u_{1:t}, m) \; p(\bar{c}_t^k = c)} \tag{8.8}$$

Here the variable c in the denominator takes on any of the four values $\{\text{hit}, \text{short}, \text{max}, \text{rand}\}$. Using the notation in Equation (8.7), the prior $p(\bar{c}_t^k = c)$ is given by the variables z_{hit}, z_{short}, z_{max}, and z_{rand}, for the four different values of c. The remaining probability in (8.8) is obtained by integrating out the pose x_t:

$$p(z_t^k \mid \bar{c}_t^k = c, z_{1:t-1}, u_{1:t}, m)$$

$$= \quad \int p(z_t^k \mid x_t, \bar{c}_t^k = c, z_{1:t-1}, u_{1:t}, m) \; p(x_t \mid \bar{c}_t^k = c, z_{1:t-1}, u_{1:t}, m) \; dx_t$$

$$= \quad \int p(z_t^k \mid x_t, \bar{c}_t^k = c, m) \; p(x_t \mid z_{1:t-1}, u_{1:t}, m) \; dx_t$$

$$= \quad \int p(z_t^k \mid x_t, \bar{c}_t^k = c, m) \; \overline{bel}(x_t) \; dx_t \tag{8.9}$$

Probabilities of the form $p(z_t^k \mid x_t, \bar{c}_t^k = c, m)$ were abbreviated as p_{hit}, p_{short}, p_{max}, and p_{rand} in Chapter 6.3. This gives us the expression for desired probability (8.8):

$$p(\bar{c}_t^k = \text{short} \mid z_t^k, z_{1:t-1}, u_{1:t}, m) \quad = \quad \frac{\displaystyle\int p_{\text{short}}(z_t^k \mid x_t, m) \; z_{\text{short}} \; \overline{bel}(x_t) \; dx_t}{\displaystyle\int \sum_c p_c(z_t^k \mid x_t, m) \; z_c \; \overline{bel}(x_t) \; dx_t}$$

$$\tag{8.10}$$

In general, these integrals do not possess closed-form solutions. To evaluate them, it suffices to approximate them with a representative sample of the posterior $\overline{bel}(x_t)$ over the state x_t. Those samples might be high-likelihood grid cells in grid localizer, or particles in a MCL algorithm. The measurement is then rejected if its probability of being caused by an unexpected obstacle exceeds a user-selected threshold χ.

Table 8.4 depicts an implementation of this technique in the context of particle filters. It requires as input a particle set $\bar{\mathcal{X}}_t$ representative of the belief $\overline{bel}(x_t)$, along with a range measurement z_t^k and a map. It returns "reject" if with probability larger than χ the measurement corresponds to an unexpected object; otherwise it returns "accept." This routine precedes the measurement integration step in MCL.

```
1:          Algorithm test_range_measurement(z_t^k, X̄_t, m):
2:              p = q = 0
3:              for m = 1 to M do
4:                  p = p + z_hit · p_hit(z_t^k | x_t^[m], m)
5:                  q = q + z_hit · p_hit(z_t^k | x_t^[m], m) + z_short · p_short(z_t^k | x_t^[m], m)
6:                      +z_max · p_max(z_t^k | x_t^[m], m) + z_rand · p_rand(z_t^k | x_t^[m], m)
7:              endfor
8:              if p/q ≤ χ then
9:                  return accept
10:             else
11:                 return reject
12:             endif
```

Table 8.4 Algorithm for testing range measurements in dynamic environment.

Figure 8.19 illustrates the effect of the filter. Shown in both panels are a range scan, for a different alignment of the robot pose. The lightly shaded scans are above threshold and rejected. A key property of our rejection mechanism is that it tends to filter out measurements that are "surprisingly" short, but leaves others in place that are "surprisingly" long. This asymmetry reflects the fact that people's presence tends to cause shorter-than-expected measurements. By accepting surprisingly long measurements, the approach maintains its ability to recover from global localization failures.

Figure 8.20 depicts an episode during which a robot navigates through an environment that densely populated with people (see Figure 8.18). Shown there is the robot's estimated path along with the endpoints of all scans incorporated into the localizer. This figure shows the effectiveness of removing measurements that do not correspond to physical objects in the map: there are very few "surviving" range measurements in the freespace for the right diagram, in which measurements are accepted only if they surpass the threshold test.

As a rule of thumb, filtering out measurements is generally a good idea. There exist almost no static environments; even in office environments furniture is moved, doors are opened/closed, etc. Our specific implementation here benefits from the asymmetry of range measurements: people make measurements shorter, not longer. When applying the same idea to other data (e.g., vision data) or other types of environment modifications (e.g., the removal of a physical obstacle), such an asymmetry might not exist. Nevertheless, the same probabilistic analysis is usually applicable. The disadvantage
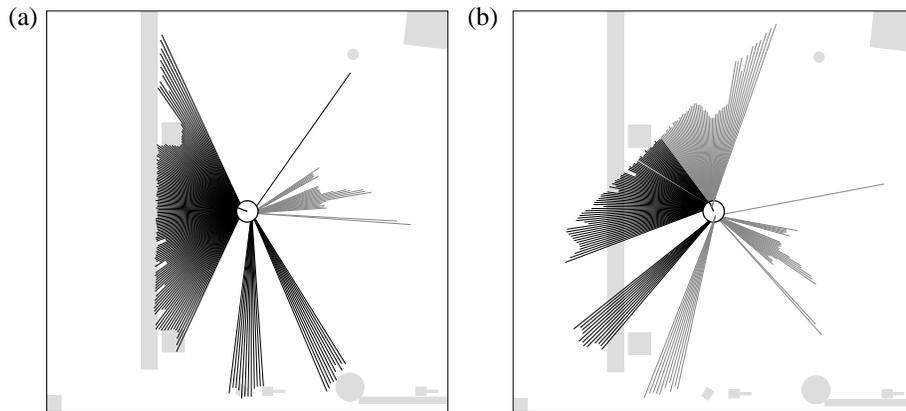
Figure 8.19   Illustration of our measurement rejection algorithm: Shown in both diagrams are range scans (no max-range readings). Lightly shaded readings are filtered out.

of the lack of such a symmetry might be that it becomes impossible to recover from global localization failures, as every surprising measurement is rejected. In such cases, it may make sense to impose additional constraints, such as a limit on the fraction of measurements that may be corrupted.

We note that our rejection test has found successful application even in highly static environments, for reasons that are quite subtle. The beam-based sensor model is discontinuous: Small changes of pose can drastically alter the posterior probability of a sensor measurement. This is because the result of ray casting is not a continuous function in pose parameters such as the robot orientation. In environment with cluttered objects, this discontinuity increases the number of particle necessary for successful localization. By manually removing clutter from the map—and instead letting the filter manage the resulting "surprising" short measurements—the number of particles can be reduced drastically. The same strategy does not apply to the likelihood field model, since this model is smooth in the pose parameters.

8.5   PRACTICAL CONSIDERATIONS

Table 8.5 summarizes the main localization techniques discussed in this and the previous chapter. When choosing a technique, a number of requirements have to be traded off. A first question will always be whether it is preferable to extract features from
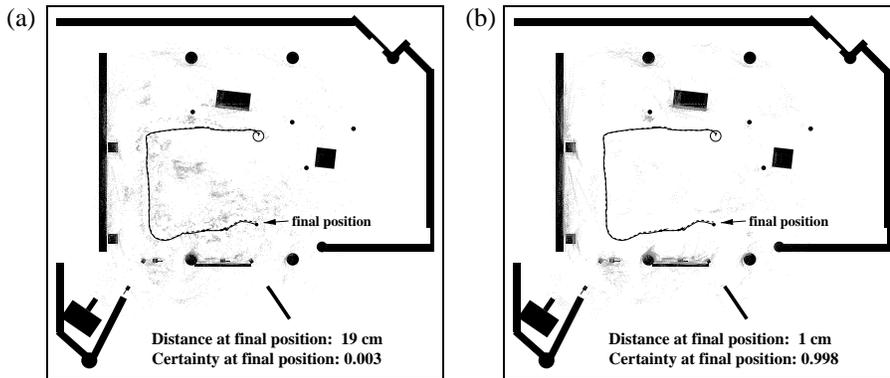
Figure 8.20 Comparison of (a) standard MCL and (b) MCL with the removal of sensor measurements likely caused by unexpected obstacles. Both diagrams show the robot path and the end-points of the scans used for localization.

sensor measurement. Extracting features may be beneficial from a computational perspective, but it comes at the price of reduced accuracy and robustness.

	EKF	MHT	Coarse (topological) grid	fine (metric) grid	MCL
Measurements	landmarks	landmarks	landmarks	raw measurements	raw measurements
Measurement noise	Gaussian	Gaussian	any	any	any
Posterior	Gaussian	mixture of Gaussians	histogram	histogram	particles
Efficiency (memory)	++	++	+	−	+
Efficiency (time)	++	+	+	−	+
Ease of implementation	+	−	+	−	++
Resolution	++	++	−	+	+
Robustness	−	+	+	++	++
Global localization	no	no	yes	yes	yes

Table 8.5 Comparison of different implementations of Markov localization.

While in this chapter, we developed techniques for handling dynamic environments in the context of the MCL algorithm, similar ideas can be brought to bear with other localization techniques as well. In fact, the techniques discussed here are only representative of a much richer body of approaches.

When implementing a localization algorithm, it is worthwhile to play with the various parameter settings. For example, the conditional probabilities are often inflated when integrating nearby measurements, so as to accommodate unmodeled dependencies that always exist in robotics. A good strategy is to collect reference data sets, and tune the algorithm until the overall result is satisfactory. This is necessary because no matter how sophisticated the mathematical model, there will always remain unmodeled dependencies and sources of systematic noise that affect the overall result.

8.6   SUMMARY

In this chapter, we discussed two families of probabilistic localization algorithms, grid techniques and Monte Carlo localization (MCL).

■   Grid techniques represent posteriors through histograms.

■   The coarseness of the grid trades off accuracy and computational efficiency. For course grids, it is usually necessary to adjust the sensor and motion models to account for effects that arise from the coarseness of the representation. For fine grids, it may be necessary to update grid cells selectively to reduce the overall computation.

■   The Monte Carlo localization algorithm represents the posterior using particles. The accuracy-computational costs trade-off is achieved through the size of the particle set.

■   Both grid localization and MCL can globally localize robots.

■   By adding random particles, MCL also solves the kidnapped robot problem.

■   Mixture-MCL is an extension which inverts the particle generation process for a fraction of all particles. This yields improved performance specifically for robots with low-noise sensors, but at the expense of a more complex implementation.

■   Unmodeled environment dynamics can be accommodated by filtering sensor data, rejecting those that with high likelihood correspond to an unmodeled object. When using range sensors, the robot tends to reject measurements that are surprisingly short.

8.7 EXERCISES

Derive the additive form of the multi-feature information integration in lines 14 and 15 in Table 7.2.

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, that 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 which 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 is often referred to as the *simultaneous localization and mapping* (SLAM) or *concurrent mapping and localization problem* 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. Likewise, constructing a map when the robot's poses are known is also relatively easy—a claim that will be substantiated by 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 the 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, including the techniques described in this chapter. This example gives a good flavor of problem at stake.

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. We will discuss a popular family of algorithms, collectively called *occupancy grids*.
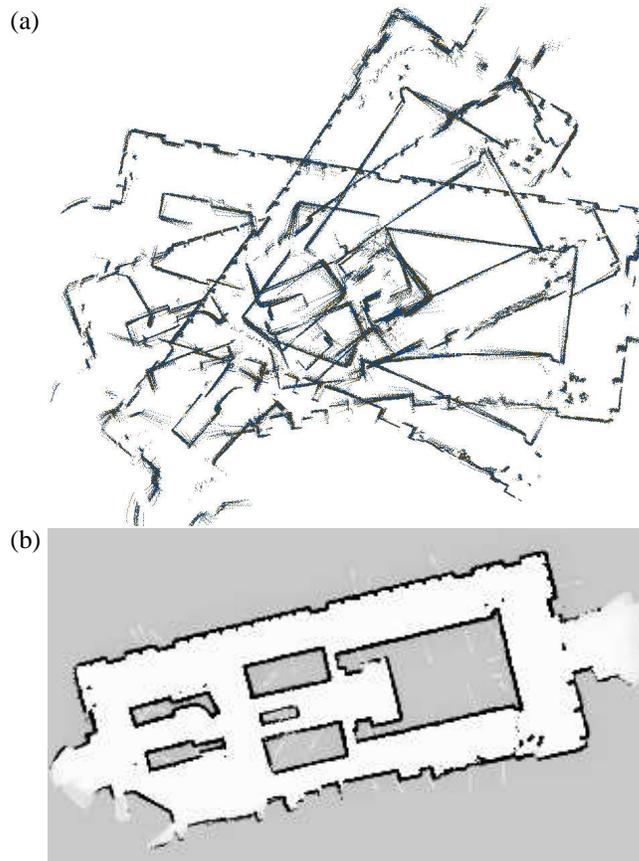
Figure 9.1 (a) Raw range data, position indexed by odometry. (b) Map

Occupancy grid maps address 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 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 is 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 navi-

gation. 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

To 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, that is, the sequence of all its 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 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 floorplan maps, which describe a 2-D slice of the 3-D world. 2-D maps are often sufficient, especially 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:

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

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 $p(\mathbf{m}_i)$ will refer to a probability that a grid cell is occupied.

The problem with the posterior (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, hence the number of maps defined in this space is often in the excess of $2^{10,000}$. Calculating a posterior for each single such 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 enable us to represent dependencies among neighboring cells; instead, the posterior over maps is approximated as the product of its marginals:

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

We will return to this issue in Section 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 4.1.4, with the corresponding algorithm depicted in Table 4.2 on page 75. 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:

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

This representation is already familiar from Chapter 4.1.4. 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:

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

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**

1:       **Algorithm occupancy_grid_mapping($\{l_{t-1,i}\}, x_t, z_t$):**
2:           for all cells $\mathbf{m}_i$ do
3:               if $\mathbf{m}_i$ in perceptual field of z_t then
4:                   $l_{t,i} = l_{t-1,i} + \textbf{inverse\_sensor\_model}(\mathbf{m}_i, x_t, z_t) - l_0$
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.

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:

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

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

$$\textbf{inverse\_sensor\_model}(\mathbf{m}_i, x_t, z_t) \;\; = \;\; p(\mathbf{m}_i \mid z_t, x_t) \tag{9.8}$$

A somewhat simplistic example of such a function for range finders is given in Table 9.2 and illustrated in Figure 9.7a&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.

The algorithm **occupancy_grid_mapping** 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
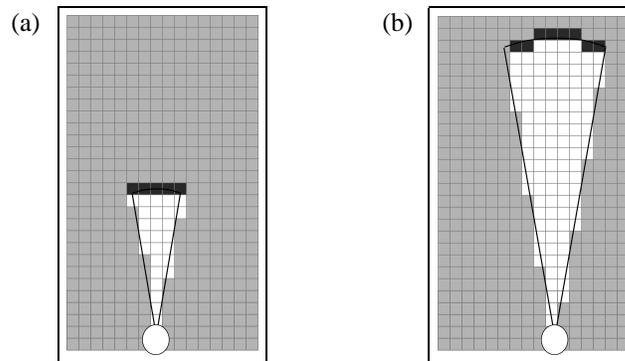
Figure 9.2 Two examples of our inverse measurement model **inverse_range_sensor_model** for two different measurement ranges. The darkness of each grid cell corresponds to the likelihood of occupancy.

$l_{\text{occ}} > l_0$ is 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 cell is shorter than the measured range by more than $\alpha/2$. The left and center panel of Figure 9.7 illustrates this calculation for a sonar beam with a $15°$ opening angle.

Figures **??** 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' time. The grey-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 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 visible at the height of the laser. Close inspection alleviates discrepancies between the blueprint and the actual environment configuration.

Figure 9.4 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 outweigh that of occupancy.

(a)

(b)
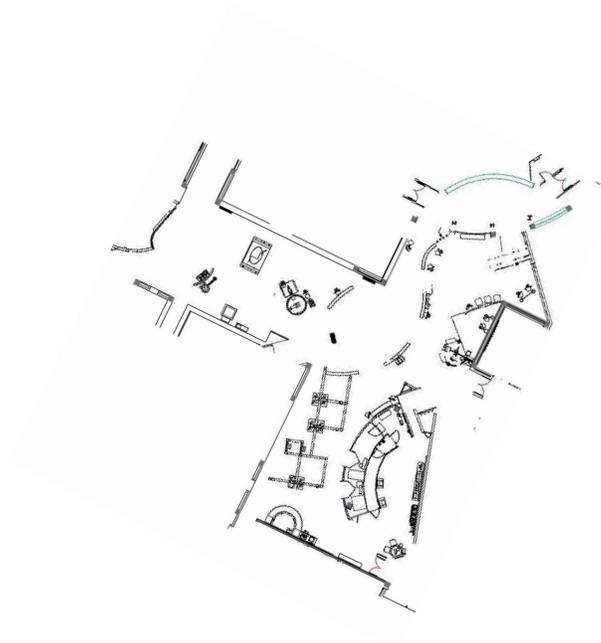
Figure 9.3   (a) Occupancy grid map and (b) architectural blue-print of a large open exhibit space. Notice that the blue-print is inaccurate in certain places.

(a)

(b)
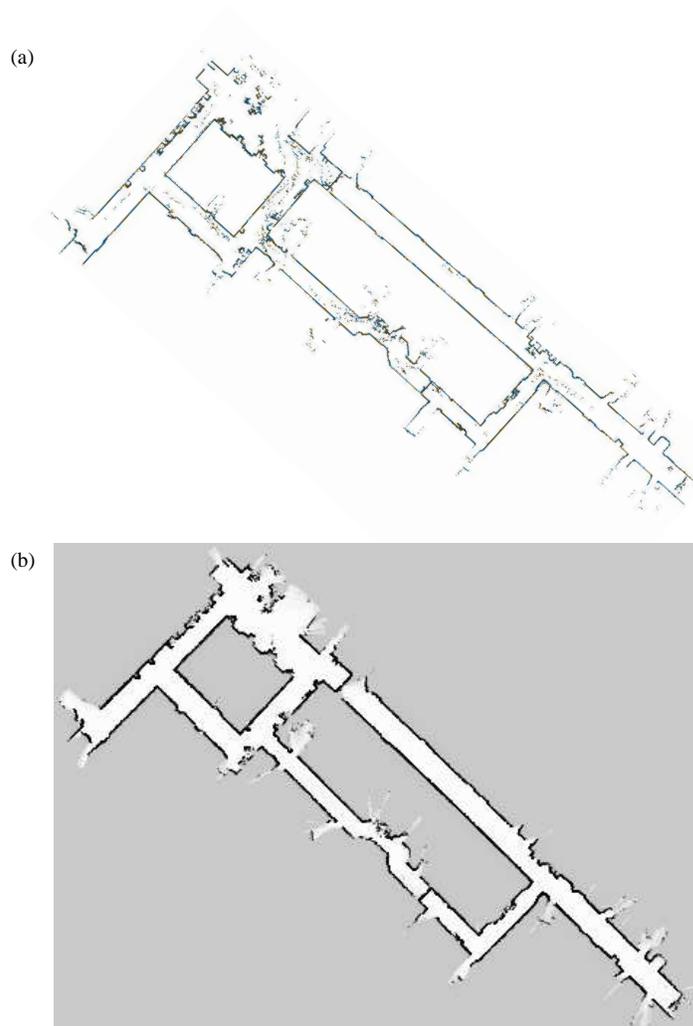
Figure 9.4 (a) Raw laser range data with correct(ed) pose information. Each dot corresponds to a detection of an obstacle. Most obstacles are static (walls etc.), but some were people that walked in the vicinity of the robot. (b) Occupancy grid map built from the data. The grey-scale indicates the posterior probability: Black corresponds to occupied with high certainty, and white to free with high certainty. The grey background color represents the prior. Figure (a) has been generated by Steffen Gutmann.

```
1:       Algorithm inverse_range_sensor_model(i, x_t, z_t):
2:           Let x_i, y_i be the center-of-mass of m_i
3:           r = √((x_i − x)² + (y_i − y)²)
4:           φ = atan2(y_i − y, x_i − x) − θ
5:           k = argmin_j |φ − θ_{j,sens}|
6:           if r > min(z_max, z_t^k + α/2) or |φ − θ_{k,sens}| > β/2 then
7:               return l_0
8:           if z_t^k < z_max and |r − z_max| < α/2
9:               return l_occ
10:          if r ≤ 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 difference cases.

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 sensors 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 9.5 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, in that that are sensitive to different types obstacles.
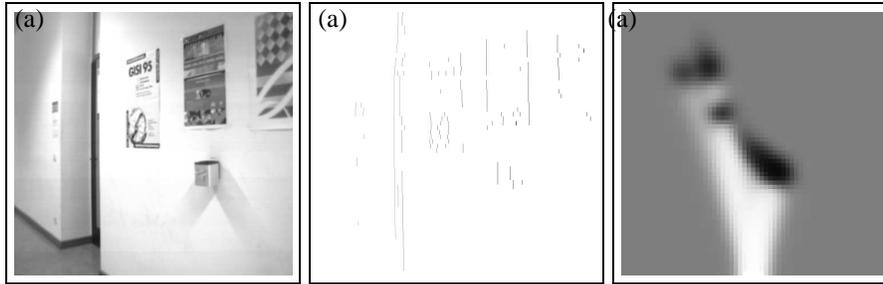
Figure 9.5 Estimation of occupancy maps using stereo vision: (a) camera image, (b) sparse disparity map, (c) occupancy map by projecting the disparity image onto the 2-D plane and convolving the result with a Gaussian.

There are two basic approaches for fusing data from multiple sensors is to use Bayes filters for sensor integration. We can execute the algorithm **occupancy_grid_mapping** in Table 9.1 with different sensor modalities, replacing the function **inverse_sensor_model** accordingly. 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, consider an obstacle that can be recognized by sensor A but not by sensor B. Then these two sensors 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.

The second, more appropriate approach to integrating information from multiple sensors is to build separate maps for each sensor types, and integrate them using the most conservative estimate. Let $m^k = \{\mathbf{m}_i^k\}$ denote the map built by the k-th sensor type. Then the combined map is given by

$$\mathbf{m}_i \quad = \quad \max_k \mathbf{m}_i^k \tag{9.9}$$

This map is the most pessimistic map given its components: If any of the sensor-specific map shows that a grid cell is occupied, so will the combined map. While this combined estimator is biased in factor of occupied maps, it is more appropriate than the Bayes filter approach when sensors with different characteristics are fused.

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

$$p(m \mid x, z) \quad = \quad \frac{p(z \mid x, m) \; p(m \mid x)}{p(z \mid x)} \tag{9.10}$$

$$= \quad \eta \; p(z \mid x, m) \; p(m) \tag{9.11}$$

Here we silently assume $p(m \mid x) = p(m)$, that is, 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:

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

This expression integrates over all maps m for which the occupancy value of grid cell i equals $\mathbf{m}_i$. Clearly, this integral 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 function approximator (or supervised learning algorithm), such as a polynomial, logistic regression, or a neural network.

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 an measurement $z_t^{[k]} \sim p(z \mid x_t^{[k]}, m^{[k]})$. This sampling step is reminiscent of a robot simulator which 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 the the grid cell $\mathbf{m}_i$. Repeated application of this sampling step yields a data set

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

These triplets may serve as training examples for function approximator, to approximate 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 $\text{occ}(\mathbf{m}_i)$ is a target for the output of the function approximator.

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 function approximator.

■ 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:

$$\begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x_{\mathbf{m}_i} - x \\ y_{\mathbf{m}_i} - y \end{pmatrix} \tag{9.14}$$

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

■ Nearby grid cells should have a similar interpretation under the inverse sensor model. This smoothness suggest 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 beam is characterized by the same inverse sensor model.

The most basic way to enforce these invariances is to constrain the function approximator by choosing appropriate input variables. A good choice is to use relative pose information, so that the function approximator 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 function approximator, we need an approximate error function. A popular example are artificial neural networks trained with the Back-propagation algorithm. Back-propagation trains neural networks by gradient descent in parameter

space. Given an error function that measures the "mismatch" between the network's actual and desired output, Back-propagation 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 function approximator so as to maximize the log-likelihood of the training data. More specifically we are given a training set of the form

$$
\begin{aligned}
\text{input}^{[1]} &\longrightarrow \text{occ}(\mathbf{m}_i)^{[1]} \\
\text{input}^{[2]} &\longrightarrow \text{occ}(\mathbf{m}_i)^{[2]} \\
\text{input}^{[3]} &\longrightarrow \text{occ}(\mathbf{m}_i)^{[3]} \\
\vdots \qquad & \qquad \vdots
\end{aligned}
\tag{9.15}
$$

$occ(\mathbf{m}_i)^{[k]}$ is the i-th sample of the desired conditional probability, and $\text{input}^{[k]}$ is the corresponding input to the function approximator. 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 plays no role in the form of the error function.

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

$$
\prod_i p(\mathbf{m}_i^{[k]} \mid \text{input}^{[k]}, W)
\tag{9.16}
$$

and its logarithm is

$$
J(W) \quad = \quad \sum_i \log p(\mathbf{m}_i^{[k]} \mid \text{input}^{[k]}, W)
\tag{9.17}
$$

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

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

output the probability of occupancy, that is:

$$p(\mathbf{m}_i^{[k]} \mid \text{input}^{[k]}, W) \quad = \quad \begin{cases} f(\text{input}^{[k]}, W) & \text{if } \mathbf{m}_i^{[k]} = 1 \\ 1 - f(\text{input}^{[k]}, W) & \text{if } \mathbf{m}_i^{[k]} = 0 \end{cases} \qquad (9.18)$$

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 (9.18) as follows:

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

$$(9.19)$$

It is easy to see that this product and Expression (9.18) are identical. In the product, one of the terms is always 1, since its exponent is zero. Substituting the product into (9.20) and multiplying the result by minus one gives us the following function:

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

$$(9.20)$$

$J(W)$ is the error function to minimize when training the function approximator. It is easily folded into any function approximator that uses gradient descent.

9.3.4   Further Considerations

Figure 9.6 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 relative coordinates in polar coordinates, and 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 gray-ish area behind obstacles matches the prior probability of occupancy, which leads to no change when used in the occupancy grid mapping algorithm. Figure 9.6b contains a faulty short reading on the bottom left. Here a single reading seems to be insufficient to predict an obstacle with high probability.
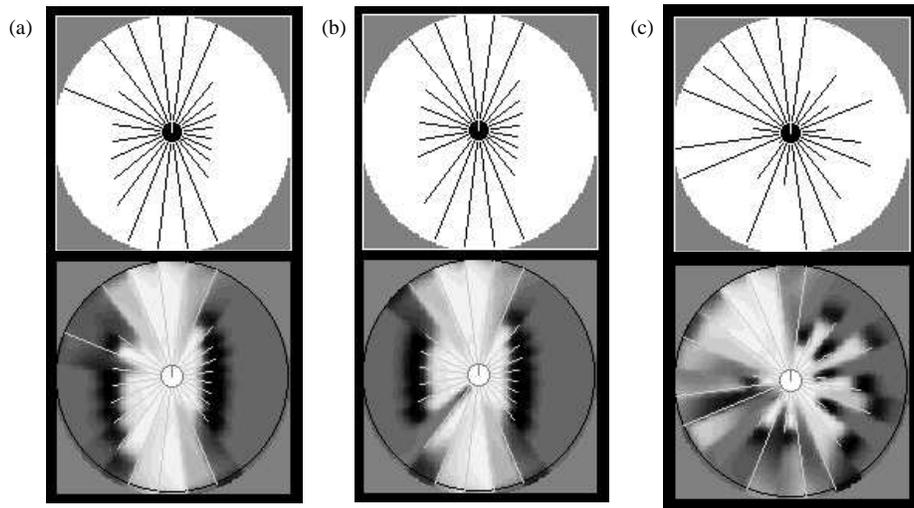
Figure 9.6 Sensor interpretation: Three sample sonar scans (top row) and local occupancy maps (bottom row), as generated by the neural network. Bright regions indicate free-space, and dark regions indicate walls and obstacles (enlarged by a robot diameter).

We note that there exists a number of ways to train a function approximator using actual data collected by a robot, instead 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 approximate map, use the learned sensor model to generate a better map, and from that map use the procedure just outlined to improve the inverse measurement model. As this book is being written, the issue of learning inverse sensor models from data remains relatively unexplored.
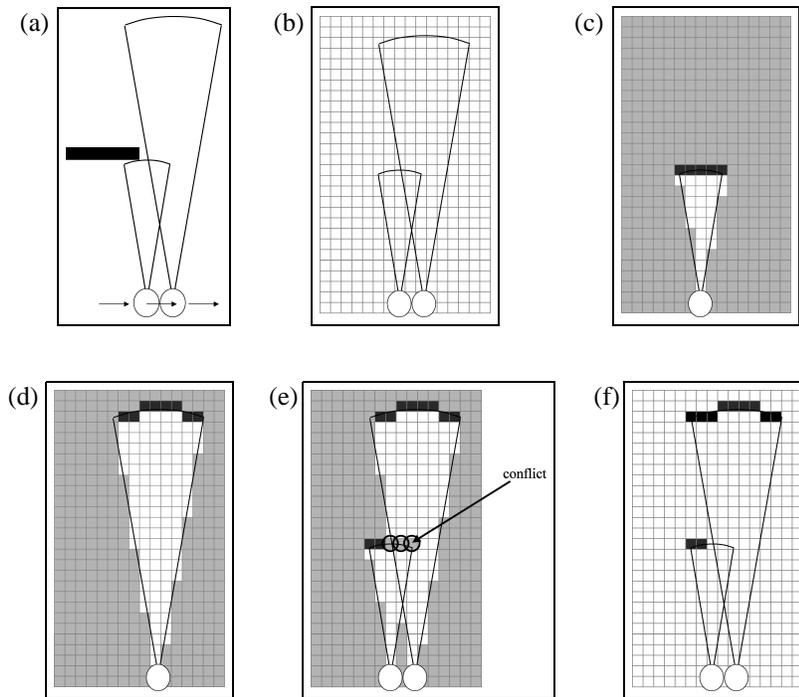
Figure 9.7 The problem with the standard occupancy grid mapping algorithm in Section 9.2: For the environment shown in Figure (a), a passing robot might receive the (noise-free) measurement shown in (b). The factorial approach maps these beams into probabilistic maps separately for each grid cell and 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), which 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.

9.4   MAXIMUM A POSTERIOR 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 Section 9.2, we assumed that we can safely

---

1:        **Algorithm MAP_occupancy_grid_mapping**($x_{1:t}, z_{1:t}$)**:**
2:            set $m = \{0\}$
3:            repeat until convergence
4:                for all cells $\mathbf{m}_i$ do
5:                    $m_i = \underset{k=0,1}{\operatorname{argmax}}(l_0)^k + \sum_t \log$
                            $\mathbf{measurement\_model}(z_t, x_t, m \text{ with } \mathbf{m}_i = k)$
6:                endfor
7:            endrepeat
8:            return m

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

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). 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 9.7 illustrates a problem that arises directly as a result of this factorization. Shown there is a situation in which the 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 9.7c&d, a conflict is created, as shown in Figure 9.7e. The standard occupancy grid mapping algorithm "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 9.7f, which perfectly explains 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, which we already encountered in Equation (9.1):

$$m^* = \underset{m}{\operatorname{argmax}} \log p(m \mid z_{1:t}, x_{1:t}) \tag{9.21}$$

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

$$\log p(m \mid z_{1:t}, x_{1:t}) = \log p(z_{1:t} \mid x_{1:t}, m) + \log p(m) \tag{9.22}$$

The log-likelihood $\log p(z_{1:t} \mid x_{1:t}, m)$ decomposes into a sum all individual measurement log-likelihoods $\log p(z_t \mid x_t, m)$. Further, the log-prior $\log p(m)$ is simply a sum of the type

$$
\begin{aligned}
\log p(m) &= \sum_i [\log p(\mathbf{m}_i)]^{\mathbf{m}_i} + [\log(1 - p(\mathbf{m}_i))]^{1-\mathbf{m}_i} \\
&= M \log(1 - p(\mathbf{m}_i)) + \sum_i (l_0)^{\mathbf{m}_i} \tag{9.23}
\end{aligned}
$$

where M denotes the number of grid cells, and 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:

$$m^* = \underset{m}{\operatorname{argmax}} \sum_t \log p(z_t \mid x_t, m) + \sum_i (l_0)^{\mathbf{m}_i} \tag{9.24}$$

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 $p(\mathbf{m}_i)$ 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.
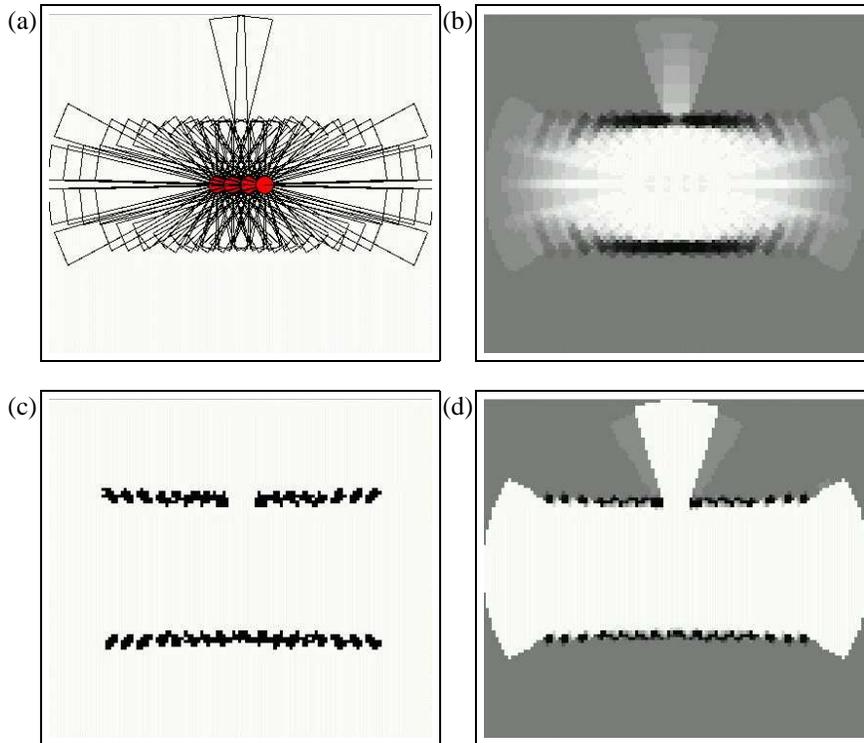
Figure 9.8 (a) sonar range measurements from a noise-free simulation; (b) Results of the standard occupancy mapper, lacking the open door. (c) A maximum a posterior map. (d) The residual uncertainty in this map, obtained by measuring the sensitivity of the map likelihood function with respect to individual grid cells..

Figure 9.8 illustrates the effect of the MAP occupancy grid algorithm. Figure 9.8a 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 9.8b. The mode of the posterior is shown in Figure 9.8c. This map models the open door correctly, hence it is better suited for robot navigation than the standard occupancy grid map algorithm. Figure 9.8d 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. It lacks the vertical stripes found in Figure 9.8a.

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 posterior 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. We leave the design of an appropriate data a structure to the reader as an exercise.

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, and extracting the most most pessimistic occupancy value when making navigation decisions. The latter procedure is 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 function approximation.

■ 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. 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.

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 have access to its own poses. 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:

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

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 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:

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

This subtle difference in the formulation of the SLAM problem between online and full SLAM has ramifications in the type 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:

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

In online SLAM, these integrations are typically performed one-at-a-time, and 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.

We already encountered similar continuous-discrete estimation problems in previous chapters. For example, EKF localization 7.5 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:

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

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 algorithm 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 assignment 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 chapter. 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 7.5, 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 the EKF, 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 or landmarks as features.

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

- **Positive measurements.** The EKF SLAM algorithm, just like the EKF localizer discussed in Chapter 7.5, can only process positive sightings of landmarks. It cannot process negative information that arises from the absence of landmarks in a sensor measurements. This is a direct consequence of the Gaussian belief representation and was already discussed in Chapter 7.5.

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 7.5, 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.

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

2: $F_x = \begin{pmatrix} 1 & 0 & 0 & 0\cdots0 \\ 0 & 1 & 0 & 0\cdots0 \\ 0 & 0 & 1 & \underbrace{0\cdots0}_{2N} \end{pmatrix}$

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

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

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

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

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

8:   $j = c_t^i$

9:   if landmark j never seen before

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

11:   endif

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

13:   $q = \delta^T\delta$

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

15:   $F_{x,j} = \begin{pmatrix} 1 & 0 & 0 & 0\cdots0 & 0 & 0 & 0 & 0\cdots0 \\ 0 & 1 & 0 & 0\cdots0 & 0 & 0 & 0 & 0\cdots0 \\ 0 & 0 & 1 & 0\cdots0 & 0 & 0 & 0 & 0\cdots0 \\ 0 & 0 & 0 & 0\cdots0 & 1 & 0 & 0 & 0\cdots0 \\ 0 & 0 & 0 & 0\cdots0 & 0 & 1 & 0 & 0\cdots0 \\ 0 & 0 & 0 & \underbrace{0\cdots0}_{2j-2} & 0 & 0 & 1 & \underbrace{0\cdots0}_{2N-2j} \end{pmatrix}$

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

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

18: endfor

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

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

21: return μ_t, Σ_t

Table 10.1 The extended Kalman filter (EKF) algorithm for the simultaneous localization and mapping problem, formulated here for a feature-based map and a robot equipped with sensors for measuring range and bearing. This version assumes knowledge of the exact correspondences.

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

$$y_t = \begin{pmatrix} x_t \\ m \end{pmatrix} \tag{10.7}$$

$$= \begin{pmatrix} x & y & \theta & m_{1,x} & m_{1,y} & s_1 & m_{2,x} & m_{2,y} & s_2 & \ldots & m_{N,x} & m_{N,y} & s_N \end{pmatrix}^T$$

Here x, y, and θ denote the robot's coordinates at time t, $m_{i,x}, m_{i,y}$ are the coordinates of the i-th landmark, for $i = 1, \ldots, N$, 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 7.5, which introduced the EKF localization algorithm. EKF SLAM calculates the online posterior

$$p(y_t \mid z_{1:t}, u_{1:t}) \tag{10.8}$$

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. In particular, 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 18 iterate through all measurements. The test 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 non-sparse, that is, information is propagated through the entire state estimate. The final filter update then occurs in Lines 19 and 20, 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
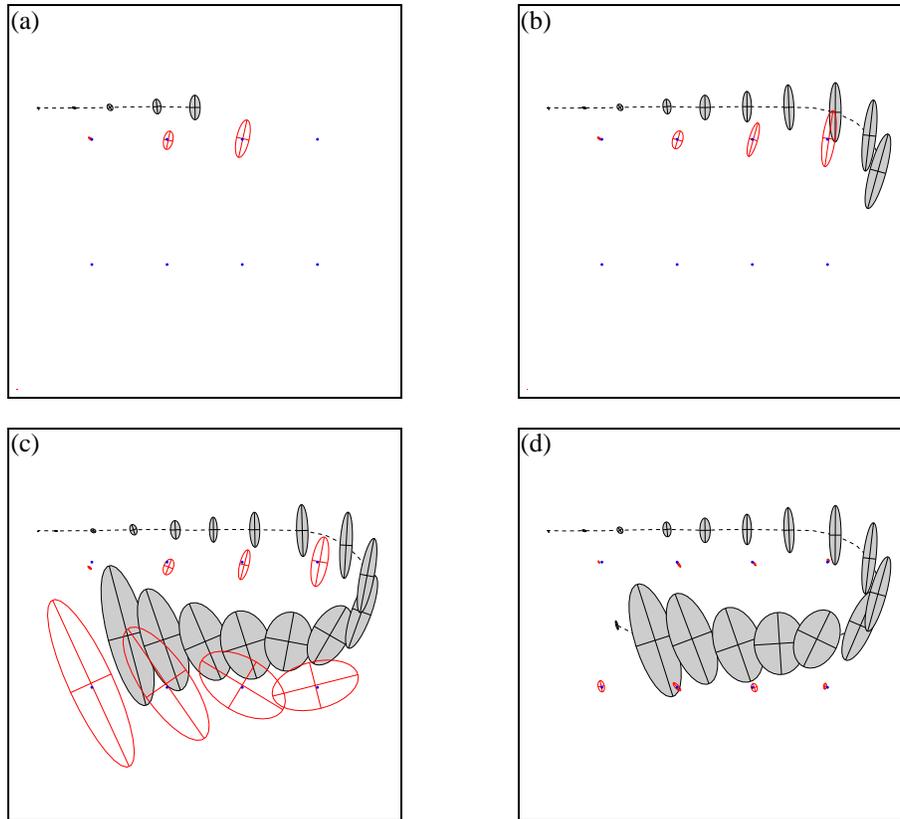
Figure 10.1 EKF applied to the online SLAM problem. The robot's path is a dotted line, and its estimations of its own position are shaded ellipses. Eight distinguishable landmarks of unknown location are shown as small dots, and their location estimations are shown as white ellipses. In (a)–(c) the robot's 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.

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 10.1 illustrates the EKF SLAM algorithm for an artificial example. The robot navigates from a start pose which 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 10.1d: 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 10.1d—notice the very small error ellipse for the final robot pose! Further, 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 causes by the robot pose, and this very uncertainty persists over time, the location estimates of those landmarks are 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

The derivation of the EKF SLAM algorithm for the case with known correspondences largely parallels that of the EKF localizer in Chapter 7.5. 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 to 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:

$$\mu_0 \;=\; \left(\; 0 \;\; 0 \;\; 0 \;\; \dots \;\; 0 \;\right)^T \tag{10.9}$$

$$\Sigma_0 \;=\; \begin{pmatrix} 0 & 0 & 0 & \infty & \cdots & \infty \\ 0 & 0 & 0 & \infty & \cdots & \infty \\ 0 & 0 & 0 & \infty & \cdots & \infty \\ \infty & \infty & \infty & \infty & \cdots & \infty \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \infty & \infty & \infty & \infty & \cdots & \infty \end{pmatrix} \tag{10.10}$$

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:

$$
y_t \;=\; y_{t-1} + \begin{pmatrix} -\frac{v_t}{\omega_t}\sin\theta + \frac{v_t}{\omega_t}\sin(\theta + \omega_t\Delta t) \\ \frac{v_t}{\omega_t}\cos\theta - \frac{v_t}{\omega_t}\cos(\theta + \omega_t\Delta t) \\ \omega_t\Delta t + \gamma_t\Delta t \\ 0 \\ \vdots \\ 0 \end{pmatrix} \tag{10.11}
$$

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:

$$
y_t \;=\; y_{t-1} + F_x \begin{pmatrix} -\frac{v_t}{\omega_t}\sin\theta + \frac{v_t}{\omega_t}\sin(\theta + \omega_t\Delta t) \\ \frac{v_t}{\omega_t}\cos\theta - \frac{v_t}{\omega_t}\cos(\theta + \omega_t\Delta t) \\ \omega_t\Delta t + \gamma_t\Delta t \end{pmatrix} \tag{10.12}
$$

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

$$
F_x = \begin{pmatrix} 1 & 0 & 0 & 0 \cdots 0 \\ 0 & 1 & 0 & 0 \cdots 0 \\ 0 & 0 & 1 & \underbrace{0 \cdots 0}_{3N \text{ columns}} \end{pmatrix} \tag{10.13}
$$

The full motion model with noise is then as follows

$$
y_t \;=\; \underbrace{y_{t-1} + F_x^T \begin{pmatrix} -\frac{v_t}{\omega_t}\sin\theta + \frac{v_t}{\omega_t}\sin(\theta + \omega_t\Delta t) \\ \frac{v_t}{\omega_t}\cos\theta - \frac{v_t}{\omega_t}\cos(\theta + \omega_t\Delta t) \\ \omega\Delta t \end{pmatrix}}_{g(u_t, y_{t-1})} + \mathcal{N}(0, F_x^T \, R_t \, F_x) \tag{10.14}
$$

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

$$g(u_t, y_{t-1}) \quad \approx \quad g(u_t, \mu_{t-1}) \; + \; G_t \, (y_{t-1} - \mu_{t-1}) \tag{10.15}$$

where the Jacobian $G_t = g'(u_t, \mu_{t-1})$ is the derivative of g at u_t, as in Equation (7.8). Obviously, the additive form in (10.14) 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:

$$G_t \quad = \quad I + F_x^T \, g_t \, F_x \tag{10.16}$$

with

$$g_t \quad = \quad \begin{pmatrix} 0 & 0 & \frac{v_t}{\omega_t} \cos \mu_{t-1,\theta} - \frac{v_t}{\omega_t} \cos(\mu_{t-1,\theta} + \omega_t \Delta t) \\ 0 & 0 & \frac{v_t}{\omega_t} \sin \mu_{t-1,\theta} - \frac{v_t}{\omega_t} \sin(\mu_{t-1,\theta} + \omega_t \Delta t) \\ 0 & 0 & 0 \end{pmatrix} \tag{10.17}$$

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 Section 7.5. In particular, we are given the following measurement model

$$z_t^i \quad = \quad \underbrace{\begin{pmatrix} \sqrt{(m_{j,x} - x)^2 + (m_{j,y} - y)^2} \\ \mathrm{atan2}(m_{j,y} - y, m_{j,x} - x) - \theta \\ m_{j,s} \end{pmatrix}}_{h(y_t, j)} + \mathcal{N}(0, \underbrace{\begin{pmatrix} \sigma_r & 0 & 0 \\ 0 & \sigma_\phi & 0 \\ 0 & 0 & \sigma_s \end{pmatrix}}_{Q_t}) \tag{10.18}$$

where 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. This

expression is approximated by the linear function

$$h(y_t, j) \approx h(\bar{\mu}_t, j) + H^i_t (y_t - \bar{\mu}_t) \tag{10.19}$$

Here H^i_t 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^i_t and a matrix $F_{x,j}$, which maps h^i_t into a matrix of the dimension of the full state vector:

$$H^i_t = h^i_t F_{x,j} \tag{10.20}$$

Here h^i_t 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:

$$h^i_t = \begin{pmatrix} \dfrac{m_{j,x} - \bar{\mu}_{t,x}}{\sqrt{q_t}} & \dfrac{y_t - \bar{\mu}_{t,y}}{\sqrt{q_t}} & 0 & \dfrac{\bar{\mu}_{t,x} - m_{j,x}}{\sqrt{q_t}} & \dfrac{\bar{\mu}_{t,y} - y_t}{\sqrt{q_t}} & 0 \\[2mm] \dfrac{\bar{\mu}_{t,y} - y_t}{q_t} & \dfrac{m_{j,x} - \bar{\mu}_{t,x}}{q_t} & -1 & \dfrac{y_t - \bar{\mu}_{t,y}}{q_t} & \dfrac{\bar{\mu}_{t,x} - m_{j,x}}{q_t} & 0 \\[2mm] 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{10.21}$$

The scalar $q_t = (m_{j,x} - \bar{\mu}_{t,x})^2 + (m_{j,y} - \bar{\mu}_{t,y})^2$, and as before, $j = c^i_t$ is the landmark that corresponds to the measurement z^i_t. The matrix $F_{x,j}$ is of dimension $(3N+3) \times 5$. It maps the low-dimensional matrix h^i_t into a matrix of dimension $(3N + 3) \times 3$:

$$F_{x,j} = \begin{pmatrix} 1 & 0 & 0 & 0 \cdots 0 & 0 & 0 & 0 & 0 \cdots 0 \\ 0 & 1 & 0 & 0 \cdots 0 & 0 & 0 & 0 & 0 \cdots 0 \\ 0 & 0 & 1 & 0 \cdots 0 & 0 & 0 & 0 & 0 \cdots 0 \\ 0 & 0 & 0 & 0 \cdots 0 & 1 & 0 & 0 & 0 \cdots 0 \\ 0 & 0 & 0 & 0 \cdots 0 & 0 & 1 & 0 & 0 \cdots 0 \\ 0 & 0 & 0 & \underbrace{0 \cdots 0}_{2j-2} & 0 & 0 & 1 & \underbrace{0 \cdots 0}_{2N-2j} \end{pmatrix} \tag{10.22}$$

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 (10.9) leads to a poor linearization. This is because with the default initialization in (10.9), 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 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

$$
\begin{pmatrix} \bar{\mu}_{j,x} \\ \bar{\mu}_{j,y} \\ \bar{\mu}_{j,s} \end{pmatrix} = \begin{pmatrix} \bar{\mu}_{t,x} \\ \bar{\mu}_{t,y} \\ s_t^i \end{pmatrix} + r_t^i \begin{pmatrix} \cos(\phi_t^i + \bar{\mu}_{t,\theta}) \\ \sin(\phi_t^i + \bar{\mu}_{t,\theta}) \\ 0 \end{pmatrix} \tag{10.23}
$$

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 to and apply triangulation to determine an appropriate initial location estimate.

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.

10.3   EKF SLAM WITH UNKNOWN CORRESPONDENCES

10.3.1   The General EKF SLAM Algorithm

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. The input to this algorithm now lacks a correspondence variable c_t. Instead, it includes the momentary size of the map, N_{t-1}.

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

2:   $N_t = N_{t-1}$

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

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

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

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

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

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

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

10:   for $k = 1$ to N_t+1 do

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

12:   $q_k = \delta_k^T \delta_k$

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

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

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

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

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

18:   endfor

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

20:   $j(i) = \underset{k}{\mathrm{argmin}}\ \pi_k$

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

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

23:   endfor

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

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

26:   return μ_t, Σ_t

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

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 24 and 25. 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 7.6; in particular, the algorithm **EKF_localization** in Table 7.3 on page 175 used an analogous equation do determine the most likely landmark. (Line 14). The measurement updates in Lines 24 and 25 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 speed.
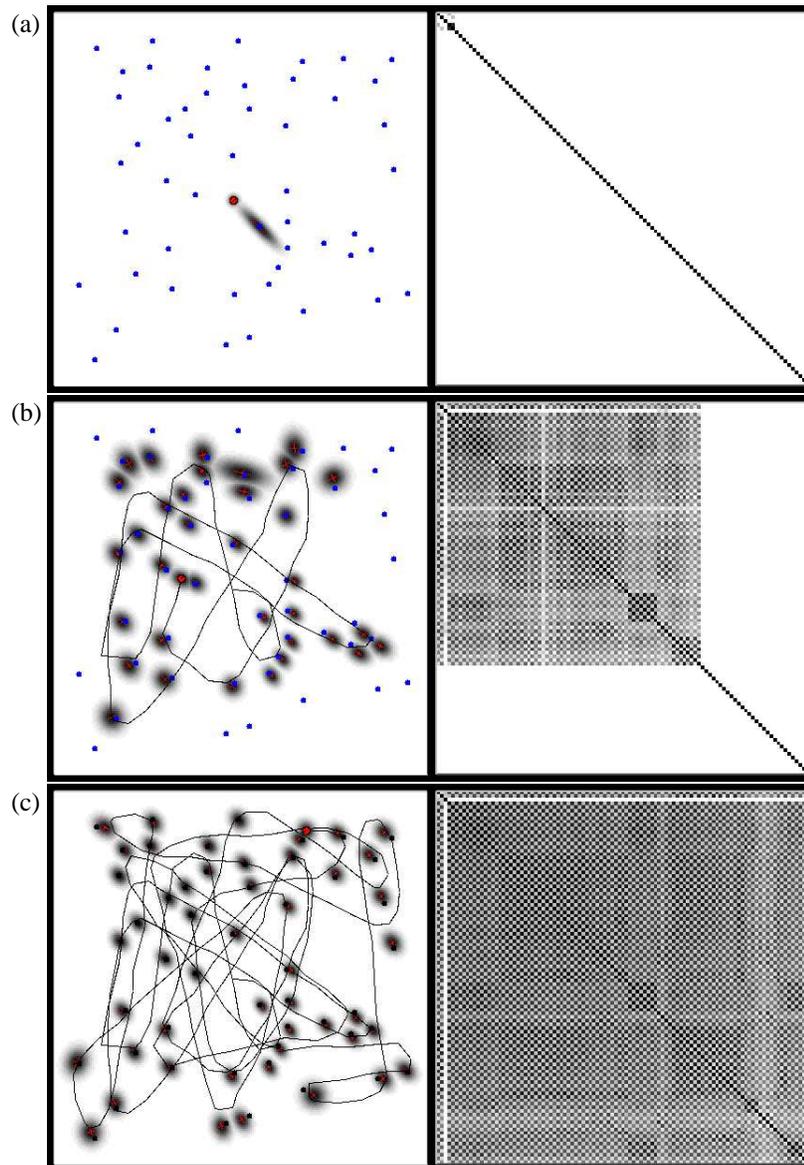
Figure 10.2 EKF SLAM with known data association in a simulated environment. The map is shown on the left, with the gray-level corresponding to the uncertainty of each landmark. The matrix on the right is the correlation matrix, which is the normalized covariance matrix of the posterior estimate. After some time, all x- and all y-coordinate estimates become fully correlated.
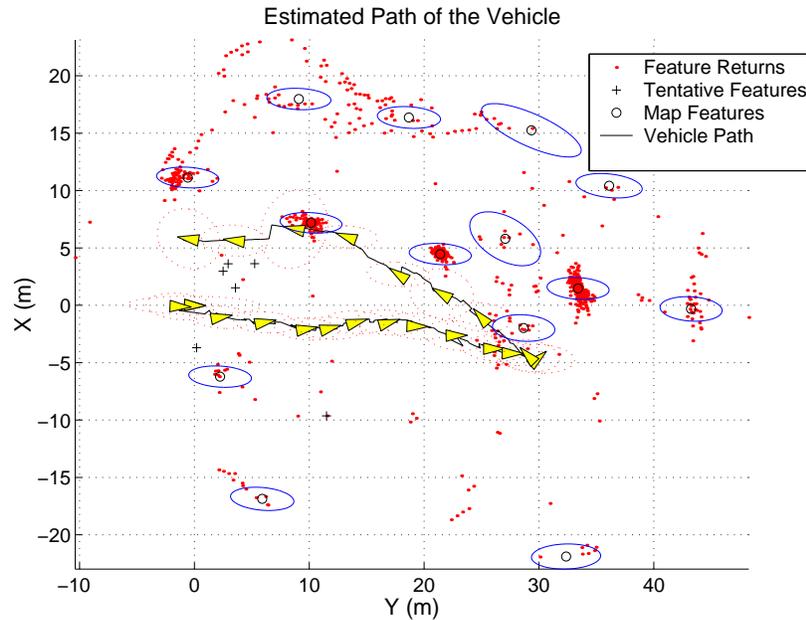
Figure 10.3 Example of Kalman filter estimation of the map and the vehicle pose. Image courtesy of Stefan Williams and Hugh Durrant-Whyte from the Australian Centre for Field Robotics, University of Sydney, Australia.

10.3.2   Examples

Figure 10.2 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 10.2c, 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. Fig-

Figure 10.4 Underwater vehicle Oberon, developed at the University of Sydney. Image courtesy of Stefan Williams and Hugh Durrant-Whyte from the Australian Centre for Field Robotics, University of Sydney, Australia.

ure 10.3 shows an example result obtained using the underwater robot Oberon, developed at the University of Sydney, Australia, and shown in Figure 10.4. This vehicle is equipped with a pencil sonar, a sonar that can scan at very high resolutions an 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 10.3, the robot moves by these landmarks, then turns around and moves back. While doing so, it measures and integrates detected landmarks into its map. using the Kalman filter approach described in this chapter.

The map shown in Figure 10.3 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 10.3 show landmark sightings, obtained by searching the sonar scan for small and highly reflective objects. The majority of these sightings is rejected, using a mechanism described further below, in the section that follows. However, some are believed to correspond to a landmark and 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 10.3. 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. In this example, the robot successfully finds and maps all of the artificial landmarks in this highly noisy domain.

10.3.3 Feature Selection and Map Management

Making EKF SLAM robust in practice often requires additional techniques for managing maps. 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.

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 compensate 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.

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 in state-of-the-art implementations, is to maintain a posterior likelihood of the existence of a landmark. Such a probability may implemented as log-odds ratio and be denoted o_j for the j-th landmark in the map. Whenever the j-th landmark is m_j 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.

As noted previously, the maximum likelihood approach to data association has a clear limitation. This limitation arises from the fact that the maximum likelihood approach

deviates from the idea of full posterior estimation in probabilistic robotic. 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 to that the probability of confusing one with another is negligible. 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 landmarks.

- **Signatures.** When selecting appropriate landmarks, it is essential to 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, in the deep see, and in abandoned mines.

A key limitation of EKF SLAM lies in the necessity to select appropriate landmarks. By doing so, most of the sensor data is usually discarded. 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 to recognize 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 chapter, and capable of handling orders of magnitude more features. This culminates into the principal dilemma of the EKF SLAM algorithm: While incremental maxi-

mum 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 large maps. We will also discuss more robust data association techniques. For its many limitation, the value of the EKF SLAM algorithm presented in this chapter is mostly historical.

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 problem 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 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 drawbacks are 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 technique. This technique makes is impossible to revise part 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

Place them here!

10.6   PROJECTS

1. Develop an incremental algorithm for posterior estimation of poses and maps (with known data association) that does not rely on linearizing the motion model and the perceptual model. Our suggestion: Replace the Kalman filter by particle filters.

2. The basic Kalman filter approach is unable to cope with the data association problem in a sound statistical way. Develop an algorithm (and a statistical framework) for posterior estimation with unknown data association, and characterize its advantages and disadvantages. We suggest to use a mixture of Gaussians representation.

3. 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).

4. Develop a Kalman filter algorithm which uses local occupancy grid maps as its basic components, instead of landmarks. Among other things, problems 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.

5. Develop and algorithm that learns its own features for Kalman filter mapping. There could be two different variants: One which chooses features so as to minimize the uncertainty (entropy) in the posterior, another which is given access to the correct map and seeks to maximize the probability of the correct map under the posterior.

11

THE EXTENDED INFORMATION
FORM ALGORITHM

11.1   INTRODUCTION

The EKF SLAM algorithm described in the previous chapter is subject to a number of limitations, as discussed there. In this chapter, we introduce an alternative SLAM algorithm, called the *extended information form SLAM algorithm*, of *EIF SLAM*. EIF SLAM has in common with the EKF that it represents the posterior by a Gaussian. Unlike EKF SLAM, which solves the online SLAM problem, EIF SLAM solves the full SLAM problem. The reader may recall that the online SLAM posterior is defined over the map and the most recent robot pose, whereas the full SLAM posterior is defined over the map and the entire robot path. Another key difference is that EIF SLAM represents the posterior Gaussian in its information form, that is, via the information (or precision) matrix and the information state vector. Clearly, the information and the moments representation carry the same information; however, as we shall see, the update mechanisms of the information form are substantially different from that of the EKF.

EIF SLAM is not an incremental algorithm; this is because it calculates posteriors over a robot path. Instead, EIF SLAM processes an entire data set at a time, to generate the full SLAM posterior. This approach is fundamentally different from EKF SLAM, which is incremental and enables a robot to update its map forever. EIF SLAM is best suited to problems where one seeks a map from a data set of fixed size, and can afford to hold the data in memory up to the time where the map is built.

Because EIF SLAM has all data available during mapping, it can apply improved linearization and data association techniques. Whereas 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 EIF SLAM *all* data can be used to linearize and to calculate correspon-

dences. Further, EIF SLAM iterates the construction of the map, the calculation of correspondence variables, and the linearization of the measurement models and motion models, so as to obtain the best estimate of all of those quantities. In doing so, EIF SLAM tends to produce maps that are superior in accuracy to that of EKFs. EIF SLAM is less brittle to erroneous data association, and it can cope with higher rotational uncertainties in the data than EKF SLAM. And finally, EIF SLAM is applicable to maps many orders of magnitude larger than those that can be accommodated by the EKF.

This chapter first describe the intuition behind EIF SLAM and its basic updates steps. We then derive the various update steps mathematically and prove its correctness relative to specific linear approximations. We then discuss actual implementations, in which posteriors are calculated over maps

11.2   INTUITIVE DESCRIPTION

The basic intuition behind EIF SLAM is remarkably simple. Suppose we are given a set of measurements $z_{1:t}$ with associated correspondence variables $c_{1:t}$, and a set of controls $u_{1:t}$. The first step of EIF SLAM will be to use those data to construct an information matrix and an information vector defined over the joint space of robot poses $x_{1:t}$ and the map $m = \{m_j\}$. 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. 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.

Make Figure: a robot path, landmarks, and an information matrix.

Figure **??** illustrates the process of constructing the information matrix. Each control u_t provides information about the relative value of the the robot pose at time $t - 1$ and the pose at time t. We can think of this information as a constraint between x_{t-1} and x_t, or a "spring" in a spring-mass model of the world. The control u_t is incorporated into Ω and ξ by adding a values between the rows and columns connecting x_{t-1} and x_t. The magnitude of these values corresponds to the stiffness of the constraint—or the residual uncertainty between the relative poses caused by the noise in the motion model. This is illustrated in Figure **??**, which shows the link between two robot poses, and the corresponding element in the information matrix.

Similarly, consider a measurement z_t^i. This measurement provides information between the location of the landmark $j = c_t^i$ and the robot pose x_t at time t. Again, this information can be thought of as a constraint. The respective update in the EIF amounts to adding values between the elements linking the pose x_t and the map feature m_j. As before, the magnitude of these values reflects the residual uncertainty 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 an information matrix Ω whose 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 landmarks 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 landmark relative to a robot pose. Thus, the information matrix is sparse; all but a linear number of elements are zero.

Of course, the information representation does not give us a map; neither does it tell us the estimated path of the robot. Both obtained via $\mu = \Omega^{-1}\xi$ (see Equation (3.72) on page 56). This operation involves the inversion of a sparse matrix. This raises the question on how efficiently we can recover the map, and the posterior Σ. It is the central question in EIF SLAM: EIF have been applied to maps with hundreds of millions of features, and inverting a matrix of this size can be a challenge.

The answer to the complexity question depends on the topology of the world. If each feature is seen only once, the graph represented by the constraints in Ω is linear. Thus, Ω can be reordered so that it becomes a band-diagonal matrix, that is, all non-zero values occur near its diagonal. The standard variable elimination technique for matrix inversion will then invert Ω in linear time, by a single sweep through Ω. 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.

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, and the robot traverses a full cycle. 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 linear matrix inversion trick

inapplicable, and matrix inversion becomes more complex. Since most worlds possess cycles, this is the case of interest.

The EIF SLAM algorithm now employs an important factorization trick, which we can think of as propagating information trough 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. More 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 landmark 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 that this operation is local: It only involves only 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 spring remains equivalent, with the only exception that m_j is now disconnected.

Make Figure: reducing the graph by shifting edges around

The virtue of this reduction step is that we can gradually transform our optimization 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 pose 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, so the remaining matrix inversion problem may still require more than linear time.

As a last step, EIF SLAM recovers the landmark locations. Conceptually, this is achieved by building a new information matrices Ω_j and information vectors ξ_j for

---

1:         **Algorithm EIF_initialize($u_{1:t}$):**

2:
$$\begin{pmatrix} \mu_{0,x} \\ \mu_{0,y} \\ \mu_{0,\theta} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

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

4:
$$\begin{pmatrix} \mu_{t,x} \\ \mu_{t,y} \\ \mu_{t,\theta} \end{pmatrix} = \begin{pmatrix} \mu_{t-1,x} \\ \mu_{t-1,y} \\ \mu_{t-1,\theta} \end{pmatrix} + \begin{pmatrix} -\frac{v_t}{\omega_t}\sin\mu_{t-1,\theta} + \frac{v_t}{\omega_t}\sin(\mu_{t-1,\theta}+\omega_t\Delta t) \\ \frac{v_t}{\omega_t}\cos\mu_{t-1,\theta} - \frac{v_t}{\omega_t}\cos(\mu_{t-1,\theta}+\omega_t\Delta t) \\ \omega_t\Delta t \end{pmatrix}$$

5:         *endfor*

6:         *return $\mu_{0:t}$*

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

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 very 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 information representation is such a natural representation. The full SLAM problem is solved by locally adding information into a large matrix, for each measurement z_t^i and each control u_t. To turn information into an estimate of the map and the robot path, 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 EIF SLAM ALGORITHM

We will now make the various computational step of the EIF SLAM precise. The full EIF SLAM algorithm will be described in a number of steps. The main difficulty in implementing the simple additive information algorithm pertains to the conversion of

1:      **Algorithm EIF_construct**$(u_{1:t}, z_{1:t}, c_{1:t}, \mu_{0:t})$**:**
2:          *set* $\Omega = 0, \xi = 0$
3:          *add* $\begin{pmatrix} \infty & 0 & 0 \\ 0 & \infty & 0 \\ 0 & 0 & \infty \end{pmatrix}$ *to* Ω *at* x_0

4:          *for all controls* $u_t = (v_t \;\; \omega_t)^T$ *do*
5:              $\hat{x}_t = \mu_{t-1} + \begin{pmatrix} -\frac{v_t}{\omega_t}\sin\mu_{t-1,\theta} + \frac{v_t}{\omega_t}\sin(\mu_{t-1,\theta} + \omega_t\Delta t) \\ \frac{v_t}{\omega_t}\cos\mu_{t-1,\theta} - \frac{v_t}{\omega_t}\cos(\mu_{t-1,\theta} + \omega_t\Delta t) \\ \omega_t\Delta t \end{pmatrix}$
6:              $G_t = \begin{pmatrix} 1 & 0 & \frac{v_t}{\omega_t}\cos\mu_{t-1,\theta} - \frac{v_t}{\omega_t}\cos(\mu_{t-1,\theta} + \omega_t\Delta t) \\ 0 & 1 & \frac{v_t}{\omega_t}\sin\mu_{t-1,\theta} - \frac{v_t}{\omega_t}\sin(\mu_{t-1,\theta} + \omega_t\Delta t) \\ 0 & 0 & 1 \end{pmatrix}$
7:              *add* $\begin{pmatrix} 1 \\ -G_t \end{pmatrix} R_t^{-1} (1 \;\; -G_t)$ *to* Ω *at* x_t *and* x_{t-1}
8:              *add* $\begin{pmatrix} 1 \\ -G_t \end{pmatrix} R_t^{-1} [\hat{x}_t + G_t\,\mu_{t-1}]$ *to* ξ *at* x_t *and* x_{t-1}
9:          *endfor*

10:         *for all measurements* z_t *do*
11:             $Q_t = \begin{pmatrix} \sigma_r & 0 & 0 \\ 0 & \sigma_\phi & 0 \\ 0 & 0 & \sigma_s \end{pmatrix}$
12:             *for all observed features* $z_t^i = (r_t^i \;\; \phi_t^i \;\; s_t^i)^T$ *do*
13:                 $j = c_t^i$
14:                 $\delta = \begin{pmatrix} \delta_x \\ \delta_y \end{pmatrix} = \begin{pmatrix} \mu_{j,x} - \mu_{t,x} \\ \mu_{j,y} - \mu_{t,y} \end{pmatrix}$
15:                 $q = \delta^T\delta$
16:                 $\hat{z}_t^i = \begin{pmatrix} \sqrt{q} \\ \text{atan2}(\delta_y, \delta_x) - \mu_{t,\theta} \end{pmatrix}$
17:                 $H_t^i = \frac{1}{q}\begin{pmatrix} \sqrt{q}\delta_x & -\sqrt{q}\delta_y & 0 & -\sqrt{q}\delta_x & \sqrt{q}\delta_y \\ \delta_y & \delta_x & -1 & -\delta_y & -\delta_x \end{pmatrix}$
18:                 *add* $H_t^{iT} Q_t^{-1} H_t^i$ *to* Ω *at* x_t *and* m_j
19:                 *add* $H_t^{iT} Q_t^{-1} [\, z_t^i - \hat{z}_t^i - H_t^i \begin{pmatrix} \mu_{t,x} \\ \mu_{t,y} \\ \mu_{t,\theta} \\ \mu_{j,x} \\ \mu_{j,y} \end{pmatrix}]$ *to* ξ *at* x_t *and* m_j
20:             *endfor*
21:         *endfor*
22:         *return* Ω, ξ

Table 11.2   Calculation of Ω and ξ in the EIF algorithm.

1:          **Algorithm EIF_reduce(Ω, ξ):**
2:            $\tilde{\Omega} = \Omega$
3:            $\tilde{\xi} = \xi$
4:            *for each feature j do*
5:               *let $\tau(j)$ be the set of all poses x_t at which feature j was observed*
6:               *subtract $\tilde{\Omega}_{\tau(j),j} \, \tilde{\Omega}_{j,j}^{-1} \, \xi_j$ from $\tilde{\xi}$ at $x_{\tau(j)}$ and m_j*
7:               *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*
8:               *remove the rows and columns corresponding to feature j from $\tilde{\Omega}$ and $\tilde{\xi}$*
9:            *endfor*
10:          *return $\tilde{\Omega}, \tilde{\xi}$*

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

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 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 **EIF_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 $\mu_{0:t}$, **EIF_initialize** only use 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 EIF SLAM algorithm constructs the full SLAM information matrix Ω and the corresponding information vector ξ. This is achieved by the algorithm **EIF_construct**, depicted in Table 11.2. This algorithm contains a good amount of mathematical notation, much of which shall become clear in our mathematical derivation of the algorithm further below. **EIF_construct** accepts as an input the

1:          **Algorithm EIF_solve($\tilde{\Omega}, \tilde{\xi}, \Omega, \xi$):**
2:               $\Sigma_{0:t} = \tilde{\Omega}^{-1}$
3:               $\mu_{0:t} = \Sigma_{0:t} \, \tilde{\xi}$
4:               *for each feature j do*
5:                    *let $\tau(j)$ be the set of all poses x_t at which feature j was observed*
6:                    $\mu_j = \Omega_{j,j}^{-1} \, (\xi_j + \Omega_{j,\tau(j)} \, \tilde{\mu}_{\tau(j)})$
7:               *endfor*
8:               *return $\mu, \Sigma_{0:t}$*

Table 11.4 Algorithm for updating the posterior μ.

set of controls, $u_{1:t}$, the measurements $z_{1:t}$ and associated correspondence 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 by locally adding submatrices in accordance with the information obtained from each measurement and each control. In particular, Line 2 in **EIF_construct** 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 **EIF_construct**. The pose $\hat{x}$ and the Jacobian G_t calculated in Lines 5 and 6 represent the linear approximation of the non-linear measurement 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 **EIF_construct**. 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. This calculation culminates into the computation of the measurement update in Lines 18 and 19. The matrix that is being address to Ω in Line 18 is of dimension 5×5. To add it, we decomposes it into a matrix of dimension 3×3 for the pose x_t, a matrix of dimension 2×2 for the feature m_j, and two matrices of dimension 3×2 and 2×3 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 **EIF_construct** is an information vector ξ and a matrix Ω. Ω is sparse, containing only non-zero submatrices along the main diagonal, between subsequent poses, and between poses and features in the map. If the number map features outnumber the number of poses by a factor of 10, some 99% of the elements of Ω are zero. 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 EIF SLAM algorithm pertains to reducing the dimensionality of the information matrix/vector. This is achieved through the algorithm **EIF_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. This transformation is achieved by removing features m_j one-at-a-time, in Lines 4 through 9 of **EIF_reduce**. The book keeping 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 $\tilde{\Omega}$: $\tilde{\Omega}_{j,j}$, which is the quadratic submatrix between m_j and m_j, and $\tilde{\Omega}_{\tau(j),j}$, which 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 **EIF_reduce** is once again linear in t.

The last step in the EIF SLAM 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 **EIF_solve** in Table 11.4. Lines 2 and 3 compute the path estimates $\mu_{0:t}$, by inverting thee reduced information matrix $\tilde{\Omega}$ and multiplying the resulting covariance with the information vector. When $\tilde{\Omega}$ involves cycles, the inversion becomes the computationally most expensive step in the EIF SLAM algorithm. Subsequently, **EIF_solve** computes the location of each feature in Lines 4 through 7. The return value of **EIF_solve** contains the mean for the robot path and all features in the map, but only the covariance for the robot path.

The quality of the solution calculated by the EIF SLAM algorithm depends on the goodness of the initial mean estimates, calculated by **EIF_initialize**. The x- and y-components of these estimates effect the respective models in a linear way, hence the

1:      **Algorithm EIF_SLAM_known_correspondence**($u_{1:t}, z_{1:t}, c_{1:t}$)**:**
2:          $\mu_{0:t} = $ **EIF_initialize**($u_{1:t}$)
3:          *repeat*
4:              $\Omega, \xi = $ **EIF_construct**($u_{1:t}, z_{1:t}, c_{1:t}, \mu_{0:t}$)
5:              $\tilde{\Omega}, \tilde{\xi} = $ **EIF_reduce**(Ω, ξ)
6:              $\mu, \Sigma_{0:t} = $ **EIF_solve**($\tilde{\Omega}, \tilde{\xi}, \Omega, \xi$)
7:          *until convergence*
8:          *return* μ

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

linearization does not depend on they value. 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 accommodate errors in the initial $\mu_{0:t}$, the procedures **EIF_construct**, **EIF_reduce**, and **EIF_solve** are run multiple times over the same data set. Each iteration takes as an input a estimated mean vector $\mu_{0:t}$ for all poses, and outputs a new, improves estimate. The iteration of the EIF SLAM 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 comprises the best estimates of the robot's path and the map.

11.4   MATHEMATICAL DERIVATION

The derivation of the EIF SLAM 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:

$$y_{0:t} = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_t \\ m \end{pmatrix} \quad \text{and} \quad y_t = \begin{pmatrix} x_t \\ m \end{pmatrix} \tag{11.1}$$

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:

$$p(y_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t})$$
$$= \eta \, p(z_t \mid y_{0:t}, z_{1:t-1}, u_{1:t}, c_{1:t}) \, p(y_{0:t} \mid z_{1:t-1}, u_{1:t}, c_{1:t}) \tag{11.2}$$

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

$$p(z_t \mid y_{0:t}, z_{1:t-1}, u_{1:t}, c_{1:t}) = p(z_t \mid y_t, c_t) \tag{11.3}$$

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

$$p(y_{0:t} \mid z_{1:t-1}, u_{1:t}, c_{1:t})$$
$$= p(x_t \mid y_{1:t-1}, z_{1:t-1}, u_{1:t}, c_{1:t}) \, p(y_{1:t-1} \mid z_{1:t-1}, u_{1:t}, c_{1:t})$$
$$= p(x_t \mid x_{t-1}, u_t) \, p(y_{1:t-1} \mid z_{1:t-1}, u_{1:t-1}, c_{1:t-1}) \tag{11.4}$$

Putting these expressions back into (11.2) gives us the recursive definition of the full SLAM posterior:

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

$$= \eta \, p(z_t \mid y_t, c_t) \, p(x_t \mid x_{t-1}, u_t) \, p(y_{1:t-1} \mid z_{1:t-1}, u_{1:t-1}, c_{1:t-1}) \quad (11.5)$$

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.

$$
\begin{aligned}
p(y_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) &= \eta \, p(y_0) \prod_t p(z_t \mid y_t, c_t) \, p(x_t \mid x_{t-1}, u_t) \quad (11.6) \\
&= \eta \, p(y_0) \prod_t \left[p(x_t \mid x_{t-1}, u_t) \prod_i p(z_t^i \mid y_t, c_t^i) \right]
\end{aligned}
$$

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 have no initial knowledge about the map m. This allows us to replace $p(y_0)$ by $p(x_0)$, subsuming the factor $p(m)$ into the normalizer η.

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

$$
\begin{aligned}
&\log p(y_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) \quad (11.7) \\
&= \text{const.} + \log p(x_0) \sum_t \left[\log p(x_t \mid x_{t-1}, u_t) \sum_i \log p(z_t^i \mid y_t, c_t^i) \right]
\end{aligned}
$$

This posterior has a simple form: It is a sum of terms. These terms include a prior, one term for each control u_t, and one term for each measurement z_t^i. As we shall see, EIF SLAM simply replaces those additive terms with quadratic constraints.

11.4.2 Taylor Expansion

The key approximation of EIF SLAM is the same as in the EKF: The motion and measurement models are approximated using linear functions with Gaussian error distributions. 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

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

$$p(z_t^i \mid y_t, c_t^i) \quad = \quad \eta \ \exp\left\{-\tfrac{1}{2}(z_t^i - h(y_t, c_t^i))^T \ Q_t^{-1} \ (z_t^i - h(y_t, c_t^i))\right\} \ (11.8)$$

The prior $p(x_0)$ in (11.7) is also easily expressed by a Gaussian-type distribution. This prior anchors the initial pose x_0 to the origin of the global coordinate system: $x_0 = (0 \ 0 \ 0)^T$:

$$p(x_0) \quad = \quad \eta \ \exp\left\{-\tfrac{1}{2} \ x_0^T \ \Omega_0 \ x_0\right\} \tag{11.9}$$

with

$$\Omega_0 \quad = \quad \begin{pmatrix} \infty & 0 & 0 \\ 0 & \infty & 0 \\ 0 & 0 & \infty \end{pmatrix} \tag{11.10}$$

It shall, at this point, 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 log-SLAM posterior in (11.7):

$$\log p(y_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t})$$
$$= \quad \text{const.} - \tfrac{1}{2} \left[x_0^T \ \Omega_0 \ x_0 + \sum_t (x_t - g(u_t, x_{t-1}))^T \ R_t^{-1} \ (x_t - g(u_t, x_{t-1})) \right.$$
$$\left. + \sum_i (z_t^i - h(y_t, c_t^i))^T \ Q_t^{-1} \ (z_t^i - h(y_t, c_t^i)) \right] \tag{11.11}$$

This representation 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.

However, these terms are quadratic in the functions g and h, not in the variables we week to estimate (poses and the map). This is alleviated by linearizing g and h via Taylor expansion, analogously to Equations (10.15) and (10.19) in the derivation of the EKF:

$$g(u_t, x_{t-1}) \quad \approx \quad g(u_t, \mu_{t-1}) + G_t(x_{t-1} - \mu_{t-1})$$
$$h(y_t, c_t^i) \quad \approx \quad h(\mu_t, c_t^i) + H_t^i \ (y_t - \mu_t) \tag{11.12}$$

Where 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 (10.20).

11.4.3   Constructing the Information Form

This linear approximation turns the log-likelihood (11.11) into a function that is quadratic in $y_{0:t}$. In particular, we obtain

$$\log p(y_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) = \text{const.} - \tfrac{1}{2}$$

$$\Big\{ x_0^T\ \Omega_0\ x_0 + \sum_t [x_t - g(u_t, \mu_{t-1}) - G_t(x_{t-1} - \mu_{t-1})]^T$$

$$R_t^{-1}\ [x_t - g(u_t, \mu_{t-1}) - G_t(x_{t-1} - \mu_{t-1})] \tag{11.13}$$

$$+ \sum_i [z_t^i - h(\mu_t, c_t^i) - H_t^i(y_t - \mu_t)]^T\ Q_t^{-1}\ [z_t^i - h(\mu_t, c_t^i) - H_t^i(y_t - \mu_t)] \Big\}$$

This function is indeed a quadratic, but it shall prove convenient to reorder its terms.

$$\log p(y_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) = \text{const.}$$

$$-\tfrac{1}{2}\ \underbrace{x_0^T\ \Omega_0\ x_0}_{\text{quadratic in } x_0}\ -\tfrac{1}{2}\sum_t \underbrace{x_{t-1:t}^T \begin{pmatrix} 1 \\ -G_t \end{pmatrix} R_t^{-1}\ (1\ -G_t)\ x_{t-1:t}}_{\text{quadratic in } x_{t-1:t}}$$

$$+ \underbrace{x_{t-1:t}^T \begin{pmatrix} 1 \\ -G_t \end{pmatrix} R_t^{-1}\ [g(u_t, \mu_{t-1}) + G_t\ \mu_{t-1}]}_{\text{linear in } x_{t-1:t}} \tag{11.14}$$

$$-\tfrac{1}{2}\sum_i \underbrace{y_t^T\ H_t^{iT}\ Q_t^{-1}\ H_t^i\ y_t}_{\text{quadratic in } y_t} + \underbrace{y_t^T\ H_t^{iT}\ Q_t^{-1}\ [z_t^i - h(\mu_t, c_t^i) - H_t^i \mu_t]}_{\text{linear in } y_t}$$

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

If we collect all quadratic terms into the matrix Ω, and all linear terms into a vector ξ, we see that expression (11.14) is of the form

$$\log p(y_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) = \text{const.} - \tfrac{1}{2}\ y_{0:t}^T\ \Omega\ y_{0:t} + y_{0:t}^T\ \xi \tag{11.15}$$

We can read off these terms directly from (11.14), and verify that they are indeed implemented in the algorithm **EIF_construct** 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

$$\Omega \quad \longleftarrow \quad \Omega_0 \tag{11.16}$$

  This initialization is performed in Lines 2 and 3 of the algorithm **EIF_construct**. **EIF_construct**.

- **Controls.** From (11.14), 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:

$$\Omega \quad \longleftarrow \quad \Omega + \begin{pmatrix} 1 \\ -G_t \end{pmatrix} R_t^{-1} \, (1 \;\; - G_t) \tag{11.17}$$

$$\xi \quad \longleftarrow \quad \xi + \begin{pmatrix} 1 \\ -G_t \end{pmatrix} R_t^{-1} \, [g(u_t, \mu_{t-1}) + G_t \, \mu_{t-1}] \tag{11.18}$$

  This is realized in Lines 4 through 9 in **EIF_construct**.

- **Measurements.** According to Equation (11.14), each measurement z_t^i transforms Ω and ξ by adding the following terms, once again assuming appropriate adjustment of the matrix dimensions:

$$\Omega \quad \longleftarrow \quad \Omega + H_t^{iT} \, Q_t^{-1} \, H_t^i \tag{11.19}$$

$$\xi \quad \longleftarrow \quad \xi + H_t^{iT} \, Q_t^{-1} \, [z_t^i - h(\mu_t, c_t^i) - H_t^i \mu_t] \tag{11.20}$$

  This update occurs in Lines 10 through 21 in **EIF_construct**.

This proves the correctness of the construction algorithm **EIF_construct**, relative to our linear Taylor expansion approximation.

We also note that the steps above only affect elements on the (generalized) main diagonal of the matrix, or 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

$$\Omega = \begin{pmatrix} \Omega_{xx} & \Omega_{xy} \\ \Omega_{yx} & \Omega_{yy} \end{pmatrix} \qquad \text{and} \qquad \xi = \begin{pmatrix} \xi_x \\ \xi_y \end{pmatrix} \qquad (11.21)$$

If Ω_{yy} is invertible, the marginal $p(x)$ is a Gaussian whose information representation is

$$\bar{\Omega}_{xx} = \Omega_{xx} - \Omega_{xy} \, \Omega_{yy}^{-1} \, \Omega_{yx} \qquad \text{and} \qquad \bar{\xi}_x = \xi_x - \Omega_{xy} \, \Omega_{yy}^{-1} \, \xi_y \qquad (11.22)$$

Proof. The marginal for a Gaussian in its moments representation

$$\Sigma = \begin{pmatrix} \Sigma_{xx} & \Sigma_{xy} \\ \Sigma_{yx} & \Sigma_{yy} \end{pmatrix} \qquad \text{and} \qquad \mu = \begin{pmatrix} \mu_x \\ \mu_y \end{pmatrix}$$

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 44. Let $P = (0 \ 1)^T$, and let $[\infty]$ be a matrix of the same size as Ω_{yy} but whose entries are all infinite (and 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:

$$(\Omega + P[\infty]P^T)^{-1}$$
$$= \Omega - \Omega \, P([\infty]^{-1} + P^T \, \Omega \, P)^{-1} \, P^T \, \Omega$$
$$= \Omega - \Omega \, P(0 + P^T \, \Omega \, P)^{-1} \, P^T \, \Omega$$
$$= \Omega - \Omega \, P(\Omega_{yy})^{-1} \, P^T \, \Omega$$
$$= \begin{pmatrix} \Omega_{xx} & \Omega_{xy} \\ \Omega_{yx} & \Omega_{yy} \end{pmatrix} - \begin{pmatrix} \Omega_{xx} & \Omega_{xy} \\ \Omega_{yx} & \Omega_{yy} \end{pmatrix} \begin{pmatrix} 0 & 0 \\ 0 & \Omega_{yy}^{-1} \end{pmatrix} \begin{pmatrix} \Omega_{xx} & \Omega_{xy} \\ \Omega_{yx} & \Omega_{yy} \end{pmatrix}$$
$$\overset{(*)}{=} \begin{pmatrix} \Omega_{xx} & \Omega_{xy} \\ \Omega_{yx} & \Omega_{yy} \end{pmatrix} - \begin{pmatrix} 0 & \Omega_{xy} \, \Omega_{yy}^{-1} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \Omega_{xx} & \Omega_{xy} \\ \Omega_{yx} & \Omega_{yy} \end{pmatrix}$$
$$= \begin{pmatrix} \Omega_{xx} & \Omega_{xy} \\ \Omega_{yx} & \Omega_{yy} \end{pmatrix} - \begin{pmatrix} \Omega_{xy} \, \Omega_{yy}^{-1} \, \Omega_{yx} & \Omega_{xy} \\ \Omega_{yx} & \Omega_{yy} \end{pmatrix} = \begin{pmatrix} \bar{\Omega}_{xx} & 0 \\ 0 & 0 \end{pmatrix}$$

The remaining statement, $\Sigma_{xx}^{-1} \mu_x = \bar{\xi}_x$, is obtained analogously, exploiting the fact that $\mu = \Omega^{-1}\xi$ (see Equation(3.72)) and the equality of the two expressions marked "$(*)$" above:

$$\begin{pmatrix} \Sigma_{xx}^{-1} \mu_x \\ 0 \end{pmatrix} = \begin{pmatrix} \Sigma_{xx}^{-1} & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \mu_x \\ \mu_y \end{pmatrix} = \begin{pmatrix} \Sigma_{xx}^{-1} & 0 \\ 0 & 0 \end{pmatrix} \Omega^{-1} \begin{pmatrix} \xi_x \\ \xi_y \end{pmatrix}$$
$$\overset{(*)}{=} \left[\Omega - \begin{pmatrix} 0 & \Omega_{xy} \, \Omega_{yy}^{-1} \\ 0 & 1 \end{pmatrix} \Omega \right] \Omega^{-1} \begin{pmatrix} \xi_x \\ \xi_y \end{pmatrix}$$
$$= \begin{pmatrix} \xi_x \\ \xi_y \end{pmatrix} - \begin{pmatrix} 0 & \Omega_{xy} \, \Omega_{yy}^{-1} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \xi_x \\ \xi_y \end{pmatrix} = \begin{pmatrix} \bar{\xi}_x \\ 0 \end{pmatrix}$$

Table 11.6 Lemma for marginalizing Gaussians in information form.

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

$$\Omega = \begin{pmatrix} \Omega_{xx} & \Omega_{xy} \\ \Omega_{yx} & \Omega_{yy} \end{pmatrix} \qquad \text{and} \qquad \xi = \begin{pmatrix} \xi_x \\ \xi_y \end{pmatrix} \qquad (11.23)$$

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:

$$
\begin{aligned}
& p(x \mid y) \\
&= \eta \, \exp\left\{ -\tfrac{1}{2} \begin{pmatrix} x \\ y \end{pmatrix}^T \begin{pmatrix} \Omega_{xx} & \Omega_{xy} \\ \Omega_{yx} & \Omega_{yy} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} x \\ y \end{pmatrix}^T \begin{pmatrix} \xi_x \\ \xi_y \end{pmatrix} \right\} \\
&= \eta \, \exp\left\{ -\tfrac{1}{2} x^T \Omega_{xx} x + x^T \Omega_{xy} y - \tfrac{1}{2} + y^T \Omega_{yy} y + x^T \xi_x + y^T \xi_y \right\} \\
&= \eta \, \exp\{ -\tfrac{1}{2} x^T \Omega_{xx} x + x^T (\Omega_{xy} y + \xi_x) \underbrace{-\tfrac{1}{2} + y^T \Omega_{yy} y + y^T \xi_y}_{\text{const.}} \} \\
&= \eta \, \exp\{ -\tfrac{1}{2} x^T \Omega_{xx} x + x^T (\Omega_{xy} y + \xi_x) \} \qquad (11.24)
\end{aligned}
$$

Table 11.7 Lemma for marginalizing Gaussians in information form.

11.4.4 Reducing the Information Form

The reduction step **EIF_reduce** is based on a factorization of the full SLAM posterior.

$$
\begin{aligned}
& p(y_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) \\
&= p(x_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) \, p(m \mid x_{0:t}, z_{1:t}, u_{1:t}, c_{1:t}) \qquad (11.25)
\end{aligned}
$$

Here $p(x_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) \sim \mathcal{N}(\Omega, \xi)$ is the posterior over paths alone, with the map integrated out:

$$p(x_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) = \int p(y_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) \, dm \qquad (11.26)$$

As we will show shortly, this probability is caculated in **EIF_reduce** in Table 11.3:

$$p(x_{0:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) \sim \mathcal{N}(\tilde{\xi}, \tilde{\Omega}) \qquad (11.27)$$

In general, such an integration will be intractable, due to the large number of variables in m. For Gaussians, this integral can be calculates 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:

$$\Omega = \begin{pmatrix} \Omega_{x_{0:t},x_{0:t}} & \Omega_{x_{0:t},m} \\ \Omega_{m,x_{0:t}} & \Omega_{m,m} \end{pmatrix} \tag{11.28}$$

$$\xi = \begin{pmatrix} \xi_{x_{0:t}} \\ \xi_m \end{pmatrix} \tag{11.29}$$

The probability (11.27) is then obtained as

$$\tilde{\Omega} = \Omega_{x_{0:t},x_{0:t}} - \Omega_{x_{0:t},m}\, \Omega_{m,m}^{-1}\, \Omega_{m,x_{0:t}} \tag{11.30}$$

$$\tilde{\xi} = \xi_{x_{0:t}} - \Omega_{x_{0:t},m}\, \Omega_{m,m}^{-1}\, \xi_m \tag{11.31}$$

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 landmarks. This makes the inversion efficient:

$$\Omega_{m,m}^{-1} = \sum_j F_j^T\, \Omega_{j,j}^{-1}\, F_j \tag{11.32}$$

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, that is

$$F_j = \begin{pmatrix} 0\cdots0 & 1\ 0\ 0 & 0\cdots0 \\ 0\cdots0 & 0\ 1\ 0 & 0\cdots0 \\ 0\cdots0 & \underbrace{0\ 0\ 1}_{j-\text{th feature}} & 0\cdots0 \end{pmatrix} \tag{11.33}$$

This makes it possible to to decompose the implement Equations (11.30) and (11.31) into a sequential update:

$$\tilde{\Omega} = \Omega_{x_{0:t},x_{0:t}} - \sum_j \Omega_{x_{0:t},j}\, \Omega_{j,j}^{-1}\, \Omega_{j,x_{0:t}} \tag{11.34}$$

$$\tilde{\xi} \quad = \quad \xi_{x_{0:t}} - \sum_j \Omega_{x_{0:t},j} \, \Omega_{j,j}^{-1} \, \xi_j \tag{11.35}$$

The matrix $\Omega_{x_{0:t},j}$ is non-zero only for elements in $\tau(j)$, the set of poses at which landmark j was observed. This essentially proves the correctness of the reduction algorithm **EIF_reduce** in Table 11.3. The operation performed on Ω in this algorithm implments an incomplete variable elimination algorithm for matrix inversion, applied to the landmark variables.

11.4.5   Recovering the Path and the Map

The algorithm **EIF_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 (3.67) and (3.72) on page 55:

$$\tilde{\Sigma} \quad = \quad \tilde{\Omega}^{-1} \tag{11.36}$$
$$\tilde{\mu} \quad = \quad \tilde{\Sigma} \, \tilde{\xi} \tag{11.37}$$

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 (11.25):

$$p(m \mid x_{0:t}, z_{1:t}, u_{1:t}, c_{1:t}) \tag{11.38}$$

The conditioning lemma, stated and proved in Table 11.7, shows that this probability distribution is Gaussian with the parameters

$$\Sigma_m \quad = \quad \Omega_{m,m}^{-1} \tag{11.39}$$
$$\mu_m \quad = \quad \Sigma_m(\xi_m + \Omega_{m,x_{0:t}}\tilde{\xi}) \tag{11.40}$$

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

$$p(m \mid x_{0:t}, z_{1:t}, u_{1:t}, c_{1:t}) \;\; = \;\; \prod_j p(m_j \mid x_{0:t}, z_{1:t}, u_{1:t}, c_{1:t}) \qquad (11.41)$$

where each $p(m_j \mid x_{0:t}, z_{1:t}, u_{1:t}, c_{1:t})$ is distributed according to

$$\Sigma_j \;\; = \;\; \Omega_{j,j}^{-1} \qquad\qquad\qquad\qquad\qquad\qquad (11.42)$$
$$\mu_j \;\; = \;\; \Sigma_j(\xi_j + \Omega_{j,x_{0:t}}\tilde{\mu}) \;\; = \;\; \Sigma_j(\xi_j + \Omega_{j,\tau(j)}\tilde{\mu}_{\tau(j)}) \qquad (11.43)$$

he 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 representation, as locations of different features are correlated through the uncertainty over the robot pose. For this reason, **EIF_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 form—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 THE EIF

Data association in EIFs is realized through correspdoncence variables, just as in EKF SLAM. Just like EKF SLAM, the EIF 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, correspondences in EIF SLAM are defined slightly differently: they are defined over pairs of features in the map, rather than associations of measurements to features. Specifrically, we say $c(j, k) = 1$ iff 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.

Our 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 EIFs consider 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, EIFs can consider the correspondence of any set of landmarks. There is no requirement to process the observed landmarks sequentially.

2. Correspondence search can be interleaved with the calculation of the map. Assuming that two observed landmarks correspond to the same physical landmark 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 EIFs can be *undone*. The goodness of a data association depends on the value of other data association decisions. What once appears to be a good choice may, at some later time in the search

In the remainder of this chapter, we will describe one specific correspondence search algorithms that exploits the first two properties (but not the third). Our data associaiton algorithm will still be greedy, and it will sequentially seaerch the space of possible correspondences to arrive at a plausibale 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, and searching through all of them is infeasible in most environments. Hence, we will be content with a hill climbing algorithm.

11.5.1   The EIF SLAM Algorithm With Unknown Correspondence

The key component of our algorithm is a likelihood test for correspondence. Specifially, EIF 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.

The correspond 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 feagtures corespond to the same feature in the physical world. To calculate this probability,

1:      **Algorithm EIF_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:          $\mu_{\Delta j,k} \; = \; \Omega_{\Delta j,k}^{-1} \, \xi_{\Delta j,k}$

7:          *return* $|2\pi \, \Omega_{\Delta j,k}^{-1}|^{-\frac{1}{2}} \, \exp\left\{ -\frac{1}{2} \, \mu_{\Delta j,k}^{T} \, \Omega_{\Delta j,k}^{-1} \, \mu_{\Delta j,k} \right\}$

Table 11.8   The EIF SLAM test for correspondence: It accepts as input an information representation of the SLAM posterior, along with the result of the **EIF_solve** step. It then outputs the posterior probability that m_j corresponds to m_k.

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 **EIF_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 posterir 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 sub-elements of the information form Ω, ξ, the mean feature locations as specifcied 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 informaiton parameters $\Omega_{\Delta j,k}, \xi_{\Delta j,k}$ are calculated in Lines 4 and 5, and the corresponding expectation for the difference is computed in Line 6. Line 7 return the desired probability: the probability that the difference between m_j and m_k is 0.

The correspondence test provides us now with an algorithm for performing data association search in EIF SLAM. 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 EIF SLAM 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

```
1:      Algorithm EIF_SLAM(u_{1:t}, z_{1:t}):
2:          initialze all c_t^i with a unique value
3:          μ_{0:t} = EIF_initialize(u_{1:t})
4:          Ω, ξ = EIF_construct(u_{1:t}, z_{1:t}, c_{1:t}, μ_{0:t})
5:          Ω̃, ξ̃ = EIF_reduce(Ω, ξ)
6:          μ, Σ_{0:t} = EIF_solve(Ω̃, ξ̃, Ω, ξ)
7:          repeat
8:              for each pair of non-corresponding features m_j, m_k do
9:                  calculate π_{j=k} = EIF_correspondence_test(Ω, ξ, μ, Σ_{0:t}, j, k)
10:                 if π_{j=k} > χ then
11:                     for all c_t^i = k set c_t^i = j
12:                     Ω, ξ = EIF_construct(u_{1:t}, z_{1:t}, c_{1:t}, μ_{0:t})
13:                     Ω̃, ξ̃ = EIF_reduce(Ω, ξ)
14:                     μ, Σ_{0:t} = EIF_solve(Ω̃, ξ̃, Ω, ξ)
15:                 endif
16:             endfor
17:         until no more pair m_j, m_k found with π_{j=k} < χ
18:         return μ
```

Table 11.9 The EIF SLAM algorithm for the full SLAM problem with unknown correspondence. The inner loop of this version 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.

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 EIF SLAM algorithm them interates the construction, reduction, and solution of the SLAM posterior (Lines 12 through 14). As a result, subsequent correspondence tests factor in previous correspondence decisionsm though a newly constructed map. The map construction is terminated when no further features are found in its inner loop.

Clearly, the alorithm **EIF_SLAM** is not particularly efficient. In particular, it tests all feature pairs for correspondence, not just nearby ones. Further, it reconstructs the map whenevr a single correspondence is found; rather than processing sets of corresponding features in batch. Such modifications, however, are relatively straiughtforward. A good implementation of **EIF_SLAM** will clearly be more refined than our basic implementation discussed here.

11.5.2   Mathematical Derivation

We essentially restrict our derivatoion to showing the correctness of the correspon-
dence test in Table **??**. Our first goal shall be to define a posterior probability distribu-
tion 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 iff their location is the
same. Hence, by calculating the posteiror 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:

$$
\begin{aligned}
p(m_j, m_k \mid z_{1:t}, u_{1:t}, c_{1:t}) & \\
= \quad & \int p(m_j, m_k \mid x_{1:t}, z_{1:t}, c_{1:t}) \; p(x_{1:t} \mid z_{1:t}, u_{1:t}, c_{1:t}) \; dx_{1:t}
\end{aligned}
\qquad (11.44)
$$

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 subma-
trices of the joint information form.

The distribution (11.44)is obtained from the joint posterior over $y_{0:t}$, by applying the
marginalization lemma. Specifically, here Ω and ξ represent the joint posterior over
the full state vector $y_{0:t}$, and $\tau(j)$ and $\tau(k)$ denote the sets of poses at which the robot
observed feature j, and feature k, respectively. Further, from our EIF solution we are
given the mean pose vector $\tilde{\mu}$. To apply the marginalization lemma (Table 11.6), we
shall leverage the result of the algorithm **EIF_solve**. Specifically, **EIF_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:

$$
\mu_{[j,k]} \quad = \quad \Omega_{jk,jk}^{-1} \left(\xi_{jk} + \Omega_{jk,\tau(j,k)} \mu_{\tau(j,k)} \right)
\qquad (11.45)
$$

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
EIF_solve, simply because the joint covariance over multiple features requires space
quadratic in the number of features. However, for pairs of feautures the covariance
of the joint is easily recovered. Specicially, 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 **EIF_solve**. Then the marginalization lemma provides us

with the marginal information matrix for the posterior over $(m_j \; m_k)^T$:

$$\Omega_{[j,k]} \;\; = \;\; \Omega_{jk,jk} - \Omega_{jk,\tau(j,k)} \, \Sigma_{\tau(j,k),\tau(j,k)} \, \Omega_{\tau(j,k),jk} \tag{11.46}$$

The information form representation for the desired posterior is now completed by the following information vector:

$$\xi_{[j,k]} \;\; = \;\; \Omega_{[j,k]} \, \mu_{[j,k]} \tag{11.47}$$

Hence we have for the joint

$$p(m_j, m_k \mid z_{1:t}, u_{1:t}, c_{1:t})$$

$$= \;\; \eta \, \exp \left\{ -\tfrac{1}{2} \left(\begin{array}{c} m_j \\ m_k \end{array} \right)^T \Omega_{[j,k]} \left(\begin{array}{c} m_j \\ m_k \end{array} \right) + \left(\begin{array}{c} m_j \\ m_k \end{array} \right)^T \xi_{[j,k]} \right\}$$

$$\tag{11.48}$$

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

$$\Delta_{j,k} \;\; = \;\; m_j \; - \; m_k \tag{11.49}$$

$$= \;\; \left(\begin{array}{c} 1 \\ -1 \end{array} \right)^T \left(\begin{array}{c} m_j \\ m_k \end{array} \right)$$

$$= \;\; \left(\begin{array}{c} m_j \\ m_k \end{array} \right)^T \left(\begin{array}{c} 1 \\ -1 \end{array} \right) \tag{11.50}$$

Plugging this into the definition of a Gaussian in informtation representation, we obtain:

$$p(\Delta_{j,k} \mid z_{1:t}, u_{1:t}, c_{1:t})$$

$$= \;\; \eta \, \exp \left\{ -\tfrac{1}{2} \, \Delta_{j,k}^T \underbrace{\left(\begin{array}{c} 1 \\ -1 \end{array} \right)^T \Omega_{[j,k]} \left(\begin{array}{c} 1 \\ -1 \end{array} \right)}_{=: \, \Omega_{\Delta j,k}} \Delta_{j,k} + \Delta_{j,k}^T \underbrace{\left(\begin{array}{c} 1 \\ -1 \end{array} \right)^T \xi_{[j,k]}}_{=: \, \xi_{\Delta j,k}} \right\}$$

$$= \quad \eta \, \exp\left\{ -\tfrac{1}{2} \, \Delta_{j,k}^T \, \Omega_{\Delta j,k} \, + \Delta_{j,k}^T \, \xi_{\Delta j,k} \right\}^T \tag{11.51}$$

which is Gausian 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 form:

$$p(\Delta_{j,k} \mid z_{1:t}, u_{1:t}, c_{1:t}) \tag{11.52}$$
$$= \quad |2\pi \, \Omega_{\Delta j,k}^{-1}|^{-\tfrac{1}{2}} \, \exp\left\{ -\tfrac{1}{2} \, (\Delta_{j,k} - \mu_{\Delta j,k})^T \, \Omega_{\Delta j,k}^{-1} \, (\Delta_{j,k} - \mu_{\Delta j,k}) \right\}$$

where the mean is given by the obvious expression:

$$\mu_{\Delta j,k} \quad = \quad \Omega_{\Delta j,k}^{-1} \, \xi_{\Delta j,k} \tag{11.53}$$

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:

$$p(\Delta_{j,k} = 0 \mid z_{1:t}, u_{1:t}, c_{1:t}) \quad = \quad |2\pi \, \Omega_{\Delta j,k}^{-1}|^{-\tfrac{1}{2}} \, \exp\left\{ -\tfrac{1}{2} \, \mu_{\Delta j,k}^T \, \Omega_{\Delta j,k}^{-1} \, \mu_{\Delta j,k} \right\}$$
$$\tag{11.54}$$

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 EIF SLAM rely on a number of additional insights and techniques for improving efficiency. Possibly the biggest deficiency of EIF SLAM, as discussed thus far, is due to the fact that in the very beginnning, we assume that all observed features constitute different landmarks. Our algorithm them unifies them one-by-one. For any reasonable number of features, such an approach will be unbearably slow. Further, it will neglect the imprtant constraint that at any point in time, the same feature can only be observed once, but not twice.

Existing implementations of the EIF SLAM idea exploit such opportunities. Spoecifically,

1. Features that are immediately identified to correspond with high likelihood are often unified form the very beginning, before running the full EIF SLAM solution. For exmaple, it is quite common to compule short segments into *local maps*, e.g., local occupancy grid maps. EIF inference is then performed onlhy 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 the EIF solution, specifically the ability to perform data association over large data sets.

2. Many robots are equipped with sensors that observe large number of features at a time. For exampl, laser range finders observe dozens of features. 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. It therefore follows that $i \neq j \longrightarrow c_t^i \neq c_t^j$, that is, at no point in time correspond to different measurements to the same feature.

   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 assiciate 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.

3. The EIF algorithm stated in this chapter does not make use of its ability to *undo* data asociations. 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. in particular, one can change the correspondence variables of any two measurements in arbitrary ways in our algothm above. However, it is more difficult to test whether a data associaiton should be undone, as there is no (obvious) test for testing wether two previously associated features should be distinct. A simple implementation involves undoing a data association in question, rebyuilding the map, and testing whether our criterion above still calls for correspondence. Such an approach can be computationally involved, as it provides no means of detecting which data association to test. Mechnicsms for detecting unlikely associations are outside the scope of this book, but should be considered when implementing this approach.

Figure 11.1 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.

4. Finally, the EIF algorihtm does not consider negative information. In practice, not seeing a feature can be as informative as seeing one. However, the simple formulaiton above 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 probabiliustic calculations through approximations. One such example will be given in the next section.

11.7   EMPIRICAL IMPLEMENTATION

In the remainder of tihs chapter, we will highlight empirical results for an EIF SLAM implementation. The vehicle used in our experiment is Figure 11.1; it is a robot designed to map abandoned mines.

The type map collected by the robot is shown in Figure 11.2. 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 EIF SLAM, 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 11.2.
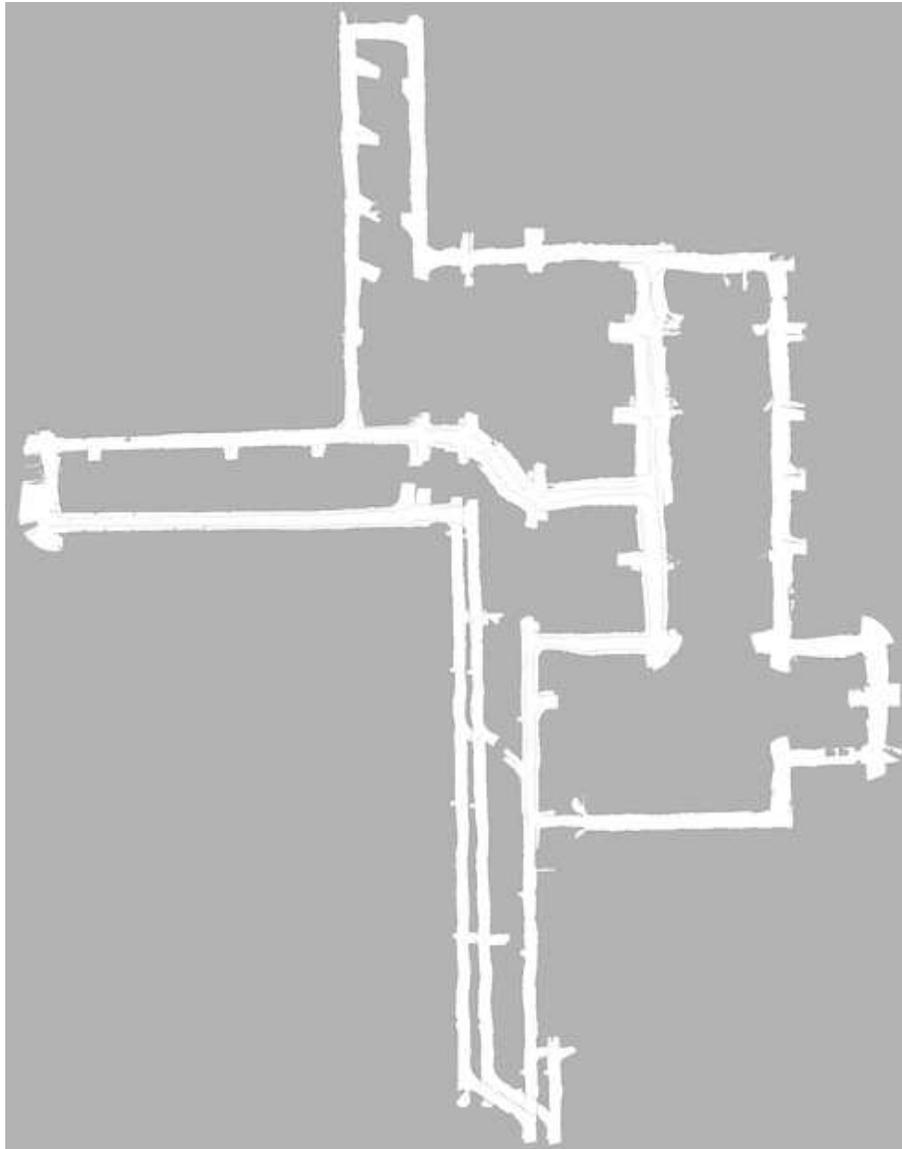
Figure 11.2 Map if a mine, acquired by pairwise scan matching. The diameter of this environment is approximately 250 meters. The map is obvisouly inconsistent, in that several hallways show up more than once.
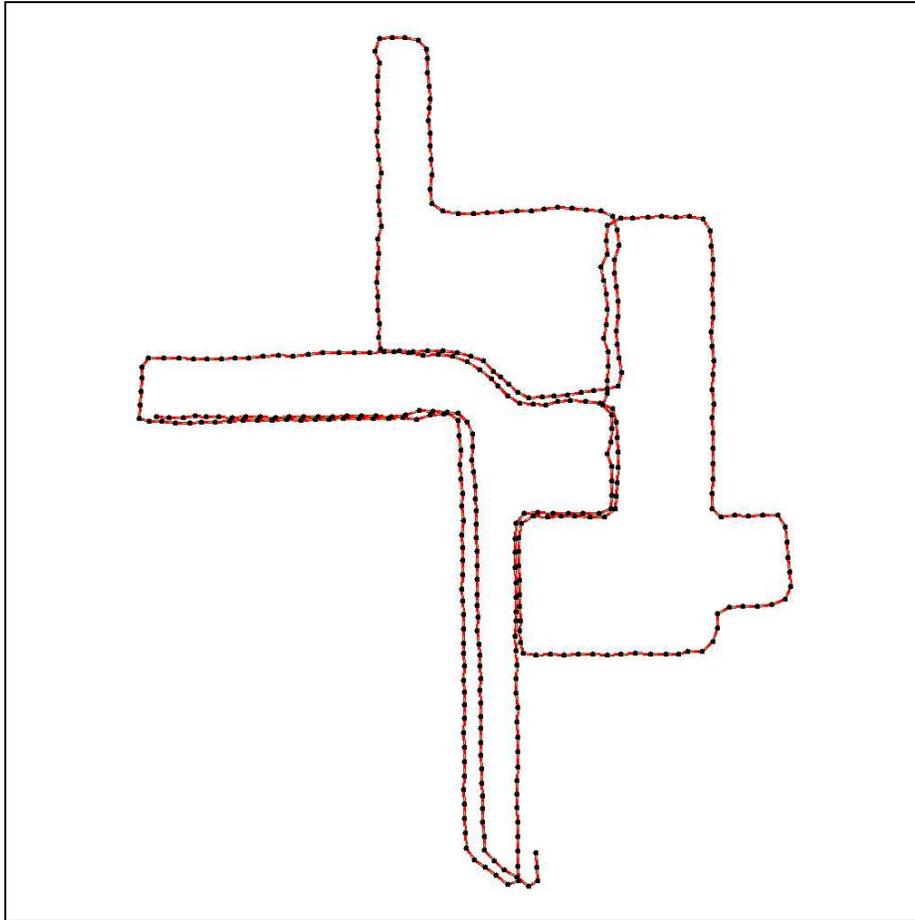
Figure 11.3 Map if a mine, acquired by pairwise scan matching. The diameter of this environment is approximately 250 meters. The map is obvisouly inconsistent, in that several hallways show up more than once.

To apply the EIF SLAM 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 the scan matching data association technique performs essentially flawlessly. Each submap's coordinates become a pose node in the EIF SLAM. Adjacent submaps are linked through the relative motion constraints between them. The resulting structure is shown in Figure 11.3.

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 11.4 illustrates the process of data association: The circles each correspond to a new constraint that is imposed when contracting the information form of the EIF. 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 11.5. 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.
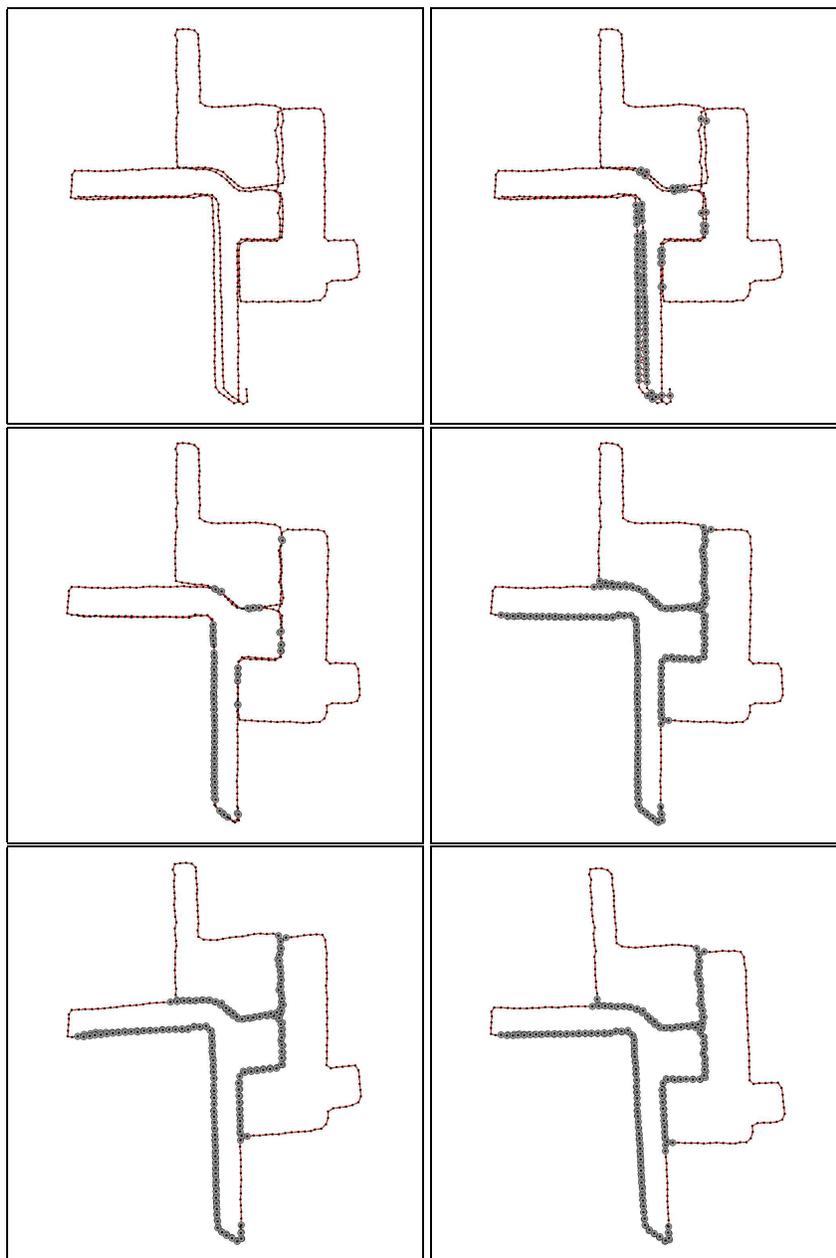
Figure 11.4   Data association search.
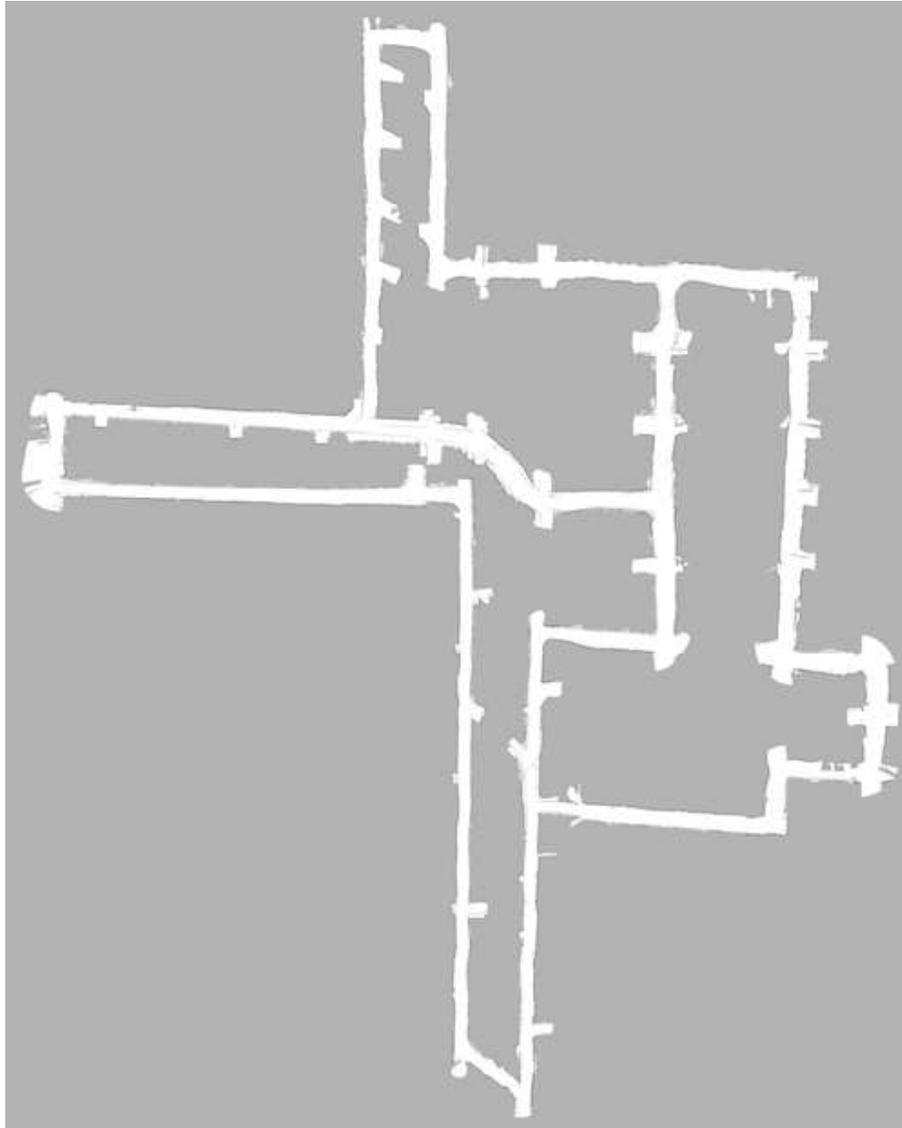
Figure 11.5 Final map, after optimizing for data associations.

11.8   SUMMARY

This chapter introduced the extended information filter (EIF) approach to to the full SLAM problem.

- the EIF SLAM algorithm addresses the full SLAM problem. It calculates posteriors over the full robot path along with the map. Therefore, EIF SLAM is a batch algorithm, not an online algorithm like EKF SLAM.

- The key insight of the EIF SLAM algorithm is that the structure of information is sparse. Specifically,

    - 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.

    EIF SLAM simply records all this information, 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 the robot path.

- The EIF SLAM algorithm recovers maps through an iterative procedure which involves three steps: Construction of a linear information form through Taylor extension; reduction of this form; and solving the resulting optimization problem. These three steps effectively resolve the information, and produce a consistent probabilistic posterior over the path and the map. Since EIF SLAM is run batch, we can repeat the linearization step and achieve gradually improved results.

- Data association in EIF SLAM is performed by calculating the probability that two features have identical world coordinates. Since EIF SLAM 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 EIF SLAM 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.

- We briefly provided results for a variant of EIF SLAM that follows the decomposition idea, and that uses occupancy grid maps for representing sets of range

scans. Despite these approximations, we find that the EIF data association and inference techniques yield favorable results in a large-scale mapping problem.

12

THE SPARSE EXTENDED
INFORMATION FILTER

12.1   INTRODUCTION

The previous two chapters covered two ends of a spectrum of SLAM algorithms.EKF SLAM is *proactive*. Every time information is acquired, it resolves this information into a probability distribution. This step is computationally expensive: for n features in the map, it requires $O(n^2)$ computations per update. The EIF SLAM algorithm is different: It simply accumulates information. Such an accumulation is *lazy*: At the time of data acquisition, EIF SLAM simply memorizes the information is receives. To turn the accumulated information into a map, EIF SLAM had to perform inference. In EIF SLAM, this inference is performed after all data is acquired. This makes EIF an offline algortihm.

This raises the question as to whether we can devise an *online* SLAM algorithm which inherits the efficiency of the information representation. The answer is yes, but only with a number of approximations. The *sparse extended information filter*, or SEIF, implements an information solution to the online SLAM problem. Just like the EKF, the SEIF integrates out past robot poses, and only maintains a posterior over the present robot pose and the map. But like EIFs and unlike EKFs, the SEIF maintains an information representation of all knowledge. In doing so, updating the SEIF becomes a lazy information shifting operation; which is superior to the proactive probabilistic update of the EKF. Thus, SEIFs can be seen as the best of both worlds: They can be run online, and they are efficient.
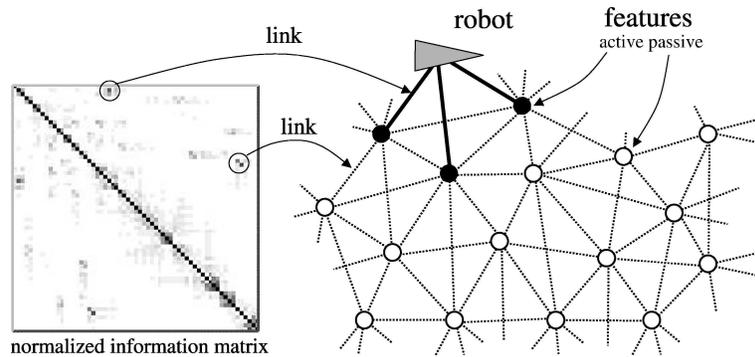
Figure 12.1 Illustration of the network of features generated by our approach. Shown on the left is a sparse information matrix, and on the right a map in which entities are linked whose information matrix element is non-zero. As argued in the paper, the fact that not all features are connected is a key structural element of the SLAM problem, and at the heart of our constant time solution.

As an online algorithm, the SEIF maintains a belief over the very same state vector as the EKF:

$$y_t = \begin{pmatrix} x_t \\ m \end{pmatrix} \qquad\qquad (12.1)$$

Here x_t is the robot state, and m the map. The posterior under known correspondences is given by $p(y_t \mid z_{1:t}, u_{1:t}, c_{1:t})$. This posterior differs from the one estimated by EIFs in that EIFs recovr the entire robot path.

The key insight for turning EIFs into an online SLAM algorithm is illustrated in Figure **??**. This figure shows the result of the EKF SLAM algorithm, in a simulated environment containing 50 landmarks. The left panel shows a moving robot, along with its probabilistic estimate of the location of all 50 point features. The central information maintained by the EKF SLAM is a covariance matrix of these different estimates. The correlation, which is the normalized covariance, is visualized in the center panel of this figure. Each of the two axes lists the robot pose (location and orientation) followed by the 2-D locations of all 50 landmarks. Dark entries indicate strong correlations. We already discussed in the EKF SLAM chapter that in the limit, all feature coordinates become fully correlated—hence the checker-board appearance of the correlation matrix.

The right panel of Figure **??** shows the information matrix Ω_t, normalized just like the correlation matrix. As in the previous chapter, elements in this normalized information matrix can be thought of as constraints, or links, which constrain the relative locations of pairs of features in the map: The darker an entry in the display, the stronger the link. As this depiction suggests, the normalized information matrix appears to be *sparse*: it is dominated by a small number of strong links; and it possesses a large number of links whose values, when normalized, are near zero. Furthermore, the strength of each link is related to the distance of the corresponding features: Strong links are found only between metrically nearby features. The more distant two features, the weaker their link. This sparseness is distinctly different from that in the previous chapter: First, there exist strong links between pairs of landmarks. In the previous chapter, no such links could exist. Second, the sparseness is only approximate: In fact, all elements of the normalized information matrix are non-zero, but nearly all of them are very close to zero.

The SEIF SLAM algorithm exploits this insight by maintaining a *sparse* information matrix, in which only nearby features are linked through a non-zero element. The resulting network structure is illustrated in the right panel of Figure 12.1, where disks correspond to point features and dashed arcs to links, as specified in the information matrix visualized on the left. This diagram also shows the robot, which is linked to a small subset of all features only. Those features are called *active features* and are drawn in black. Storing a sparse information matrix requires space linear in the number of features in the map. More importantly, all essential updates in SEIF SLAM can be performed in constant time, regardless of the number of features in the map. This result is somewhat surprising, as a naive implementation of motion updates in information filters—a perhaps in Table **??** on page 60—requires inversion of the entire information matrix.

The SEIF is an online SLAM algorithm that maintains such a sparse information matrix, and for which the time required for all update steps is *independent* of the size of the map. This makes SEIF the first efficiet online algorithm for the SLAM problem, encountered in this book.

12.2 INTUITIVE DESCRIPTION

We begin with an intuitive description of the SEIF update, using grphical illustrations. Specifically, a SEIF update is composed of 4 steps: a mogtion update step, a measurement update step, a sparsificaiton step, and a state estimation step.
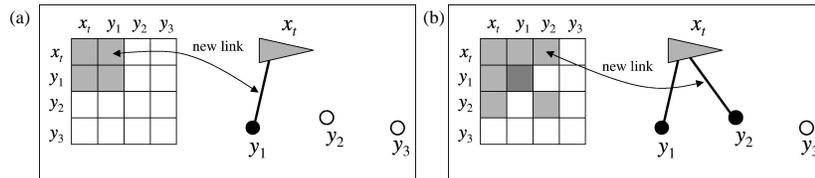
Figure 12.2   The effect of measurements on the information matrix and the associated network of features: (a) Observing y_1 results in a modification of the information matrix elements Ω_{x_t, y_1}. (b) Similarly, observing y_2 affects Ω_{x_t, y_2}. Both updates can be carried out in constant time.
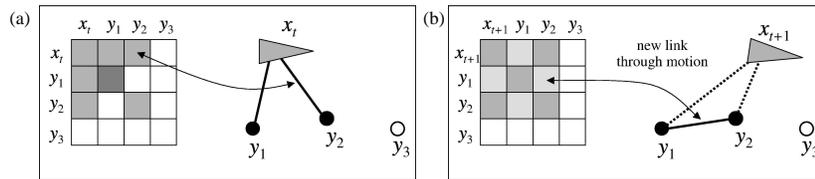
Figure 12.3   The effect of motion on the information matrix and the associated network of features: (a) before motion, and (b) after motion. If motion is non-deterministic, motion updates introduce new links (or reinforce existing links) between any two active features, while weakening the links between the robot and those features. This step introduces links between pairs of features.
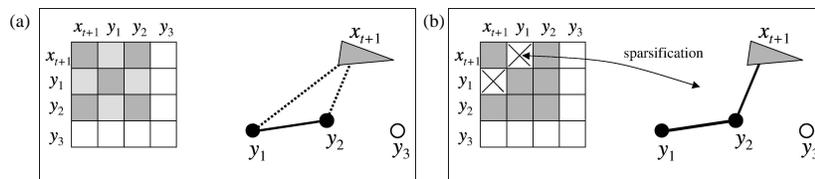
Figure 12.4   Sparsification: A feature is deactivated by eliminating its link to the robot. To compensate for this change in information state, links between active features and/or the robot are also updated. The entire operation can be performed in constant time.

We egin with the measurement update step, depicted in Figure 12.2. Each of the two panels show the information matrix maintained by the SEIF, along with the graph defined by the information links. In the measurement update, sensing a feature y_1 leads the SEIF to update the off-diagonal element of its information matrix that links the robot pose estimate x_t to the observed feature y_1. This is illustrate din the left panel of Figure 12.2a. Similarly, sensing y_2 leads it to update the elements in the information matrix that link the robot pose x_t and the feature y_2, as illustrated in Figure 12.2b. As we shall see, each of these updates correspond to local additions in the information matrix (and the information vector). In both cases (information matrix and vector), this addition touches only elements that link the robot pose variable to the observed feature. This makes the measurement update noticeably efficient than the corresponding update steo in EKFs. In particular, the complexity of updating a SEIF in response to a measurement takes time independent of the size of the map.

The motion update is shown in Figure 12.3. Here a robot's pose changes; Figure 12.3a depicts a the information state before, and Figure 12.3b after motion, respectively. The motion affects the information state in multiple ways. First, the links between the robot's pose and the features y_1, y_2 are weakened. This is a result of the fact that robot motion is erroneous, hence causes us to lose information about where the robot is relative to features in the map. However, this information is not entirely lost. Some of it is mapped into information links between pairs of features. This shift of information comes about since even though we lost information on the robot pose, we did not lose information on the relative location of features in the map. Whereas previously, those features were linked indirectly through the robot pose, they are now linked also directly after the update step.

The shift of information from robot pose links to between-feature links is a key element of the SEIF. In particular, the EIF discussed in the previous chapter never introduced any links between pairs of features. It is a direct consequence of using the information form as a filter, for the online SLAM problem. By integrating out past pose variables, we lose those links, and they are mapped back into the between-feature elements in the information matrix.

For a pair of features to acquire a direct link in this process, both have to be active before the update, that is, their corresponding elements that links them to the robot pose in the information matrix have to be non-zero. This is illustrated in Figure 12.3: A between-feature link is only introduced between features y_1 and y_2. Feature y_3, which is not active, remains untouched. This suggests that by controlling the number of active landmarks at any point in time, we can control the computational complexity of the motion update, and the number of links in the information matrix. If the number of active links remains small, so will the update complexity for the motion update, and so will the number of non-zero between-landmark elements in the information matrix.

The information matrix in SLAM problem is approximately sparse. As we have argued above, the information matrix is dominated by a small number of elements between nearby features in the world, and links between features further away are comparatively small. However, the sparseness is only approximate; it is not exact. SEIF therefore employs a *sparsification* step, illustrated in Figure 12.4. The sparsification involves the removal of a link between the robot and an active feature, effectively turning the active feature into a passive one. In SEIFs, this arc removal leads to a redistribution of information into neighboring links, specifically between other active features and the robot pose, and the time to perform sparsification is independent of the size of the map. However, it is an approximation, one that induces an information loss in the robot's posterior. The benefit of this approximation is that it induces true sparseness, and hence makes it possible to update the filter efficiently.

There exists one final step in the SEIF algorithm, which is not depicted in any of the figures. This step involves the propagation of a mean estimate through the graph. As was already discussed in Chapter 3, the extended information filter requires an estimate of the state μ_t for linearization of the motion and the measurement model. SEIFs also require a state estimate for the sparsification step.

Clearly, one could recover the state estimate through the equation $\mu = \Omega^{-1}\xi$, where Ω is the information vector, and ξ the information state. However, this would require inverting a large matrix, which would render SEIFs computationally inefficient. SEIFs circumvent the step by an iterative algorithm that effectively propagates state estimates through the information graph. Each local state estimate is updated based on the best estimates of its neighbors in the information graph. This iterative algorithm converges to the true mean μ. Since the information form is sparse in SEIFs, each such update requires constant time, though with the caveat that more than a finite number of such updates may be needed to achieve good results. To keep the computation independent of the size of the state space, SEIFs perform a fixed number of such updates at any iteration. The resulting state vector is only an approximation, which is used instead of the correct mean estimate in all updating steps.

12.3   THE SEIF SLAM ALGORITHM

The outer loop of the SEIF update is depicted in Table 12.1. The algorithm accepts as input an information matrix Ω_{t-1}, the information vector ξ_{t-1}, and an estimate of the state μ_{t-1}. It also accepts a measurement z_t, a control u_t, and a correspondence vector c_t. The output of the algorithm **SEIF_SLAM_known_correspondences** is a new state

---

1:          **Algorithm SEIF_SLAM_known_correspondences**$(\xi_{t-1}, \Omega_{t-1}, \mu_{t-1}, u_t, z_t, c_t)$:

2:          $\bar{\xi}_t, \bar{\Omega}_t, \bar{\mu}_t = $ **SEIF_motion_update**$(\xi_{t-1}, \Omega_{t-1}, \mu_{t-1}, u_t)$

3:          $\mu_t = $ **SEIF_update_state_estimate**$(\bar{\xi}_t, \bar{\Omega}_t, \bar{\mu}_t)$

4:          $\xi_t, \Omega_t = $ **SEIF_measurement_update**$(\bar{\xi}_t, \bar{\Omega}_t, \mu_t, z_t, c_t)$

5:          $\tilde{\xi}_t, \tilde{\Omega}_t = $ **SEIF_sparsification**(ξ_t, Ω_t)

6:          *return* $\tilde{\xi}_t, \tilde{\Omega}_t, \mu_t$

Table 12.1   The Sparse Extended Information Filter algorithm for the SLAM Problem, here with known data association.

1:          **Algorithm SEIF_motion_update**$(\xi_{t-1}, \Omega_{t-1}, \mu_{t-1}, u_t)$:

2:          $F_x = \begin{pmatrix} 1 & 0 & 0 & 0\cdots0 \\ 0 & 1 & 0 & 0\cdots0 \\ 0 & 0 & 1 & \underbrace{0\cdots0}_{2N} \end{pmatrix}$

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

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

5:          $\Psi_t = F_x^T \left[(I + \Delta)^{-1} - I \right] F_x$

6:          $\lambda_t = \Psi_t^T \, \Omega_{t-1} + \Omega_{t-1} \, \Psi_t + \Psi_t^T \, \Omega_{t-1} \, \Psi_t$

7:          $\Phi_t = \Omega_{t-1} + \lambda_t$

8:          $\kappa_t = \Phi_t \, F_x^T (R_t^{-1} + F_x \, \Phi_t \, F_x^T)^{-1} \, F_x \, \Phi_t$

9:          $\bar{\Omega}_t = \Phi_t - \kappa_t$

10:         $\bar{\xi}_t = \xi_{t-1} + (\lambda_t - \kappa_t) \, \mu_{t-1} + \bar{\Omega}_t \, F_x^T \, \delta_t$

11:         $\bar{\mu}_t = \mu_{t-1} + F_x^T \, \delta$

12:         *return* $\bar{\xi}_t, \bar{\Omega}_t, \bar{\mu}_t$

Table 12.2   The motion update in SEIFs.

1:     **Algorithm SEIF_measurement_update$(\bar{\xi}_t, \bar{\Omega}_t, \mu_t, z_t, c_t)$:**

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

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

4:     $j = c_t^i$

5:     *if landmark j never seen before*

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

7:     *endif*

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

9:     $q = \delta^T \delta$

10:     $\hat{z}_t^i = \begin{pmatrix} \sqrt{q} \\ \text{atan2}(\delta_y, \delta_x) - \mu_{t,\theta} \\ \mu_{j,s} \end{pmatrix}$

11:     $H_t^i = \frac{1}{q} \begin{pmatrix} \sqrt{q}\delta_x & -\sqrt{q}\delta_y & 0 & 0\cdots0 & -\sqrt{q}\delta_x & \sqrt{q}\delta_y & 0 & 0\cdots0 \\ \delta_y & \delta_x & -1 & 0\cdots0 & -\delta_y & -\delta_x & 0 & 0\cdots0 \\ 0 & 0 & 0 & \underbrace{0\cdots0}_{3j-3} & 0 & 0 & 1 & \underbrace{0\cdots0}_{3j} \end{pmatrix}$

12:     *endfor*

13:     $\xi_t = \bar{\xi}_t + \sum_i H_t^{iT} \, Q_t^{-1} \, [z_t^i - \hat{z}_t^i - H_t^i \, \mu_t]$

14:     $\Omega_t = \bar{\Omega}_t + \sum_i H_t^{iT} \, Q_t^{-1} \, H_t^i$

15:     *return ξ_t, Ω_t*

Table 12.3   The measurement update step in SEIFs.

1:        **Algorithm SEIF_sparsification(ξ_t, Ω_t):**
2:        *define F_{m_0}, F_{x,m_0}, F_x as projection matrices from y_t to m_0, $\{x, m_0\}$,*
             and x, respectively

3:        $\tilde{\Omega}_t = \Omega_t - \Omega_t^0 F_{m_0} (F_{m_0}^T \Omega_t^0 F_{m_0})^{-1} F_{m_0}^T \Omega_t^0$
             $+ \Omega_t^0 F_{x,m_0} (F_{x,m_0}^T \Omega_t^0 F_{x,m_0})^{-1} F_{x,m_0}^T \Omega_t^0$
             $- \Omega_t F_x (F_x^T \Omega_t F_x)^{-1} F_x^T \Omega_t$

4:        $\tilde{\xi}_t = \xi_t + \mu_t (\tilde{\Omega}_t - \Omega_t)$
5:        *return* $\tilde{\xi}_t, \tilde{\Omega}_t$

Table 12.4   The sparsification step in SEIFs.

1:        **Algorithm SEIF_update_state_estimate($\bar{\xi}_t, \bar{\Omega}_t, \bar{\mu}_t$):**
2:        *for a small set of map features m_i do*
3:        $F_i = \begin{pmatrix} 0 \cdots 0 & 1 & 0 & 0 \cdots 0 \\ \underbrace{0 \cdots 0}_{2(N-i)} & 0 & 1 & \underbrace{0 \cdots 0}_{2(i-1)x} \end{pmatrix}$
4:        $\mu_{i,t} = (F_i \Omega_t F_i^T)^{-1} F_i [\xi_t - \Omega_t \bar{\mu}_t + \Omega_t F_i^T F_i \bar{\mu}_t]$
5:        *endfor*

6:        *for all other map features m_i do*
7:        $\mu_{i,t} = \bar{\mu}_{i,t}$
8:        *endfor*

9:        $F_x = \begin{pmatrix} 1 & 0 & 0 & 0 \cdots 0 \\ 0 & 1 & 0 & 0 \cdots 0 \\ 0 & 0 & 1 & \underbrace{0 \cdots 0}_{2N} \end{pmatrix}$
10:       $\mu_{x,t} = (F_x \Omega_t F_x^T)^{-1} F_x [\xi_t - \Omega_t \bar{\mu}_t + \Omega_t F_x^T F_x \bar{\mu}_t]$

11:       *return* μ_t

Table 12.5   The amortized state update step in SEIFs updates a small number of state estimates.

estimate, represented by the information matrix Ω_t and the information vector ξ_t. The algorithm also outputs an improved estimate μ_t.

The SEIF update is implemented in 4 steps. The motion update in Table 12.2 incorporates the control u_t into the filter estimate. It does so through a number of operations, which ensure computational efficiency. Specifically, the only components of the information vector/matrix that are modified in this update are those of the robot pose and the active features. The measurement update in Table 12.3 incorporates the measurement vector z_t under known correspondence c_t. This step is also local, just like the motion update step. It only updates the informaiton values of the robot pose and the observed features in the map. The sparsification step, shown in Table 12.4, is an approximate step: It removes active features by transforming the informaiton matrix and hte information vector accordingly. This step is again efficient; it only modifies links between the robot and the active landmarks. Finally, the state estimate update in Table 12.5, applies an amortized coordinate descent technique to recover the state estimate μ_t. This step once again exploits the sparseness of the SEIF, through which it only has to consult a small number of other state vector elements in each incremental update.

Together, the entire update loop of the SEIF is constant time, in that the processing time is independent of the size of the map. This is at stark contrast to the EKF, which requires time quadraticin the size of the map for each update. However, this "constant time" statement should be taken with a grain of salt: The recovery of state estimates is a computational problem for which no linear-time solution is presently known, if the environment possesses large cycles. Thus, the result of SEIF, when run as constant time filter, may degrade as the map diameter increases. We will revisit this issue at the very end of the paper.

12.4   MATHEMATICAL DERIVATION

As usual, the reader may skip the mathematical derivation of the SEIF, and proceed directly to Section 12.7 on page 323.

12.4.1   Motion Update

The motion update in SEIF processes the control u_t. It does so by transforming the information matrix Ω_{t-1} and the information vector ξ_{t-1} into a new matrix $\bar{\Omega}_t$ and vector $\bar{\xi}_t$, representing the belief at time t. As usual, the bar in our notation indicates

that this prediction is only based on the control; it does not yet take the measurement into account.

The derivation of the motion update is best started with the corresponding formula for the EKF. We begin with the algorithm **EKF_SLAM_known_correspondences** in Table 10.1, page 249. Lines 3 and 5 state the motion update, which we restate here for the reader's convenience:

$$\bar{\mu}_t = \mu_{t-1} + F_x^T \, \delta \tag{12.2}$$
$$\bar{\Sigma}_t = G_t \, \Sigma_{t-1} \, G_t^T + F_x^T \, R_t \, F_x \tag{12.3}$$

The essential elements of this update were defined as follows:

$$F_x = \begin{pmatrix} 1 & 0 & 0 & 0\cdots 0 \\ 0 & 1 & 0 & 0\cdots 0 \\ 0 & 0 & 1 & 0\cdots 0 \end{pmatrix} \tag{12.4}$$

$$\delta = \begin{pmatrix} -\frac{v_t}{\omega_t}\sin\mu_{t-1,\theta} + \frac{v_t}{\omega_t}\sin(\mu_{t-1,\theta} + \omega_t\Delta t) \\ \frac{v_t}{\omega_t}\cos\mu_{t-1,\theta} - \frac{v_t}{\omega_t}\cos(\mu_{t-1,\theta} + \omega_t\Delta t) \\ \omega_t\Delta t \end{pmatrix} \tag{12.5}$$

$$\Delta = \begin{pmatrix} 0 & 0 & \frac{v_t}{\omega_t}\cos\mu_{t-1,\theta} - \frac{v_t}{\omega_t}\cos(\mu_{t-1,\theta} + \omega_t\Delta t) \\ 0 & 0 & \frac{v_t}{\omega_t}\sin\mu_{t-1,\theta} - \frac{v_t}{\omega_t}\sin(\mu_{t-1,\theta} + \omega_t\Delta t) \\ 0 & 0 & 0 \end{pmatrix} \tag{12.6}$$

$$G_t = I + F_x^T \, \Delta \, F_x \tag{12.7}$$

In SEIFs, we have to define the motion update over the information vector ξ and the information matrix Ω. From Equation (12.3), the definition of G_t in (12.7), and the information matrix equation $\Omega = \Sigma^{-1}$, it follows that

$$\begin{aligned} \bar{\Omega}_t &= \left[G_t \, \Omega_{t-1}^{-1} \, G_t^T + F_x^T \, R_t \, F_x \right]^{-1} \\ &= \left[(I + F_x^T \, \Delta \, F_x) \, \Omega_{t-1}^{-1} \, (I + F_x^T \, \Delta \, F_x)^T + F_x^T \, R_t \, F_x \right]^{-1} \end{aligned} \tag{12.8}$$

A key insight is this update can be implemented in constant time—regardless of the dimension of Ω. The fact that this is possible for sparse matrices Ω_{t-1} is somewhat non-trivial, since Equation (12.8) seems to require two nested inversions of matrices of size $(2n + 3) \times (2n + 3)$. As we shall see, if Ω_{t-1} is sparse, this update step can

be carried out efficiently. We define

$$
\begin{aligned}
\Phi_t &= \left[G_t\, \Omega_{t-1}^{-1}\, G_t^T \right]^{-1} \\
&= [G_t^T]^{-1}\, \Omega_{t-1}\, G_t^{-1}
\end{aligned}
\tag{12.9}
$$

and hence Equation (12.8) can be rewritten as

$$
\bar{\Omega}_t = \left[\Phi_t^{-1} + F_x^T\, R_t\, F_x \right]^{-1}
\tag{12.10}
$$

We now apply the matrix inversion lemma and obtain:

$$
\begin{aligned}
\bar{\Omega}_t &= \left[\Phi_t^{-1} + F_x^T\, R_t\, F_x \right]^{-1} \\
&= \Phi_t - \underbrace{\Phi_t\, F_x^T (R_t^{-1} + F_x\, \Phi_t\, F_x^T)^{-1}\, F_x\, \Phi_t}_{\kappa_t} \\
&= \Phi_t - \kappa_t
\end{aligned}
\tag{12.11}
$$

Here κ_t is defined as indicated. This expression can be calculated in constant time *if* we can compute Φ_t in constant time from Ω_{t-1}. To see that this is indeed possible, we note that the argument inside the inverse, $R_t^{-1} + F_x\, \Phi_t\, F_x^T$, is 3-dimensional. Multiplying this inverse with F_x^T and F_x induces a matrix that is of the same size as Ω; however, this matrix is only non-zero for the 3×3 sub-matrix corresponding to the robot pose. Multiplying this matrix with a sparse matrix Ω_{t-1} (left and right) touches only elements for which the off-diagonal element in Ω_{t-1} between the robot pose and a map feature is non-zero. Put differently, the result of this operation only touches rows and columns that correspond to active features in the map. Since sparsity implies that the number of active features in Ω_{t-1} is independent of the size of Ω_{t-1}, the total number of non-zero elements in κ_t is also $O(1)$. Consequently, the subtraction requires $O(1)$ time.

It remains to be shown that we can calculate Φ_t from Ω_{t-1} in constant time. We begin with a consideration of the inverse of G_t, which is efficiently calculated as follows:

$$
\begin{aligned}
G_t^{-1} &= (I + F_x^T\, \Delta\, F_x)^{-1} \\
&= (I \underbrace{- F_x^T\, I\, F_x + F_x^T\, I\, F_x}_{=\,0} + F_x^T\, \Delta\, F_x)^{-1}
\end{aligned}
$$

$$
\begin{aligned}
&= (I - F_x^T \, I \, F_x + F_x^T \, (I + \Delta) \, F_x)^{-1} \\
&= I - F_x^T \, I \, F_x + F_x^T \, (I + \Delta)^{-1} \, F_x b \\
&= I + \underbrace{F_x^T \, [(I + \Delta)^{-1} - I] \, F_x}_{\Psi_t} \\
&= I + \Psi_t
\end{aligned}
\tag{12.12}
$$

By analogy, we get for the transpose $[G_t^T]^{-1} = (I + F_x^T \, \Delta^T \, F_x)^{-1} = I + \Psi_t^T$. Here the matrix Ψ_t is only non-zero for elements that correspond to the robot pose. It is zero for all features in the map, and hence can be computed in constant time. This gives us for our desired matrix Φ_t the following expression:

$$
\begin{aligned}
\Phi_t &= (I + \Psi_t^T) \, \Omega_{t-1} \, (I + \Psi_t) \\
&= \Omega_{t-1} + \underbrace{\Psi_t^T \, \Omega_{t-1} + \Omega_{t-1} \, \Psi_t + \Psi_t^T \, \Omega_{t-1} \, \Psi_t}_{\lambda_t} \\
&= \Omega_{t-1} + \lambda_t
\end{aligned}
\tag{12.13}
$$

where Ψ_t is zero except for the sub-matrix corresponding to the robot pose. We nore because Ω_{t-1}, λ_t is zero except for a finite number of elements, which correspond to active map features and the robot pose.

Hence, Φ_t can be computed from Ω_{t-1} in constant time, assuming that Ω_{t-1} is sparse. Equations ... are equivalent to Lines ... in Table 10.1, which proves the correctness of the information matrix update in **EKF_SLAM_known_correspondences**.

Finally, we show a similar result for the information vector. Line From (12.2) we obtain

$$
\bar{\mu}_t = \mu_{t-1} + F_x^T \, \delta_t
\tag{12.14}
$$

This implies for the information vector:

$$
\begin{aligned}
\bar{\xi}_t &= \bar{\Omega}_t \, (\Omega_{t-1}^{-1} \, \xi_{t-1} + F_x^T \, \delta_t) \\
&= \bar{\Omega}_t \, \Omega_{t-1}^{-1} \, \xi_{t-1} + \bar{\Omega}_t \, F_x^T \, \delta_t \\
&= (\bar{\Omega}_t + \Omega_{t-1} - \Omega_{t-1} + \Phi_t - \Phi_t) \, \Omega_{t-1}^{-1} \, \xi_{t-1} + \bar{\Omega}_t \, F_x^T \, \delta_t \\
&= (\bar{\Omega}_t \, \underbrace{-\Phi_t + \Phi_t}_{= 0} \, \underbrace{-\Omega_{t-1} + \Omega_{t-1}}_{= 0}) \, \Omega_{t-1}^{-1} \, \xi_{t-1} + \bar{\Omega}_t \, F_x^T \, \delta_t
\end{aligned}
$$

$$
\begin{aligned}
&= \quad \underbrace{(\bar{\Omega}_t - \Phi_t}_{= -\kappa_t} + \underbrace{\Phi_t - \Omega_{t-1})}_{= \lambda_t} \ \underbrace{\Omega_{t-1}^{-1} \, \xi_{t-1}}_{= \mu_{t-1}} + \underbrace{\Omega_{t-1} \, \Omega_{t-1}^{-1}}_{= I} \, \xi_{t-1} + \bar{\Omega}_t \, F_x^T \, \delta_t \\
&= \quad \xi_{t-1} + (\lambda_t - \kappa_t) \, \mu_{t-1} + \bar{\Omega}_t \, F_x^T \, \delta_t \qquad\qquad\qquad (12.15)
\end{aligned}
$$

Since λ_t and κ_t are both sparse, the product $(\lambda_t - \kappa_t) \, \mu_{t-1}$ only contains finitely many non-zero elements and can be calculated in constant time. Further, $F_x^T \, \delta_t$ is a sparse matrix. The sparseness of the product $\bar{\Omega}_t \, F_x^T \, \delta_t$ follows now directly from the fact that $\bar{\Omega}_t$ is sparse as well.

12.4.2   Measurement Updates

The second important step of SLAM concerns the update of the filter in accordance to robot motion. The measurement update in SEIF directly implements the general extended information filter update, as stated in Lines 6 and 7 of Table 3.5 on page 60:

$$
\begin{aligned}
\Omega_t &= \bar{\Omega}_t + H_t^T \, Q_t^{-1} \, H_t && (12.16) \\
\xi_t &= \bar{\xi}_t + H_t^T \, Q_t^{-1} \, [z_t - h(\bar{\mu}_t) - H_t \, \mu_t] && (12.17)
\end{aligned}
$$

Writing the prediction $\hat{z}_t = h(\bar{\mu}_t)$ and summing over all individual elements in the measurement vector leads to the form in Lines 13 and 14 in Table 12.3:

$$
\Omega_t = \bar{\Omega}_t + \sum_i H_t^{iT} \, Q_t^{-1} \, H_t^i \qquad\qquad (12.18)
$$

$$
\xi_t = \bar{\xi}_t + \sum_i H_t^{iT} \, Q_t^{-1} \, [z_t^i - \hat{z}_t^i - H_t^i \, \mu_t] \qquad\qquad (12.19)
$$

Here Q_t, δ, q, and H_t^i are defined as before (e.g., Table 11.2 on page 272).

12.5   SPARSIFICATION

12.5.1   General Idea

The key step in SEIFs concerns the sparsification of the information matrix Ω_t. Because sparsification is so essential to SEIFs, let us first discuss it in general terms before we apply it to the information filter. Sparsification is an approximation whereby a

posterior distribution is approximated by two of its marginals. Suppose a, b, and c are sets of random variables (not to be confused with any other occurrence of these variables in this book!), and suppose we are given a joint distribution $p(a, b, c)$ over these variables. To sparsify this distribution, suppose we would like to remove any direct link between the variables a and b. In other words, we would like to approximate p by a distribution $\tilde{p}$ for which the following property holds: $\tilde{p}(a \mid b, c) = p(a \mid c)$ and $\tilde{p}(b \mid a, c) = p(b \mid c)$. In multivariate Gaussians, it is easily shown that this conditional independence is equivalent to the absence of a direct link between a and b, that is, the corresponding element in the information matrix is zero.

A good approximation $\tilde{p}$ is obtained by a term proportional to the product of the marginals, $p(a, c)$ and $p(b, c)$. Neither of these marginals retain dependence between the variables a and b, since they both contain only one of those variables. Thus, the product $p(a, c)\ p(b, c)$ does not contain any *direct* dependencies between a and b; instead, a and b are conditionally independent given c. However, $p(a, c)\ p(b, c)$ is not yet a valid probability distribution over a, b, and c. This is because c occurs twice in this expression. However, proper normalization by $p(c)$ yields a probability distribution (assuming $p(c) > 0$):

$$\tilde{p}(a, b, c) \quad = \quad \frac{p(a, c)\ p(b, c)}{p(c)} \tag{12.20}$$

To understand the effect of this approximation, we apply the following transformation:

$$
\begin{aligned}
\tilde{p}(a, b, c) \quad &= \quad \frac{p(a, b, c)}{p(a, b, c)}\ \frac{p(a, c)\ p(b, c)}{p(c)} \\[2mm]
&= \quad p(a, b, c)\ \frac{p(a, c)}{p(c)}\ \frac{p(b, c)}{p(a, b, c)} \\[2mm]
&= \quad p(a, b, c)\ \frac{p(a \mid c)}{p(a \mid b, c)} \tag{12.21}
\end{aligned}
$$

In other words, removing the direct dependence between a and b is equivalent to approximating the conditional $p(a \mid b, c)$ by a conditional $p(a \mid c)$. We also note (without proof) that among all approximations q of p where a and b are conditionally independent given c, the one described here is "closest" to p, where closeness is measured by the Kullback Liebler divergence, a common (asymmetric) information-theoretic measure of the "nearness" of one probability distribution to another.

An important observation pertains to the fact that the original $p(a \mid b, c)$ is *at least as informative* as $p(a \mid c)$, the conditional hat replaces $p(a \mid b, c)$ in $\tilde{p}$. This is because

$p(a \mid b, c)$ is conditioned on a superset of variables of the conditioning variables in $p(a \mid c)$. For Gaussians, this implies that the variances of the approximation $p(a \mid c)$ is equal or larger than the variance of the original conditional, $p(a \mid b, c)$. Further, the variances of the marginals $\tilde{p}(a)$, $\tilde{p}(b)$, and $\tilde{p}(c)$ are also larger than or equal to the corresponding variances of $p(a)$, $p(b)$, and $p(c)$. In other words, it is impossible that the variance shrinks under this approximation.

12.5.2   Sparsifications in SEIFs

The SEIF applies the idea of sparsification to the posterior $p(y_t \mid z_{1:t}, u_{1:t}, c_{1:t})$, thereby maintaining an information matrix Ω_t that is sparse at all times. This sparseness is at the core of SEIF's efficiency. We already remarked that sparsification is an approximative step, since information matrices in SLAM are naturally not sparse—even though normalized information matrices tend to be almost sparse. In the context of SLAM, it suffices to deactivate links between the robot pose and individual features in the map; if done correctly, this also limits the number of links between pairs of features.

To see, let us briefly consider the two circumstances under which a new link may be introduced. First, observing a passive feature activates this feature, that is, introduces a new link between the robot pose and the very feature. Second, motion introduces links between any two active features. This consideration suggests that controlling the number of active features can avoid violation of both sparseness bounds. Thus, sparseness is achieved simply by keeping the number of active features small at any point in time.

To define the sparsification step, it will prove useful to partition the set of all features into three disjoint subsets:

$$m \;\; = \;\; m^+ \; + \; m^0 \; + \; m^- \tag{12.22}$$

where m^+ is the set of all active features that shall remain active. The set m^0 are one or more active features that we seek to deactivate (remove the link to the robot). And finally, m^- are all currently passive features; they shall remain passive in the process of sparsification. Since $m^+ \cup m^0$ contains all currently active features, the posterior can be factored as follows:

$$
\begin{aligned}
&p(y_t \mid z_{1:t}, u_{1:t}, c_{1:t}) \\
&\quad = \;\; p(x_t, m^0, m^+, m^- \mid z_{1:t}, u_{1:t}, c_{1:t})
\end{aligned}
\tag{12.23}
$$

$$= \quad p(x_t \mid m^0, m^+, m^-, z_{1:t}, u_{1:t}, c_{1:t}) \; p(m^0, m^+, m^- \mid z_{1:t}, u_{1:t}, c_{1:t})$$
$$= \quad p(x_t \mid m^0, m^+, m^- = 0, z_{1:t}, u_{1:t}, c_{1:t}) \; p(m^0, m^+, m^- \mid z_{1:t}, u_{1:t}, c_{1:t})$$

In the last step we exploited the fact that if we know the active features m^0 and m^+, the variable x_t does not depend on the passive features m^-. We can hence set m^- to an arbitrary value without affecting the conditional posterior over x_t, $p(x_t \mid m^0, m^+, m^-, z_{1:t}, u_{1:t}, c_{1:t})$. Here we simply chose $m^- = 0$.

Following the sparsification idea discussed in general terms in the previous section, we now replace $p(x_t \mid m^0, m^+, m^- = 0)$ by $p(x_t \mid m^+, m^- = 0)$, that is, we drop the dependence on m^0.

$$\tilde{p}(x_t, m \mid z_{1:t}, u_{1:t}, c_{1:t}) \tag{12.24}$$
$$= \quad p(x_t \mid m^+, m^- = 0, z_{1:t}, u_{1:t}, c_{1:t}) \; p(m^0, m^+, m^- \mid z_{1:t}, u_{1:t}, c_{1:t})$$

This approximation is obviously equivalent to the following expression:

$$\tilde{p}(x_t, m \mid z_{1:t}, u_{1:t}, c_{1:t}) \tag{12.25}$$
$$= \quad \frac{p(x_t, m^+ \mid m^- = 0, z_{1:t}, u_{1:t}, c_{1:t})}{p(m^+ \mid m^- = 0, z_{1:t}, u_{1:t}, c_{1:t})} \; p(m^0, m^+, m^- \mid z_{1:t}, u_{1:t}, c_{1:t})$$

12.5.3 Mathematical Derivation

In the remainder of this section, we show that the algorithm **SEIF_sparsification** in Table 12.4 implements this probabilistic calculation, and that it does so in constant time. We begin by calculating the information matrix for the distribution $p(x_t, m^0, m^+ \mid m^- = 0)$ of all variables but m^-, and conditioned on $m^- = 0$. This is obtained by extracting the sub-matrix of all state variables but m^-:

$$\Omega_t^0 \quad = \quad F_{x,m^+,m^0} \; F_{x,m^+,m^0}^T \; \Omega_t \; F_{x,m^+,m^0} \; F_{x,m^+,m^0}^T \tag{12.26}$$

With that, the matrix inversion lemma (Table 3.2 on page 44) leads to the following information matrices for the terms $p(x_t, m^+ \mid m^- = 0, z_{1:t}, u_{1:t}, c_{1:t})$ and $p(m^+ \mid m^- = 0, z_{1:t}, u_{1:t}, c_{1:t})$, denoted Ω_t^1 and Ω_t^2, respectively:

$$\Omega_t^1 \quad = \quad \Omega_t^0 - \Omega_t^0 \; F_{m_0} \; (F_{m_0}^T \; \Omega_t^0 \; F_{m_0})^{-1} \; F_{m_0}^T \; \Omega_t^0$$
$$\Omega_t^2 \quad = \quad \Omega_t^0 - \Omega_t^0 \; F_{x,m_0} \; (F_{x,m_0}^T \; \Omega_t^0 \; F_{x,m_0})^{-1} \; F_{x,m_0}^T \; \Omega_t^0 \tag{12.27}$$

Here the various F-matrices are projection matrices that project the full state y_t into the appropriate sub-state containing only a subset of all variables—in analogy to the matrix F_x used in various previous algorithms. The final term in our approximation (12.25), $p(m^0, m^+, m^- \mid z_{1:t}, u_{1:t}, c_{1:t})$, possesses the following information matrix:

$$\Omega_t^3 \quad = \quad \Omega_t - \Omega_t F_x (F_x^T \Omega_t F_x)^{-1} F_x^T \Omega_t \tag{12.28}$$

Putting these expressions together according to Equation (12.25) yields the following information matrix, in which the feature m^0 is now indeed deactivated:

$$
\begin{aligned}
\tilde{\Omega}_t \quad &= \quad \Omega_t^1 - \Omega_t^2 + \Omega_t^3 \\
&= \quad \Omega_t - \Omega_t^0 F_{m_0} (F_{m_0}^T \Omega_t^0 F_{m_0})^{-1} F_{m_0}^T \Omega_t^0 \\
&\quad + \Omega_t^0 F_{x,m_0} (F_{x,m_0}^T \Omega_t^0 F_{x,m_0})^{-1} F_{x,m_0}^T \Omega_t^0 \\
&\quad - \Omega_t F_x (F_x^T \Omega_t F_x)^{-1} F_x^T \Omega_t
\end{aligned}
\tag{12.29}
$$

The resulting information vector is now obtained by the following simple consideration:

$$
\begin{aligned}
\tilde{\xi}_t \quad &= \quad \tilde{\Omega}_t \, \mu_t \\
&= \quad (\Omega_t \, - \, \Omega_t \, + \, \tilde{\Omega}_t) \, \mu_t \\
&= \quad \Omega_t \, \mu_t \, + \, (\tilde{\Omega}_t \, - \, \Omega_t) \, \mu_t \\
&= \quad \xi_t \, + \, (\tilde{\Omega}_t \, - \, \Omega_t) \, \mu_t
\end{aligned}
\tag{12.30}
$$

This completes the derivation of Lines 3 and 4 in Table 12.4.

12.6   AMORTIZED APPROXIMATE MAP RECOVERY

The final update step in SEIFs is concerned with the computation of the mean μ (since the consideration in this section does not depend on the time index t, it is simply omitted for brevity). Before deriving an algorithm for recovering the state estimate μ from the information form, let us briefly consider what parts of μ are needed in SEIFs, and when. SEIFs need the state estimate μ of the robot pose and the active features in the map. These estimates are needed at three different occasions:

1. The mean is used for the linearization of the motion model, which takes place in Lines 3, 4, and 10 in Table 12.2.

2. It is also used for linearization of the measurement update, see Lines 6, 8, 10, 13 in Table 12.3.

3. Finally, it is used in the sparsification step, specifically in Line 4 in Table 12.4.

However, we never need the full vector μ. We only need an estimate of the robot pose, and an estimate of the locations of all active features. This is a small subset of all state variables in μ. Nevertheless, computing these estimates efficiently requires some additional mathematics, as the *exact* approach for recovering the mean via $\mu = \Omega^{-1} \xi$ requires matrix inversion—even when recovering a subset of variables only.

Once again, the key insight is derived from the sparseness of the matrix Ω. In particular the sparseness enables us do define an iterative algorithm that enables us to recover state variables online, as the data is being gathered and the estimates ξ and Ω are being constructed. To do so, it will prove convenient to reformulate $\mu = \Omega^{-1} \xi$ as an *optimization problem*. As we will show in just a minute, the state μ is the mode

$$\hat{\mu} \;\; = \;\; \underset{\mu}{\mathrm{argmax}} \; p(\mu) \tag{12.31}$$

of the following Gaussian distribution, defined over the variable μ:

$$p(\mu) \;\; = \;\; \eta \; \exp \left\{ -\tfrac{1}{2} \, \mu^T \, \Omega \, \mu + \xi^T \, \mu \right\} \tag{12.32}$$

Here μ is a vector of the same form and dimensionality as μ. To see that this is indeed the case, we note that the derivative of $p(\mu)$ vanishes at $\mu = \Omega^{-1} \xi$:

$$\frac{\partial p(\mu)}{\partial \mu} \;\; = \;\; \eta \, (-\Omega \, \mu + \xi) \; \exp \left\{ -\tfrac{1}{2} \, \mu^T \, \Omega \, \mu + \xi^T \, \mu \right\} \;\; \overset{!}{=} \;\; 0 \tag{12.33}$$

which implies $\Omega \, \mu = \xi$ or, equivalently, $\mu = \Omega^{-1} \xi$.

This suggests that recovering the state vector μ is equivalent to finding the mode of (12.32). Thus, it transforms a matrix inversion problem into an optimization problem. For this optimization problem, we will now describe an iterative hill climbing algorithm which, thanks to the sparseness of the information matrix, requires only constant time per optimization update.

Our approach is an instantiation of coordinate descent. For simplicity, we state it here for a single coordinate only; our implementation iterates a constant number K of such optimizations after each measurement update step. The mode $\hat{\mu}$ of (12.32) is attained at:

$$
\begin{aligned}
\hat{\mu} &= \underset{\mu}{\operatorname{argmax}} \ \exp\left\{ -\tfrac{1}{2}\, \mu^T\, \Omega\, \mu + \xi^T\, \mu \right\} \\
&= \underset{\mu}{\operatorname{argmin}} \ \tfrac{1}{2}\, \mu^T\, \Omega\, \mu - \xi^T\, \mu \tag{12.34}
\end{aligned}
$$

We note that the argument of the min-operator in (12.34) can be written in a form that makes the individual coordinate variables μ_i (for the i-th coordinate of μ_t) explicit:

$$
\tfrac{1}{2}\mu^T\, \Omega\, \mu - \xi^T\mu \ = \ \tfrac{1}{2}\sum_i \sum_j \mu_i^T\, \Omega_{i,j}\, \mu_j - \sum_i \xi_i^T\, \mu_i \tag{12.35}
$$

where $\Omega_{i,j}$ is the element with coordinates (i,j) in the matrix Ω, and ξ_i if the i-th component of the vector ξ. Taking the derivative of this expression with respect to an arbitrary coordinate variable μ_i gives us

$$
\frac{\partial}{\partial \mu_i} \left\{ \tfrac{1}{2}\sum_i \sum_j \mu_i^T\, \Omega_{i,j}\, \mu_j - \sum_i \xi_i^T\, \mu_i \right\} \ = \ \sum_j \Omega_{i,j}\, \mu_j - \xi_i \tag{12.36}
$$

Setting this to zero leads to the optimum of the i-th coordinate variable μ_i given all other estimates μ_j:

$$
\mu_i \ = \ \Omega_{i,i}^{-1} \left[\xi_i - \sum_{j \neq i} \Omega_{i,j}\, \mu_j \right] \tag{12.37}
$$

The same expression can conveniently be written in matrix notation. Here we define $F_i = (\, 0 \ldots 0 \ \ 1 \ \ 0 \ldots 0 \,)$ to be a projection matrix for extracting the i-th component from the matrix Ω:

$$
\mu_i \ = \ (F_i\, \Omega\, F_i^T)^{-1}\, F_i\, [\xi - \Omega\, \mu + \Omega\, F_i^T\, F_i\, \mu] \tag{12.38}
$$

This consideration derives our incremental update algorithm. By repeatedly updating

$$\mu_i \quad \longleftarrow \quad (F_i \, \Omega \, F_i^T)^{-1} \, F_i \, [\xi - \Omega \, \mu + \Omega \, F_i^T \, F_i \, \mu] \tag{12.39}$$

for some element of the state vector μ_i reduces the error between the left-hand side and the right-hand side of Equation (12.38). Repeating this update indefinitely for all elements of the state vector converges to the correct mean (without proof).

As is easily seen, the number of elements in the summation in (12.37), and hence the vector multiplication in the update rule (12.39), is constant if Ω is sparse. Hence, each update requires constant time. To maintain the constant-time property of our SLAM algorithm, we can afford a constant number of updates K per time step. This will generally not lead to convergence, but the relaxation process takes place over multiple time steps, resulting in small errors in the overall estimate.

However, a note of caution is in order. The quality of this approximation depends on a number of factors, among them the size of the largest cyclic structure in the map. In general, a constant number of K updates might be insufficient to yield good results. How to best update μ is presently an open question. Also, there exists a number of optimization techniques that are more efficient than the coordinate descent algorithm described here. A "classical" example is conjugate gradeitn. In practical implementations it is advisable to rely on efficient optimization techniques to recover μ.

12.7 HOW SPARSE SHOULD SEIFS BE?

To determine the degree of sparseness one is willing to accommodate in SEIFs, one should compare a sparse SEIF to a SEIF without the sparsification step and with closed-form map recovery. Such a SEIF is functionally equvalent to the EKF SLAM alorithm.

The following comparison characterizes the three key performance measures that set sparse SEIFs apart from EKFs. Our comparison is based on a simulated robot world, in which the robot senses the range, proximity, and identity of nearby landmarks,

1. **Computation.** Figures 12.6 compares the computation per update in SEIFs with that in EKFs; in both cases the implementation is optimized. This graph illustrates the major computational ramification of the probabilistic versus informa-
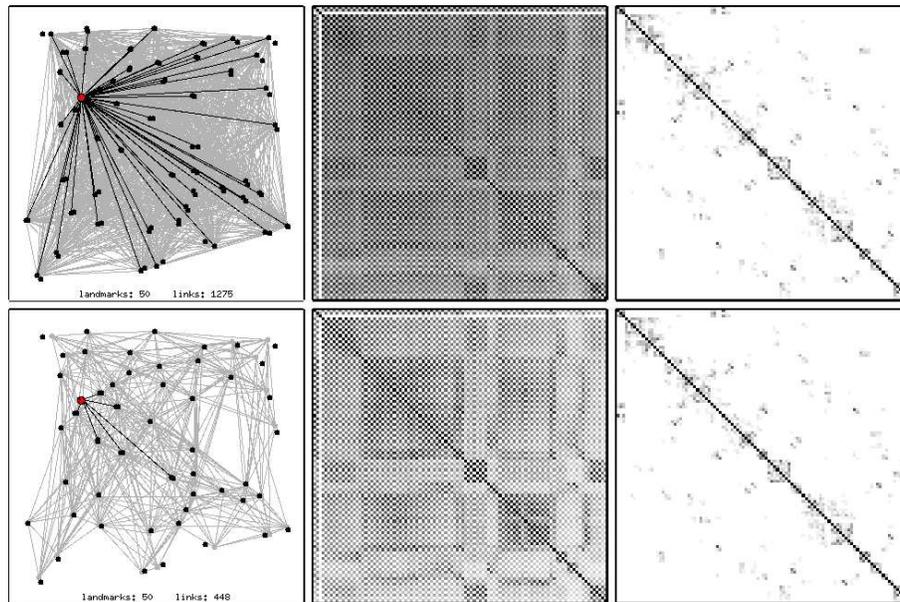
Figure 12.5   Comparison of SEIF without sparsification (top row) with SEIF with 4 active landmarks (bottom row)m in a simulated environment with 50 landmarks. In each row, the left panel shows the set of links in the filter; the center panel the correlation matrix; and the right panel the noramlized information matrix. Obviously, the sparsified SEIF maintains many fewer links, but its result is less confident as indicated by its less-expressed correlation matrix.

tion representation in the filter. While EKFs indeed require time quadratic in the map size, SEIFs level off and require constant time. .

2. **Memory.** Figure 12.7 compares the memory use of EKFs with that of SEIFs. Here once again, EKFs scale quadratically, whereas SEIFs scale linearly, due to the sparseness of its information representation.

3. **Accuracy.** Here EKFs outperform SEIFs, due to the fact that SEIFS require approximation for maintaining sparseness, and when recovering the state estimate μ_t. This is shown in Figure 12.8, which plots the error of both methods as a function of map size.

The number of acitve features in SEIFs determines the degree of sparseness. This effective trades off two factors: the computational efficiency of the SEIF, and the accuracy of the result. When implementing a SEIF algorithm, it is therefore advisable to get a feeling for this trade-off.
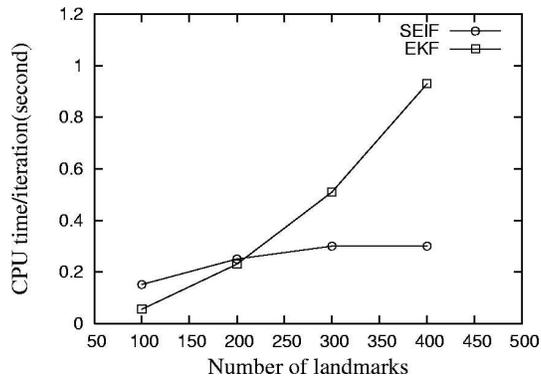
Figure 12.6   The comparison of average CPU time between SEIF and EKF.
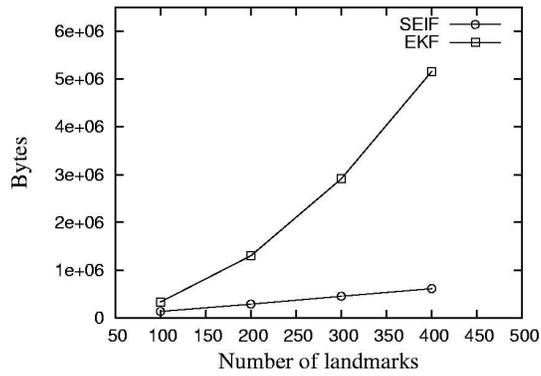
Figure 12.7   The comparison of average memory usage between SEIF and EKF.
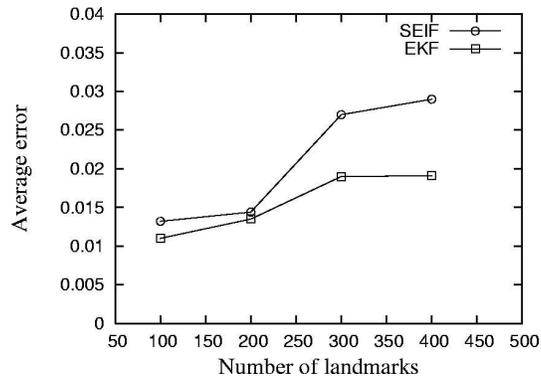
Figure 12.8   The comparison of root mean square distance error between SEIF and EKF.
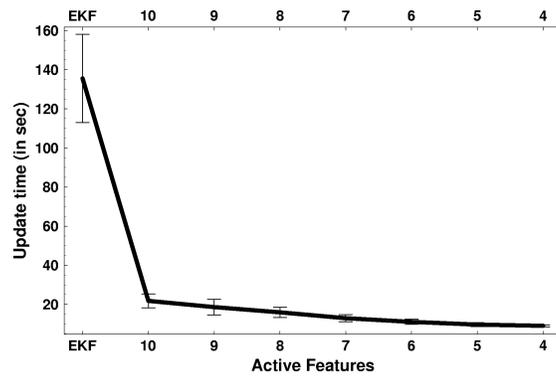
Figure 12.9 The update time of the EKF (leftmost data point only) and the SEIF, for different degrees of sparseness, as induced by a bound on the number of active features as indicated.
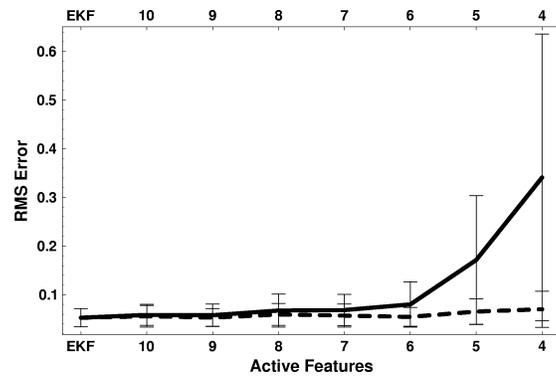
Figure 12.10 The approximation error EKF (leftmost data point only) and SEIF for different degrees of sparseness. In both figures, the map consists of 50 landmarks.

One way to get a feeling for the effect of the degree of sparseness can be obtained via simulation. Figure 12.9 plots the update time and the approximation error as a function of the number of active landmarks in the SEIF update, for a map consisting of 50 landmarks. The update time falls monotonically with the number of active features. Figure 12.10 shows the corresponding plot for the error, comparing the EKF with the SEIF at different degrees of sparseness. The solid line is the SEIF as described, whereas the dashed line corresponds to a SEIF that recovers the mean μ_t exactly, via matrix inversion and multiplication. As this plot suggests, 6 active features seem to provide competitive results, at significant computational savings over the EKF. For smaller numbers of active features, the error increases drastically. A careful implementation of SEIFs will require the experimento to vary this important parameter, and graph its effect on key factors as done here.

12.8 INCREMENTAL DATA ASSOCIATION

We will now turn our attention to the problem of data association in SEIFs. Our first technique will be the familiar incremental approach,which greedily identifies the most liekly correspondence, and then treat this value as if it was ground truth. We already encountered an instance of such a greedy data association technique in Chapter 10.3, where we discussed data association in the EKF. In fact, the only difference between greedy incremental data associaiton in SEIFs and EKSs pertains to the calculation of the data association probability. As a rule of thumb, computing this probability is generally more difficult in an information filter than in a probabilistic filter such as the EKF, since the information filter does not keep track of covariances.

12.8.1 Computing Data Association Probabilities

As before, the data association vector at time will be denoted c_t. The greedy incremental technique maintains a set of data assiciation guesses, denoted $\hat{c}_{1:t}$. At time t, we are already given from previous updates a set $\hat{c}_{1:t-1}$. The data associaton step then pertains to the estimation of the most likely value for the data association variable $\hat{c}_t$ at time t. This is achieved via the following maximum likelihood estimator:

$$
\begin{aligned}
\hat{c}_t &= \underset{c_t}{\operatorname{argmax}}\, p(z_t \mid z_{1:t-1}, u_{1:t}, \hat{c}_{1:t-1}, c_t) \\
&= \underset{c_t}{\operatorname{argmax}} \int p(z_t \mid y_t, c_t)\, \underbrace{p(y_t \mid z_{1:t-1}, u_{1:t}, \hat{c}_{1:t-1})}_{\bar{\Omega}_t, \bar{\xi}_t}\, dy_t \\
&= \underset{c_t}{\operatorname{argmax}} \int \int p(z_t \mid x_t, y_{c_t}, c_t)\, p(x_t, y_{c_t} \mid z_{1:t-1}, u_{1:t}, \hat{c}_{1:t-1}) \quad (12.40)
\end{aligned}
$$

Our notation $p(z_t \mid x_t, y_{c_t}, c_t)$ of the sensor model makes the correspondence variable c_t explicit. Calculating this probability exactly is not possible in constant time, since it involves marginalizing out almost all variables in the map (which requires the inversion of a large matrix). However, the same type of approximation that was essential for the efficient sparsification can also be applied here as well.

In particular, let us denote by $m_{c_t}^+$ the combined Markov blanket of the robot pose x_t and the landmark y_{c_t}. This Markov blanket is the set of all features in the map that
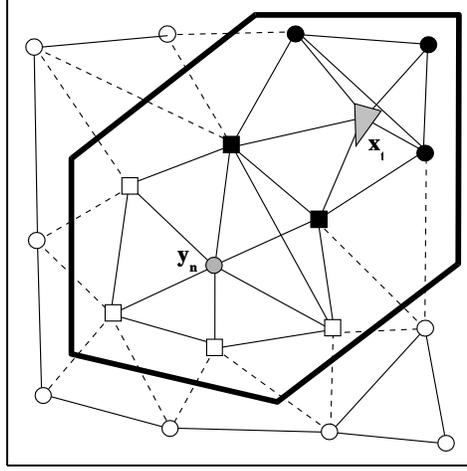
Figure 12.11 The combined Markov blanket of feature y_n and the obsrved features is usually suffi cient for approximating the posterior probability of the feature locations, conditioning away all other features.

are linked to the robot of landmark y_{c_t}. Figure 12.11 illustrates this set. Notice that $m_{c_t}^+$ includes by definition all active landmarks. The spareness of $\bar{\Omega}_t$ ensures that $m_{c_t}^+$ contains only a fixed number of features, regardless of the size of the map N. If the Markov blanket of x_t and of y_{c_t}, further features are added that represent the shortest path in the information graph between x_t and of y_{c_t}.

All remaining features will now be collectively referred to as $m_{c_t}^-$, that is:

$$m_{c_t}^- \;\; = \;\; m - m_{c_t}^+ - \{y_{c_t}\} \tag{12.41}$$

The set $m_{c_t}^-$ contains only features which have only a minor impact on the target variables, x_t and y_{c_t}. Our approach approximates the probability $p(x_t, y_{c_t} \mid z_{1:t-1}, u_{1:t}, \hat{c}_{1:t-1})$ in Equation (12.40) by essentially ignoring these indirect influences:

$$
\begin{aligned}
&p(x_t, y_{c_t} \mid z_{1:t-1}, u_{1:t}, \hat{c}_{1:t-1}) \\
&= \int \int p(x_t, y_{c_t}, m_{c_t}^+, m_{c_t}^- \mid z_{1:t-1}, u_{1:t}, \hat{c}_{1:t-1}) \, dm_{c_t}^+ \, dm_{c_t}^- \\
&= \int \int p(x_t, y_{c_t} \mid m_{c_t}^+, m_{c_t}^-, z_{1:t-1}, u_{1:t}, \hat{c}_{1:t-1}) \, p(m_{c_t}^+ \mid m_{c_t}^-, z_{1:t-1}, u_{1:t}, \hat{c}_{1:t-1})
\end{aligned}
$$

$$p(m_{c_t}^- \mid z_{1:t-1}, u_{1:t}, \hat{c}_{1:t-1}) \; dm_{c_t}^+ \; dm_{c_t}^- \qquad\qquad (12.42)$$

$$\approx \int p(x_t, y_{c_t} \mid m_{c_t}^+, m_{c_t}^- = \mu_{c_t}^-, z_{1:t-1}, u_{1:t}, \hat{c}_{1:t-1})$$

$$p(m_{c_t}^+ \mid m_{c_t}^- = \mu_{c_t}^-, z_{1:t-1}, u_{1:t}, \hat{c}_{1:t-1}) \; dm_{c_t}^+$$

This probability can be computed in constant time if the set of features considered in this calculation is independent of the map size (which it generally is). In complete analogy to various derivations above, we note that the approximation of the posterior is simply obtained by carving out the submatrix corresponding to the two target variables:

$$
\begin{aligned}
\Sigma_{t:c_t} &= F_{x_t, y_{c_t}}^T \; (F_{x_t, y_{c_t}, m_{c_t}^+}^T \; \Omega_t \; F_{x_t, y_{c_t}, m_{c_t}^+})^{-1} \; F_{x_t, y_{c_t}} \\
\mu_{t:c_t} &= \mu_t F_{x_t, y_{c_t}}
\end{aligned}
\qquad (12.43)
$$

This calculation is constant time, since it involves a matrix whose size is independent of N. From this Gaussian, the desired measurement probability in Equation (12.40) is now easily recovered, as described in Section **??**.

As in our EKF SLAM algorithm, features are labeled as new when the likelihood $p(z_t \mid z_{1:t-1}, u_{1:t}, \hat{c}_{1:t-1}, c_t)$ remains below a threshold α. We then simply set $\hat{c}_t = C_{t-1} + 1$ and $C_t = C_{t-1} + 1$. Otherwise the size of the map remains unchanged, that is, $C_t = C_{t-1}$ and the value $\hat{c}_t$ is chosen that maximizes the data association probebility.

As last caveat, sometimes the combines Markov blanket is insufficient, in that it does not contain a path between the robot pose and the landmark that is being tested for corresponence. This will usually xbe the case when closing a large cycle in the environment. Here we need to augment the set of features $m_{c_t}^+$ by a set of landmarks along at least one path between m_{c_t} and the robot pose x_t. Depending on the size of thie cycle, the numbers of landmarks contained in the resulting set may now depend on N, the size of the map. We leave the details of such an extension as an exercise.

12.8.2   Practical Considerations

In general, the incremental greedy data association technique is brittle. Spurious measurements can easily cause false associations, and induce significant errors into the SLAM estimate.

Figure 12.12 The vehicle used in our experiments is equipped with a 2D laser range finder and a differential GPS system. The vehicle's ego-motion is measured by a linear variable differential transformer sensor for the steering, and a wheel-mounted velocity encoder. In the background, the Victoria Park test environment can be seen.

The standard approach—in EKFs and SEIFs alike—-pertains to the creation of a *candidate list*. For any detected object that can not be explained by existing features, a new feature candidate is generated but not put into SEIF directly. Instead it is added into the candidate list with a weight representing its probability of being a useful feature. In the next measurement step, the newly arrived candidates are checked against all candidates in the waiting list; reasonable matches increase the weight of corresponding candidates. Candidates that are not matched lose weight because they are more likely to be a moving object. When a candidate has its weight above a certain threshold, it joins the SEIF network of features.

We notice that data association violates the constant time property of SEIFs. This is because when calculating data associations, multiple features have to be tested. If we can ensure that all plausible features are already connected in the SEIF by a short path to the set of active features, it would be feasible to perform data association in constant time. In this way, the SEIF structure naturally facilitates the search of the most likely feature given a measurement. However, this is not the case when closing a cycle for the first time, in which case the correct association might be far away in the SEIF adjacency graph.

The remainder of this section describes an implementation of the SEIF algorithm using a physical vehicle. The data used here is a common benchmark in the SLAM field [11, 23, 27]. This data set was collected with an instrumented outdoor vehicle driven through a park in Sydney, Australia.

Figure 12.13   The testing environment: A 350 meters by 350 meters patch in Victoria Park in Sydney. Overlayed is the integrated path from odometry readings.

Figure 12.14   The path recovered by the SEIF, is correct within ± 1m.
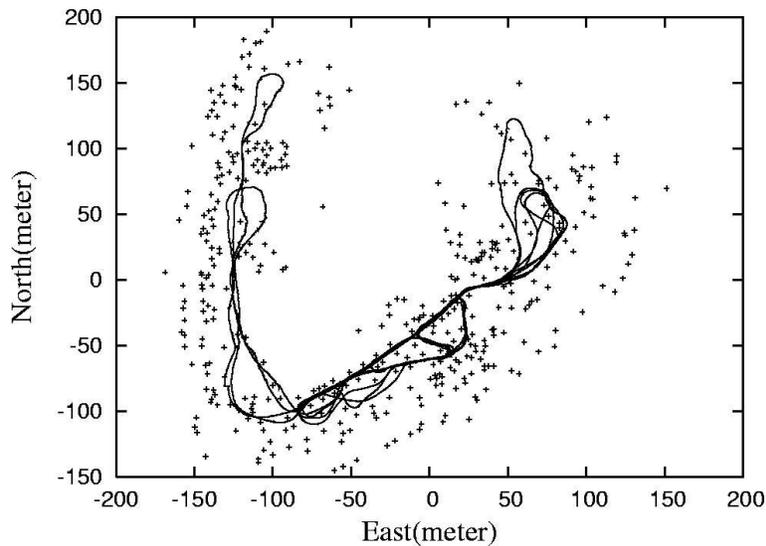
Figure 12.15 Overlay of estimated landmark positions and robot path.

The vehicle and its environment are shown in Figures 12.12 and 12.13, respectively. The robot is equipped with a SICK laser range finder and a system for measuring steering angle and forward velocity. The laser is used to detect trees in the environment, but it also picks up hundreds of spurious features such as corners of moving cars on a nearby highway. The raw odometry, as used in our experiments, is poor, resulting in several hundred meters of error when used for path integration along the vehicle's 3.5km path. This is illustrated in Figure 12.13, which shows the path of the vehicle. The poor quality of the odometry information along with the presence of many spurious features make this dataset particularly amenable for testing SLAM algorithms.

The path recovered by the SEIF is shown in Figure 12.14. This path is quantitatively indistinguishable from the one produced by the EKF. The average position error, as measured through differential GPS, is smaller than 0.50 meters, which is small compared to the overall path length of 3.5 km. Comparing with EKF, SEIF runs approximately twice as fast and consumes less than a quarter of the memory EKF uses.
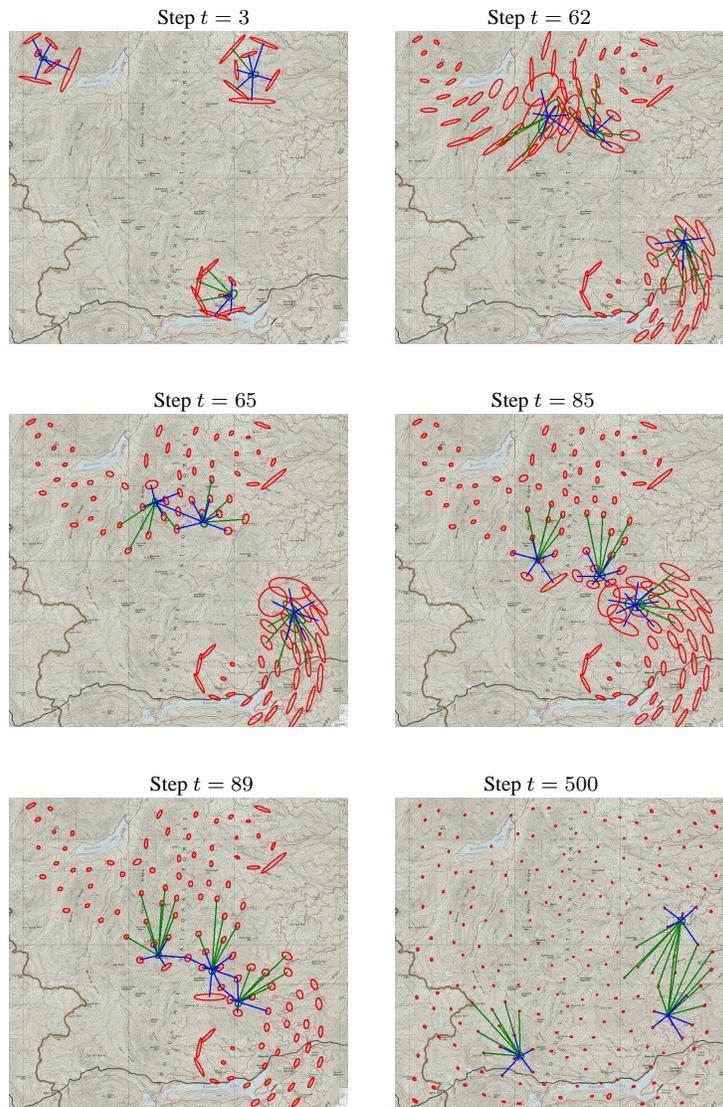
Figure 12.16 Snapshots from our multi-robot SLAM simulation at different points in time. Initially, the poses of the vehicles are known. During Steps 62 through 64, vehicle 1 and 2 traverse the same area for the first time; as a result, the uncertainty in their local maps shrinks. Later, in steps 85 through 89, vehicle 2 observes the same landmarks as vehicle 3, with a similar effect on the overall uncertainty. After 500 steps, all landmarks are accurately localized.

12.9 TREE-BASED DATA ASSOCIATION

SEIFs make it possible to define a radically different data association approach, which can be proven to yield the optimal results (although possibly in exponential time). The technique is built on three key insights:

- Just like the EIF studied in the previous chapter, SEIFs make is possible to add "soft" data association constraints. Given two features m_i and m_i, a soft data association constraints is nothing else but an information link that forces the distance between m_i and m_j to be small. We already encountered examples of such soft links in the previous chapter. In sparse extended information filters, introducing such a link is a simple, local addition of values in the information matrix.

- We can also easily *remove* soft asociation constraints. Just as introducing a new constraints amounts to a local addition in the information matrix, removing it is nothing else but a a local subtraction. Such an "undo" operation can be applied to arbitrary data association links, regardless when they were added, or when the respective feature was last observed. This makes it possible to revise past data association decisions.

- The ability to freely add and subtract data associations arbitrarily enables us to search the tree of possible data associations in a way that is both efficient and complete—as will be shown below.

To develop tree-based data association, it shall prove useful to consider the data association tree that is defined over sequences of data association decisions over time. At each point in time, each observed feature can be associated with a number of other features, or considered a new, previously unobserved feature. The resulting tree of data association choices, starting at time $t = 1$ all the way to the present time, is illustrated in Figure 12.17a. Of course, the tree grows exponentially over time, hence searching it exhaustively is impossible. The greedy approach described in the previous section, in contrast, follows one path through this tree, defined by the locally most likely data associations. Such a path is visualized in Figure 12.17a as the thick gray path.

Obviously, if the incremental greedy approach succeeds, the resulting path is optimal. However, the incremental greedy technique may fail. Once a wrong choice has been made, the incremental approach cannot recover. Moreover, wrong data association decisions introduce errors in the map which, subsequently, can induce more errors in the data association.

The approach discussed in the remainder of this chapter generalizes the incremental greedy algorithm into a full-blown search algorithm for the tree that is provably op-

timal. Of course, following all branches in the tree is intractable. However, if we maintain the log-likelihood of all nodes on the *fringe* of the tree extracted thus far, we can guarantee optimality. Figure 12.17b illustrates this idea: The tree-based SEIF maintains not just a single path through the data association tree, but an entire frontier. Every time a node is expanded (e.g., through incremental ML), all alternative outcomes are also assessed and the corresponding likelihoods are memorized. This is illustrated in Figure 12.17b, which depicts the log-likelihood for an entire frontier of the tree.

Finding the maximum in Equation (12.40) implies that the log-likelihood of the chosen leaf is greater or equal to that of any other leaf at the same depth. Since the log-likelihood decreases monotonically with the depth of the tree, we can guarantee that we indeed found the optimal data association values when the log-likelihood of the chosen leaf is greater or equal to the log-likelihood of any other node on the frontier. Put differently, when a frontier node assumes a log-likelihood greater than the one of the chosen leaf, there might be an opportunity to further increase the likelihood of the data by revising past data association decisions. Our approach then simply expands such frontier nodes. If an expansion reaches a leaf, this leaf is chosen as the new data association; otherwise the search is terminated when the entire frontier possesses values that are all smaller or equal to the one of the chosen leaf. This approach is guaranteed to always maintain the best set of values for the data association variables; however, occasionally it might require substantial search.

12.9.1   Calculating Data Association Probaiblities

We have now, on numerous occasions, discussed techniques for calculating data association probabilities. In our case we need a technique which, for any two features in the map, calculates the probability of equality. This is different from previous filter techniques, in which we only considered the most recent measurements, but the mathematical derivation is essentially the same.

Table 12.6 lists an algorithm that tests the probability that two features in the map are one and the same—this test is sufficient to implement the greedy data association. The key calculation here pertains to the recovery of a joint covariance and mean vector over a small set of map elements B. To determine whether two features in the map are identical, we need to consider the information links between them. Technically, the more links are included in this consideration, the more accrate the result, but at the expense of increased compuation. In practice, it usually suffices to identify the two *Karkov blankets* of the features in question. A Markov blanket of a feature is the
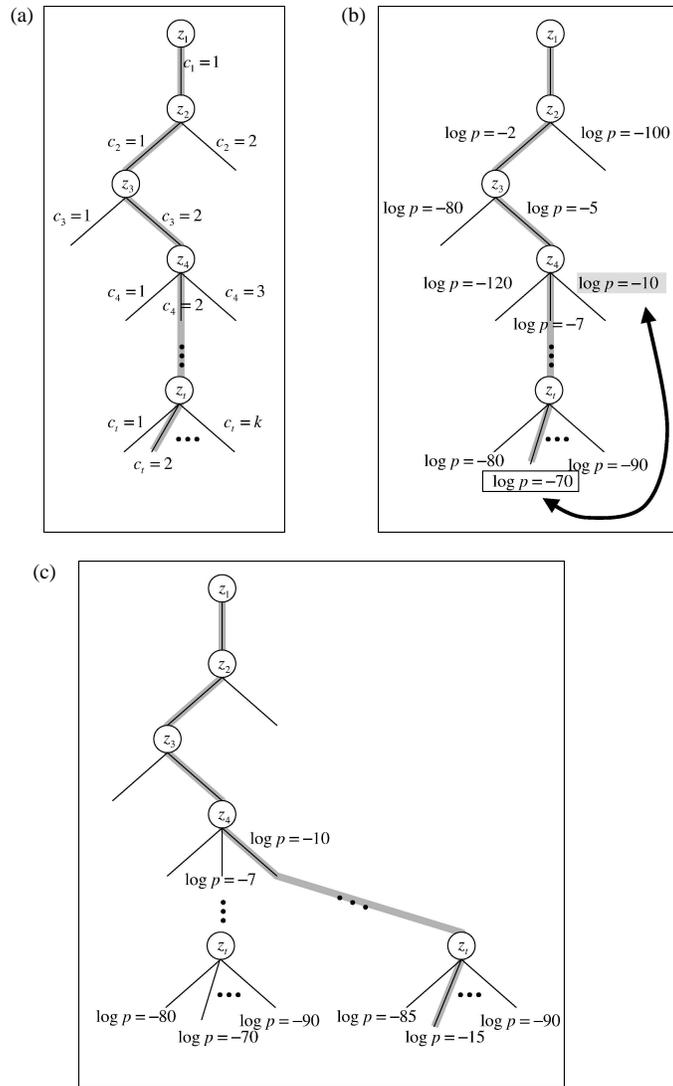
Figure 12.17 (a) The data association tree, whose branching factor grows with the number of landmarks in the map. The tree-based SEIF maintains the log-likelihood for the entire frontier of expanded nodes, enabling it to find alternative paths. (c) Improved path.

1:          **Algorithm SEIF_correspondence_test($\Omega, \xi, \mu, m_j, m_k$):**
1:              *let $B(j)$ be the blaket of m_j*
1:              *let $B(k)$ be the blaket of m_k*
1:              $B = B(j) \cup B(k)$
1:              *if $B(j) \cap B(k) = \emptyset$*
1:                  *add features along the shortest path between m_i and m_j to B*
1:              *endif*

1:          $F_B = \begin{pmatrix} 0\cdots0 & 1 & 0 & 0\cdots0 & & \cdots & \\ 0\cdots0 & 0 & 1 & 0\cdots0 & & \cdots & \\ & \cdots & & 0\cdots0 & 1 & 0 & 0\cdots0 \\ & \cdots & & 0\cdots0 & 0 & 1 & 0\cdots0 \\ & & & & & & \ddots \\ & & \cdots & & & & 0\cdots0 \\ & & \cdots & & & & 0\cdots0 \end{pmatrix}$

1:                          *(size $(2N+3)$ by $2|B|$)*

1:          $\Sigma_B = (F_B \ \Omega \ F_B^T)^{-1}$
1:          $\mu_B = \Sigma_B \ F_B \ \xi$

1:          $F_\Delta = \begin{pmatrix} 0\cdots0 & 1 & 0 & 0\cdots0 & -1 & 0 & 0\cdots0 \\ 0\cdots0 & \underbrace{0 \ 1}_{\text{feature } m_j} & & 0\cdots0 & \underbrace{0 \ -1}_{\text{feature } m_j} & & 0\cdots0 \end{pmatrix}$

1:          $\Sigma_\Delta = (F_\Delta \ \Omega \ F_\Delta^T)^{-1}$
1:          $\mu_\Delta = \Sigma_\Delta \ F_\Delta \ \xi$

1:          *return* $\det(2\pi \ \Sigma_\Delta)^{-\frac{1}{2}} \ \exp\{-\frac{1}{2} \ \mu_\Delta^T \ \Sigma_\Delta^{-1} \ \mu_\Delta\}$

Table 12.6   The SEIF SLAM test for correspondence.

feature itself, and all other features that are connected via a non-zero element in the information matrix. In most cases, both Markov blankets intersect; if they do not, the algoithm in Table 12.6 identifies a path between the landmarks (which must exis if both were observed by the same robot).

The algotihm in Table 12.6 then proceeds by cutting out a local information matrix and informaiton vector, employing the very same mathamtical "trick" thta led to an efficient sparsification step: we condition away features outside the Markov blankets.

As a result, we obtain an efficient technique for calculating the desired probability, one that is approximate (because of the conditioning), but works very well in practoce.

The result is interesting in two dimensions. First, as before, it les us make a data association decision. But second, it provides the likelihood of this decision. The logarithm of the result of this procedure corresponds to the log-likelihood of this specific data item, and summing those up along the path in the data associaiton tree becomes the total dta log-likelihood under a specific association.

12.9.2   Tree Search

The tree-based data association technique uses a search procedure for considering alternative data association decisions not just at the present time step, but also for time steps in the past. A simple argument (reminiscent of that underlying the correctness of the A* algorithm [32]) enables us to drastically reduce the number of nodes expended during this search. Figure 12.17b illustrates the basic idea: Our approach maintains not just a single path through the data association tree, but an entire frontier. Every time a node is expanded (e.g., through incremental ML), all alternative outcomes are also assessed and the corresponding likelihoods are memorized. This is illustrated in Figure 12.17b, which depicts the log-likelihood for an entire frontier of the tree. Notice that we chose to represent the likelihood values as log-likelihoods, which is numerically more stable than probabilities.

The goal of the tree search is to maximize the overall data likelihood

$$\hat{c}_{1:t} \quad = \quad \operatorname*{argmax}_{c_{1:t}} p(z_{1:t}\Theta \mid u_{1:t}, c_{1:t}) \tag{12.44}$$

Notice the difference to Equation (12.40), which only maximizes the most recent data association. Finding the maximum in Equation (12.44) implies that the log-likelihood of the chosen leaf is greater or equal to that of any other leaf at the same depth. Since the log-likelihood decreases monotonically with the depth of the tree, we can guarantee that we indeed found the optimal data association values when the log-likelihood of the chosen leaf is greater or equal to the log-likelihood of any other node on the frontier. Put differently, when a frontier node assumes a log-likelihood greater than the one of the chosen leaf, there might be an opportunity to further increase the likelihood of the data by revising past data association decisions. Our approach then simply expands such frontier nodes. If an expansion reaches a leaf, this leaf is chosen as the new data association; otherwise the search is terminated when the entire frontier possesses values that are all smaller or equal to the one of the chosen leaf. This approach is

guaranteed to always maintain the best set of values for the data association variables; however, occasionally it might require substantial search.

12.9.3   Equivalency Constraints

Once two features in the map have determined to be equivalent in the data association search, we have to add a soft link. Suppose the first feature is m_i and the second is m_j. The soft link constraints their position to be equal through the following exponential-quadratic constraint

$$\exp\left\{-\tfrac{1}{2}\,(m_i - m_j)^T\,C\,(m_i - m_j)\right\} \tag{12.45}$$

Here C is a diagonal penalty matrix. The larger the elements on the diagonal of C, the stronger the constraint. It is easily seen that this non-normalized Gaussian can be written as a link between m_i and m_j in the information matrix. Simplyu define the projection matrix

$$F_{m_i - m_j} \;\; = \;\; \begin{pmatrix} 0\cdots0 & 1 & 0 & 0\cdots0 & -1 & 0 & 0\cdots0 \\ 0\cdots0 & \underbrace{0 \;\; 1}_{m_i} & 0\cdots0 & \underbrace{0 \;\; -1}_{m_j} & 0\cdots0 \end{pmatrix} \tag{12.46}$$

This matrix mapes the state y_t to the difference $m_i - m_j$. Thus, the expression (12.45) becomes

$$\exp\left\{-\tfrac{1}{2}\,(F_{m_i - m_j}\,y_t)^T\,C\,(F_{m_i - m_j}\,y_t)\right\} \;\; = \;\; y_t^T\,F_{m_i - m_j}^T\,C\,F_{m_i - m_j}\,y_t \tag{12.47}$$

Thus, to implement this soft constraint, we have to add $F_{m_i - m_j}^T\,C\,F_{m_i - m_j}$ to the information matrix, while leaving the information vector unchanged:

$$\Omega_t \;\; \longleftarrow \;\; \Omega_t + F_{m_i - m_j}^T\,C\,F_{m_i - m_j} \tag{12.48}$$

Clearly, the additive term is sparse: it only contains non-zero off-diagonal elements between the features m_i and m_j. Once a soft link has been added, it can be removed by the inverse operation

$$\Omega_t \;\; \longleftarrow \;\; \Omega_t - F_{m_i - m_j}^T\,C\,F_{m_i - m_j} \tag{12.49}$$
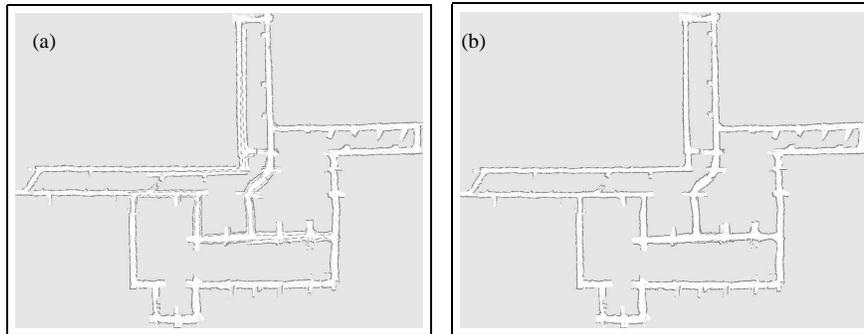
Figure 12.18   (a) Mine map with incremental ML scan matching and (b) using our lazy data association approach. The map is approximately 250 meters wide, and acquired without odometry information.

In SEIFs, This removal can occur even if the constraint was added at a previous time step. However, careful bookkeeping is necessary to guarantee that we never remove a non-existend data associaiton constraints—otherwise the information matrix may no longer be positive semidefinite, and the resulting belief might not correspond to a valid probability distribution.

12.9.4   Practical Considerations

In any competitive implementation of this approach, there will usually only a small number of data association paths that are plausible at any point in time. When closing a loop in an indoor environment, for example, there are usually at most three plausible hypothesis: a closure, a continuation on the left, and a continuation on the right. All but all quickly should become unlikely, so the number of times in which the tree is searched recursively should be small.

One way to make the data association succeed more often is to incorporate negative measurement information. Negative information pertains to situations where a robot fails to see a measurement. Range sensors, which are brought to bear in our implementation, return positive and negative information with regards to the presence of objects in the world. The positive information are object detections. The negative information applies to the space between the detection and the sensor. The fact that the robot failed to detect an object closer than its actual reading provides information about the *absence* of an object within the measurement range.
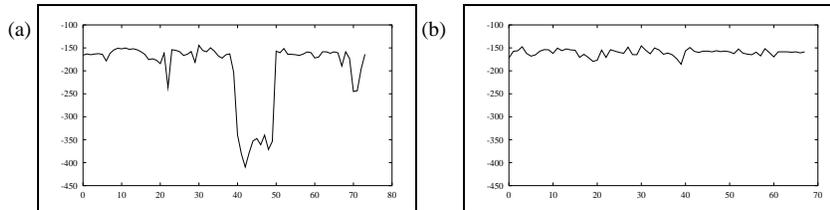
Figure 12.19 (a) Log-likelihood of the actual measurement, as a function of time. The lower likelihood is caused by the wrong assignment. (b) Log-likelihood, when recursively fixing false data association hypotheses through the tree search. The success is manifested by the lack of a distinct dip.
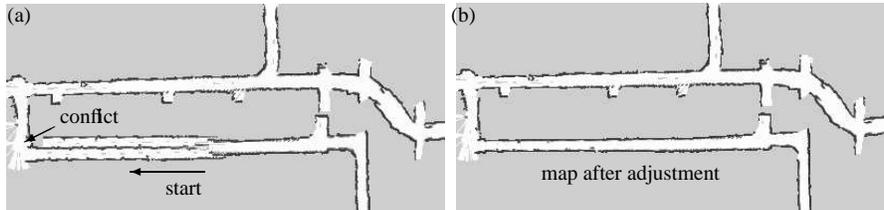
Figure 12.20 Example of the tree-based data association technique: (a) When closing a large loop, the robot first erroneously assumes the existence of a second, parallel hallway. However, this model leads to a gross inconsistency as the robot encounters a corridor at a right angle. At this point, the approach recursively searches for improved data association decisions, arriving on the map shown in diagram (b).

An approach that evaluate the effect of a new constraint on the overall likelihood considers both types of information: positive and negative. Both types are obtained by calculating the pairwise (mis)match of two scans under their pose estimate. When using range scanners, one way to obtain a combination of positive and negative information is by superimposing a scan onto a local occupancy grid map build by another scan. In doing so, it is straightforward to determine the probability of a measurement in a way that incorporates both the positive and the negative information.

The remainder of this section highlights practical results achieved using SEIFs with tree-based data association. The left panel of Figure 12.18a depicts the result of incremental ML data association, which is equivalent to regular incremental scan matching. Clearly, certain corridors are represented doubly in this map, illustrating the shortcomings of the ML approach. The right panel, in comparison, shows the result. Clearly, this map is more accurate than the one generated by the incremental ML approach. Its diameter is approximately 250 meters wide, and the floor of the mine is highly uneven.
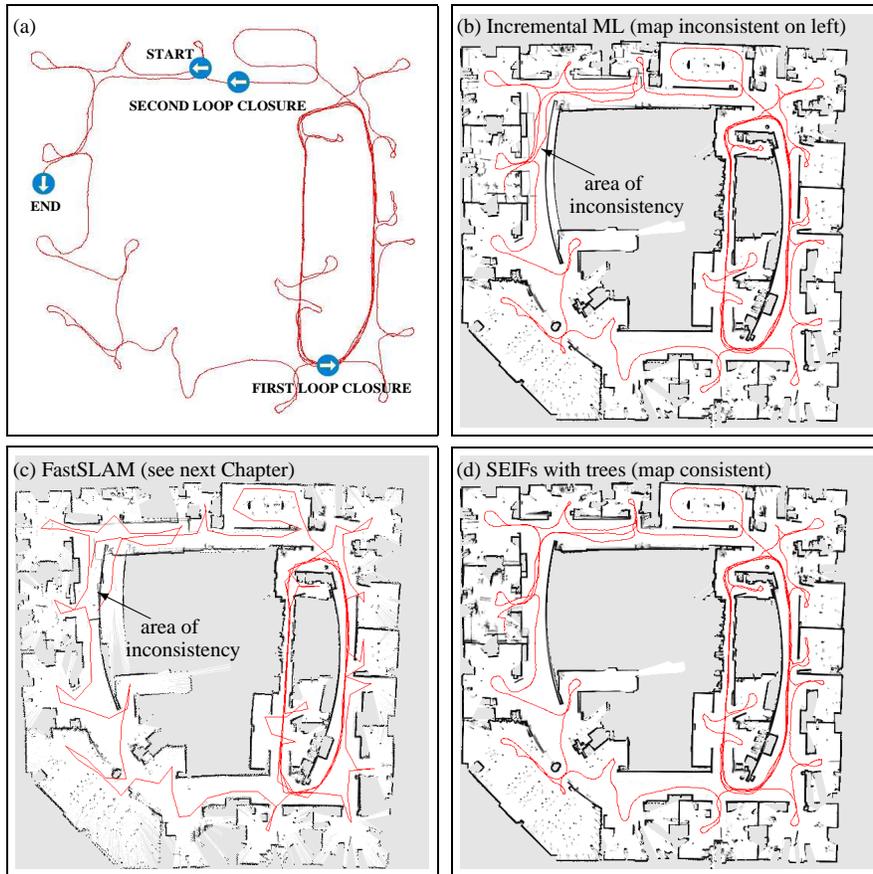
Figure 12.21   (a) Path of the robot. (b) Incremental ML (scan matching) (c) FastSLAM.
(d) Our approach.

Figure 12.19a illustrates the log-likelihood of the most recent measurement (not the entire path), which drops significantly as the map becomes inconsistent. At this point, the SEIF engages in searching alternative data association values. It quickly finds the "correct" one and produces the map shown in Figure 12.18b. The area in question is shown in Figure 12.20, illustrating the moment at which the likelihood takes its dip. The log-likelihood of the measurement is shown in Figure 12.19b.

12.10   MULTI-VEHICLE SLAM

The SEIF is also applicable to multi-robot SLAM problem. The multi-robot SLAM problem involves several robots that independently explore and map an enviroment, with the eventual goal of integrating their maps into a single, monolithic map? This problem raises two key questions: First, how can robot teams establish correspondence between individual robot maps. Second, once correspondence has been established, what are the mechanics of integrating maps?

12.10.1   Fusing Maps Acquired by Multiple Robots

Let Ω_t^k, ξ_t^k and Ω_t^j, ξ_t^j two local estimates (maps and vehicle poses) acquired by two different vehicles, k and j. To fuse these maps, we need two pieces of information: a relative coordinate transformation between these two maps (translation and rotation), and a correspondence list, that is, a list of pairs of landmarks that correspond to each other in the different maps.

Suppose we are given the translation d and the rotation matrix r that specify the co-ordinate transformation from the j-th to the k-th robot's coordinate system—we will discuss our approach to finding d and r further below. Coordinates y in the j-th robot's coordinate system are mapped into the k-th coordinate system via the linear equation $y^{k \leftarrow j} = ry + d$. This transformation is easily extended to the filter variables $\langle \Omega_t^j, \xi_t^j \rangle$

$$\Omega_t^{k \leftarrow j} = R^T \, \Omega_t^j \, R \tag{12.50}$$

$$\xi_t^{k \leftarrow j} = (\xi_t^j + \Omega_t^j \, D^T) \, R^T \tag{12.51}$$

where R and D are matrices that extend r and d to the full dimension of the posterior maintained by the j-th robot:

$$R = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & r & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & r \end{pmatrix} \quad \text{and} \quad D = \begin{pmatrix} \alpha \\ d \\ \vdots \\ d \end{pmatrix} \tag{12.52}$$

Notice the special provision for the robot's heading direction, which is the very first element in the state vector. The heading simply changes by the angle of the rotation between both maps, denoted α in (12.52).
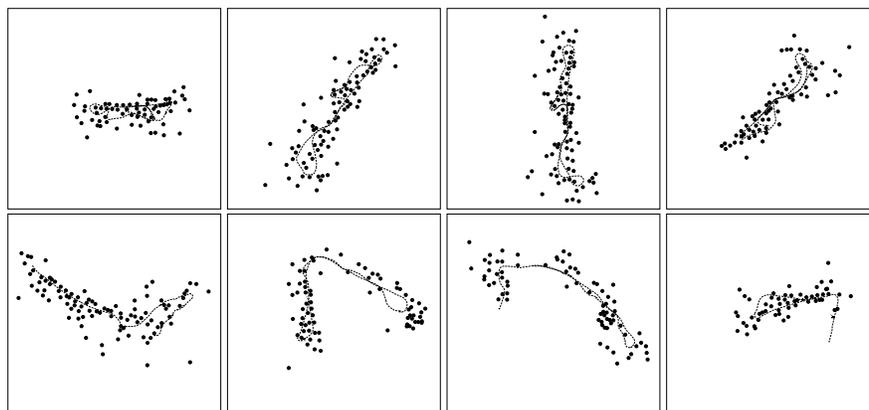
Figure 12.22   Eight local maps obtained by splitting the data into eight disjoint sequences.
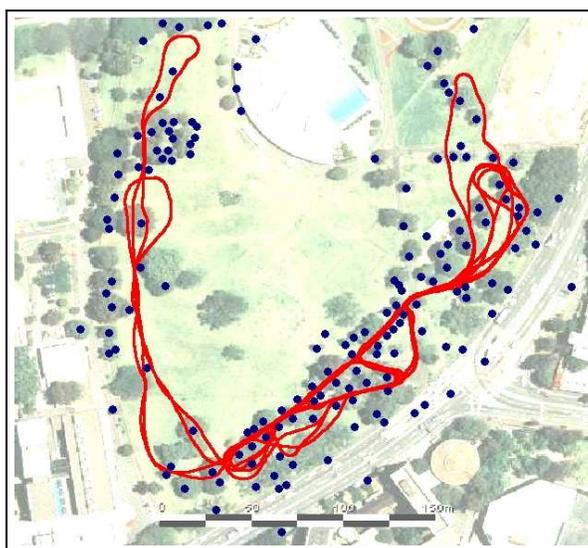
Figure 12.23   the multi-robot result, obtained using the algorithm described in this paper.

To see the correctness of (12.50) and (12.51), we recall that the parameters $\langle \Omega_t^j, \xi_t^j \rangle$ define a Gaussian over the j-th robot pose ans map $\mathbf{x_t^j} = (\ \mathbf{x_t^j} \ \ \mathbf{Y} \ )^{\mathbf{T}}$. This gives us the following derivation:

$$
\begin{aligned}
p(\mathbf{x_t^j} \mid \mathbf{Z^j}, \mathbf{U^j}) & \\
\propto \ & \exp{-\tfrac{1}{2}(R \ \mathbf{x_t^j} - \mathbf{D} - \mu_t^j)^{\mathbf{T}} \ \Sigma_t^{j,-1} \ (\mathbf{R} \ \mathbf{x_t^j} - \mathbf{D} - \mu_t^j)} \\
\propto \ & \exp{-\tfrac{1}{2}\mathbf{x_t^{j,T}} \ \mathbf{R^T} \ \Sigma_t^{j,-1} \ \mathbf{R} \ \mathbf{x_t^j} - (\mu_t^j + \mathbf{D})^{\mathbf{T}} \ \Sigma_t^{j,-1} \ \mathbf{R^T} \ \mathbf{x_t^j}} \\
= \ & \exp{-\tfrac{1}{2}\mathbf{x_t^{j,T}} \ \mathbf{R^T} \ \Omega_t^j \ \mathbf{R} \ \mathbf{x_t^j} - (\xi_t^j + \Omega_t^j \ \mathbf{D^T}) \ \mathbf{R^T} \ \mathbf{x_t^j}} \qquad (12.53)
\end{aligned}
$$

The key observations here are that the alignment takes time linear in the state vector (and not cubic as would be the case for EKFs). More importantly, the sparseness is preserved by this update step. The reader may also notice that the transformation can be applied to subsets of features (e.g., a local map), thanks to the sparseness of Ω_t^j. In such a case, one has to include the Markov blanket of the variables of interest.

After the alignment step, both maps are expressed in the same coordinate system. The joint information state is obtained by concatenating both information matrices and both information states. The correspondence list is then incorporated into this joint map by *collapsing* the corresponding rows and columns of the resulting information matrix and vector. The following example illustrates the operation of collapsing feature 2 and 4 in the filter, which would occur when our correspondence list states that landmark 2 and 4 are identical:

$$
\begin{pmatrix}
h_{11} & h_{12} & h_{13} & h_{14} \\
h_{21} & h_{22} & h_{23} & h_{24} \\
h_{31} & h_{32} & h_{33} & h_{34} \\
h_{41} & h_{42} & h_{43} & h_{44}
\end{pmatrix}
\longrightarrow
\begin{pmatrix}
h_{11} & h_{12}+h_{14} & h_{13} \\
h_{21}+h_{41} & h_{22}+h_{42}+h_{24}+h_{44} & h_{23}+h_{43} \\
h_{31} & h_{32}+h_{34} & h_{33}
\end{pmatrix}
\qquad (12.54)
$$

$$
\begin{pmatrix}
\xi_1 \\
\xi_2 \\
\xi_3 \\
\xi_4
\end{pmatrix}
\longrightarrow
\begin{pmatrix}
\xi_1 \\
\xi_2+\xi_4 \\
\xi_3
\end{pmatrix}
\qquad (12.55)
$$

Collapsing the information state exploits the additivity of the information state. The viability of a data fusion hypothesis is finally assessed by computing the *likelihood* of the data after fusing two maps. This is achieved by plugging the fused map into the original two Gaussians defining each vehicles' local maps, and by multiply the resulting probabilities. This calculation plays an important role in our algorithm's decision to accept or reject a fusion hypothesis. Technically, this operation involves

a recovery of the state, and the evaluation of two Gaussians (one per robot). The mathematics for doing so are straightforward and omitted for brevity.

12.10.2   Establishing Correspondence

The data association between multiple robots is mathematically identical to the problem with a single robot relative to its map, with the important difference than in multi-robot SLAM, the correspondence problem is *global*, that is, any pairs of features in two robots' maps can correspond. This makes global correspondence computatinally expensive.

The previous section provided a method for evaluating the goodness of a map fusion candidate, but left open how such candidates are found. Finding good candidates for fusing maps is essentially a hybrid search problem, involving continuous (alignment) and discrete (correspondence) variables.

Our approach performs this search in two states. First, it searches for corresponding pairs of local landmark configurations in different maps. In particular, our approach identifies for each landmark in each map all triplets of three adjacent landmarks that fall within a small radius (a similar grouping was used in [39]). The relative distances and angles in these triplets are then memorized in an SR-tree, to facilitate efficient retrieval. Using these SR-trees, similar local configurations are easily identified in different maps by searching the tree. Correspondences found in this search serve as a starting hypotheses for the fusion process; they are also used to recover the alignment parameters, the rotation r and the shift d (using the obvious geometric laws).

When an appropriate candidate has been identified, the space of possible data associations is searched recursively, by assuming and un-assuming correspondences between landmarks in the different maps. The search is guided by two opposing principles: The reduction of the overall map likelihood that comes from equating two landmarks, and the increase in likelihood that results from the fact that if there were really two separate landmarks, both robots should have detected both of them (and not just one). To calculate the latter, we employ a sensor model that characterizes "negative" information (not seeing a landmark).

In general, the search for the optimal correspondence is NP-hard. However, in all our experiments with real-world data we found hill climbing to be successful in every single instance. Hill climbing is extremely efficient; we suspect it to be in $O(N \log N)$ for maps of size N. In essence, it associates nearby landmarks if, as a result, the overall likelihood increases. Once the search has terminated, a fusion is finally accepted

if the resulting reduction of the overall likelihood (in logarithmic form) is offset by the number of collapsed landmarks times a constant; this effectively implements a Bayesian MAP estimator with an exponential prior over the number of landmarks in the world.

———-

In our implementation, the robots are informed of their initial pose. This is a common assumption in multi-robot SLAM, necessary for the type linearization that is applied both in EKFs and SEIFs [30]. Recent work that enables vehicles to build joint maps without initial knowledge of their relative pose can be found in [12, 37, 41].

Our simulation involves a team of three air vehicles. The vehicles are not equipped with GPS; hence they accrue positioning error over time. Figure 12.16 shows the joint map at different stages of the simulation. As in [30], we assume that the vehicles communicate updates of their information matrices and vectors, enabling them to generate a single, joint map. As argued there, the information form provides the important advantage over EKFs that communication can be delayed arbitrarily, which overcomes a need for tight synchronization inherent to the EKF. This characteristic arises directly from the fact that the information matrix Ω_t and the information vector ξ_t in SEIFs is additive, whereas covariance matrices are not. In particular, let $\langle \Omega_t^i, \xi_t^i \rangle$ be the posterior of the i-th vehicle. Assuming that all posteriors are expressed over the same coordinate system and that each map uses the same numbering for all landmarks, the joint posterior integrating all of these local maps is given by $\langle \sum_i \Omega_t^i, \sum_i \xi_t^i \rangle$. This additive nature of the information form is well-known, and has in the context of SLAM previously been exploited by Nettleton and colleagues [30]. SEIFs offer over the work in [30] that the messages sent between vehicles are small, sue to the sparse nature of the information form. A related approach for generating small messages in multi-vehicle SLAM has recently been described in [29].

Figure 12.16 shows a sequence of snapshots of the multi-vehicle system, using 3 different air vehicles. Initially, the vehicle start our in different areas, and the combined map (illustrated by the uncertainty ellipses) consists of three disjoint regions. During Steps 62 through 64, the top two vehicles discover identical landmarks; as a result, the overall uncertainty of their respective map region decreases; This illustrates that the SEIF indeed maintains the correlations in the individual landmark's uncertainties; albeit using a sparse information matrix instead of the covariance matrix. Similarly, in steps 85 through 89, the third vehicle begins to identical landmarks also seen by another vehicle. Again, the resulting uncertainty of the entire map is reduced, as can be seen easily. The last panel in Figure 12.16 shows the final map, obtained after 500 iterations. This example shows that SEIFs are well-suited for multi-robot SLAM, assuming that the initial poses of the vehicles are known.

12.11   DISCUSSION

This paper proposed an efficient algorithm for the SLAM problem. Our approach is based on the well-known information form of the extended Kalman filter. Based on the empirical observation that the information matrix is dominated by a small number of entries that are found only between nearby features in the map, we have developed a *sparse* extended information filter, or SEIF. This filter enforces a sparse information matrix, which can be updated in constant time. In the *linear* SLAM case with known data association, all updates can be performed in constant time; in the *nonlinear case*, additional state estimates are needed that are not part of the regular information form of the EKF. We proposed a amortized constant-time coordinate descent algorithm for recovering these state estimates from the information form. We also proposed an efficient algorithm for data association in SEIFs that requires logarithmic time, assuming that the search for nearby features is implemented by an efficient search tree. The approach has been implemented and compared to the EKF solution. Overall, we find that SEIFs produce results that differ only marginally from that of the EKFs, yet at a much improved computational speed. Given the computational advantages of SEIFs over EKFs, we believe that SEIFs should be a viable alternative to EKF solutions when building high-dimensional maps.

SEIFs, are represented here, possess a number of critical limitations that warrant future research. First and foremost, SEIFs may easily become overconfident, a property often referred to as *inconsistent* [20, 15]. The overconfidence mainly arises from the approximation in the sparsification step. Such overconfidence is not necessarily an problem for the convergence of the approach [23], but it may introduce errors in the data association process. In practice, we did not find the overconfidence to affect the result in any noticeable way; however, it is relatively easy to construct situations in which it leads to arbitrary errors in the data association process.

Another open question concerns the speed at which the amortized map recovery converges. Clearly, the map is needed for a number of steps; errors in the map may therefore affect the overall estimation result. Again, our real-world experiments show no sign of noticeable degradation, but a small error increase was noted in one of our simulated experiments.

Finally, SEIF inherits a number of limitations from the common literature on SLAM. Among those are the use of Taylor expansion for linearization, which can cause the map to diverge; the static world assumption which makes the approach inapplicable to modeling moving objects [43]; the inability to maintain multiple data association hypotheses, which makes the approach brittle in the presence of ambiguous features; the reliance on features, or landmarks; and the requirement that the initial pose be

known in the multi-robot implementation. Virtually all of these limitations have been addressed in the recent literature. For example, a recent line of research has devised efficient particle filtering techniques [13, 23, 26] that address most of these shortcomings. The issues addressed in this paper are somewhat orthogonal to these limitations, and it appears feasible to combine efficient particle filter sampling with SEIFs. We also note that in a recent implementation, a new lazy data association methodology was developed that uses a SEIF-style information matrix to robustly generate maps with hundreds of meters in diameter [40].

The use of sparse matrices in SLAM offers a number of important insights into the design of SLAM algorithms. Our approach puts a new perspective on the rich literature on hierarchical mapping discussed further above. As in SEIFs, these techniques focus updates on a subset of all features, to gain computational efficiency. SEIFs, however, composes submaps dynamically, whereas past work relied on the definition of static submaps. We conjecture that our sparse network structures capture the natural dependencies in SLAM problems much better than static submap decompositions, and in turn lead to more accurate results. They also avoid problems that frequently occur at the boundary of submaps, where the estimation can become unstable. However, the verification of these claims will be subject to future research. A related paper discusses the application of constant time techniques to information exchange problems in multi-robot SLAM [28].

Finally, we note that our work sheds some fresh light on the ongoing discussion on the relation of topological and metric maps, a topic that has been widely investigated in the cognitive mapping community [6, 17]. Links in SEIFs capture relative information, in that they relate the location of one landmark to another (see also [7, 9, 31]). This is a common characteristic of topological map representations [5, 35, 18, 22]. SEIFs also offer a sound method for recovering absolute locations and affiliated posteriors for arbitrary submaps based on these links, of the type commonly found in metric map representations [25, 36]. Thus, SEIFs bring together aspects of both paradigms, by defining simple computational operations for changing relative to absolute representations, and vice versa.

[...]

However, a note of caution is in order. If SEIFs were applied to a linear system (meaning, we don't need Taylor series approximations), the update would be truly constant time. However, because of the need to linearize, we need an estimate of the mean μ_t along with the information state. This estimate is not maintained in the traditional information filter, and recovering it requires a certain amount of time. Our SEIF implementation only approximates it, and the quality of the posterior estimate depends on the quality of this approximation. We will return to this issue at the very end of

this chapter, where we discuss some of the shortcomings and extensions of the SEIF paradigm. For now, we will begin with a derivation of the four essential update steps in SEIFs.

[...]

However, to attain efficient online, updating the SEIF has to make a number of approximations, which make its result less accurate than that of the EIF. In particular, the SEIF has two limitations: First, it linearizes only once, just like the EKF. The EIF can re-linearize, which generally improves the accuracy of the result. Second, the SEIF uses an approximation step to main sparsity of its information matrix. This sparsity was naturally given for the full-SLAM EIF algorithm, by nature of the information that was being integrated. SEIFs integrate out past poses, and this very step violates the sparseness of the information representation. The SEIF uses an approximation step to retain sparseness, which is essential for efficient updating of the information matrix in the context of online SLAM.

13

MAPPING WITH UNKNOWN DATA ASSOCIATION

13.1 LATEST DERIVATION

This is the correct derivation of EM, unfortunately using a slightly different notation than in the rest of the book. It sits here to remind Sebastian that it has to be incorporated into the text and parts of it have to ultimately disappear into the appendix. So, reader, don't read this. If you do this anyhow, we use s_t for the pose at time t and s^t for the poses leading up to time t. z is a measurement, and u is a control. Also, the control for the interval $(t-1, t]$ is denoted u_t, and not u_{t-1}. Good luck!

Here it goes...

$$
\begin{aligned}
p(d^t, s^t | m) &= p(d^t | s^t, m)\, p(s^t | m) \\
&= \eta \prod_\tau p(u_\tau | s_\tau, s_{\tau-1}) \prod_\tau p(z_\tau | s_\tau, m)
\end{aligned}
\tag{13.1}
$$

Taking the logarithm on both sides gives us

$$
\log p(d^t, s^t | m) = \eta' + \sum_\tau \log p(u_\tau | s_\tau, s_{\tau-1}) + \sum_\tau \log p(z_\tau | s_\tau, m)
\tag{13.2}
$$

Here η and η' are constants. Introduce binary fields of indicator variables I_{s_τ} and $I_{s_\tau, s_{\tau-1}}$. $I_{s_\tau} = 1$ if and only if the robot's pose at time t was s_τ, and $I_{s_\tau, s_{\tau-1}} = 1$ if and only if the robot's pose at time t was s_τ *and* the pose at time $t-1$ was $s_{\tau-1}$. The set of all of those variable will be called I. They are the latent variables. This gives us

the form:

$$\log p(d^t, I|m) \tag{13.3}$$
$$= \eta' + \sum_\tau \int \int I_{s_\tau, s_{\tau-1}} \log p(u_\tau|s_\tau, s_{\tau-1}) \, ds_\tau \, ds_{\tau-1} + \sum_\tau \int I_{s_\tau} \log p(z_\tau|s_\tau, m) \, ds_\tau$$

Now calculate the expectations of those indicator variables I:

$$E_I[\log p(d^t, I|m)] \tag{13.4}$$
$$= E_I \left[\eta' + \sum_\tau \int \int I_{s_\tau, s_{\tau-1}} \log p(u_\tau|s_\tau, s_{\tau-1}) \, ds_\tau \, ds_{\tau-1} + \sum_\tau \int I_{s_\tau} \log p(z_\tau|s_\tau, m) \, ds_\tau \right]$$
$$= \eta' + \sum_\tau \int \int E[I_{s_\tau, s_{\tau-1}}] \log p(u_\tau|s_\tau, s_{\tau-1}) \, ds_\tau \, ds_{\tau-1} + \sum_\tau \int E[I_{s_\tau}] \log p(z_\tau|s_\tau, m) \, ds_\tau$$

where the expectation of the indicator variables I is conditioned on the current map, $m^{[i]}$. In other words, we have (I am not sure it's good to even write it in this form):

$$E_{s^t}[\log p(d^t, s^t|m)] \tag{13.5}$$
$$= \eta' + \sum_\tau \int \int p(s_\tau, s_{\tau-1}|m^{[i]}, z^t, u^t) \log p(u_\tau|s_\tau, s_{\tau-1}) \, ds_\tau \, ds_{\tau-1}$$
$$+ \sum_\tau \int p(s_\tau|m^{[i]}, z^t, u^t) \log p(z_\tau|s_\tau, m) \, ds_\tau$$

Thus, in the E-step we have to calculate the expectations

$$I_{s_\tau} = p(s_\tau|m^{[i]}, z^t, u^t)$$
$$I_{s_\tau, s_{\tau-1}} = p(s_\tau, s_{\tau-1}|m^{[i]}, z^t, u^t) \tag{13.6}$$

for all t, s_τ, and $s_{\tau-1}$. In the M-step, we seek to maximize (13.5) under these epectations, which will give us the $(i+1)$-st map. Fortunately, not all the terms in (13.5) depend on the map m:

$$m^{[i+1]} = \underset{m}{\operatorname{argmax}} \sum_\tau \int I_{s_\tau} \log p(z_\tau|s_\tau, m) \, ds_\tau \tag{13.7}$$

In retrospect, we notice that we don't even have to calculate the indicator variables $I_{s_\tau, s_{\tau-1}}$. Relative to the map m, the term is a constant. So we can omit them early on.

This gives us for the E-step, assuming uniform prior over robot poses (here $z^{t\backslash\tau} = \{z_{\tau+1}, z_{\tau+2}, \ldots, z_t\}$):

$$
\begin{aligned}
I_{s_\tau} &= p(s_\tau | m^{[i]}, z^t, u^t) \\
&= \eta \, p(z^{t\backslash\tau}, u^{t\backslash\tau} | m^{[i]}, s_\tau, z^\tau, u^\tau) \, p(s_\tau | m^{[i]}, z^\tau, u^\tau) \\
&= \eta \, p(z^{t\backslash\tau}, u^{t\backslash\tau} | m^{[i]}, s_\tau) \, p(s_\tau | m^{[i]}, z^\tau, u^\tau) \\
&= \eta' \, p(s_\tau | m^{[i]}, z^{t\backslash\tau}, u^{t\backslash\tau}) \, p(z^{t\backslash\tau}, u^{t\backslash\tau} | m^{[i]}) \, p(s_\tau | m^{[i]}, z^\tau, u^\tau) \\
&= \eta'' \, p(s_\tau | m^{[i]}, z^{t\backslash\tau}, u^{t\backslash\tau}) \, p(s_\tau | m^{[i]}, z^\tau, u^\tau) \qquad\qquad (13.8)
\end{aligned}
$$

which is a forwards-backwards localization.

For M-step, we have:

$$
m^{[i+1]} = \operatorname*{argmax}_{m} \sum_\tau \int I_{s_\tau} \log p(z_\tau | s_\tau, m) \, ds_\tau \qquad\qquad (13.9)
$$

Conveniently decouple this for each location $\langle x, y \rangle$. Notice, this is approximate, as those cells aren't independent. An alternative would be to run Simulated Annealing. But decoupling common in the literature.

$$
m_{x,y}^{[i+1]} = \operatorname*{argmax}_{m_{x,y}} \sum_\tau \int I_{s_\tau} \log p(z_\tau | s_\tau, m_{x,y}) \, ds_\tau \qquad\qquad (13.10)
$$

We observe that

$$
\begin{aligned}
\log p(z_\tau | s_\tau, m_{x,y}) &= \log[\eta \, p(m_{x,y} | s_\tau, z_\tau) \, p(z_\tau | s_\tau)] \\
&= \eta' + \log p(m_{x,y} | s_\tau, z_\tau) + \log p(z_\tau | s_\tau) \qquad\qquad (13.11)
\end{aligned}
$$

and hence

$$
m_{x,y}^{[i+1]} = \operatorname*{argmax}_{m_{x,y}} \sum_\tau \int I_{s_\tau} [\log p(m_{x,y} | s_\tau, z_\tau) + \log p(z_\tau | s_\tau)] \, ds_\tau
$$

$$= \underset{m_{x,y}}{\operatorname{argmax}} \sum_{\tau} \int I_{s_\tau} \log p(m_{x,y}|s_\tau, z_\tau) \, ds_\tau \qquad (13.12)$$

Great, eh? The only thing I don't like is the M-step. If we can fix that, we might actually prove convergence!

All right, I let you read the motivation now. Don't pay attention to any of the derivations.

13.2 MOTIVATION

In the previous chapter, we discussed a mapping algorithm that calculates the full, joint posterior over poses and maps. From a probabilistic standpoint of view, the estimation of the posterior is clearly the gold standard, as it carries information about the map uncertainty, the pose uncertainty, and even the cross-dependencies between pose and map estimates.

However, to obtain a posterior estimation algorithm, the approach described in the previous chapter had to resort to a number of restrictive assumptions. The most important of those assumptions is the absence of a data association problem. Put differently, it is critical for these algorithms that they can can establish correspondence between sights of the same environment feature at different points in time—otherwise the posterior would not be multi-modal!. For this reason, mapping algorithms using Kalman filters often extract from the sensor data a small number of landmarks (features) which can be disambiguated easily. By doing so, they are forced to throw away most of the data. For example, in the underwater mapping results mentioned in the previous chapter, most of the measurements of the underwater sea bed were discarded, and only a small number of feature detections were used for building a map. How much better a map could one build by considering all available data! These observation motivates our desire to develop mapping algorithms that can cope with ambiguous sensor data, and with the data association problem. Such algorithms are subject of this and the following chapter.

This chapter presents a family of mapping algorithms that specifically address the data association problem. In particular, they can cope with ambiguous landmarks and estimate the correspondence between them on-the-fly. Variants of the basic algorithm can also cope with raw sensor data, by compiling sequences of data into small local maps, and merging them in a statistical sound way into a global, consistent map. On the downside, the algorithms developed here have two disadvantages when compared

to the Kalman filter approach. First, instead of computing the full posterior over maps and poses, they only strive for the most likely map. Consequently, the result of this algorithm does not explicitly represent the uncertainty in the map or the robot poses. Second, the algorithms discussed here are batch algorithms. They require multiple passes through the entire data set. Kalman filters, in contrast, are incremental. Once a measurement has been processed, it does not have to be memorized.

Nevertheless, the algorithms discussed here are important from a practical point of view. The data association problem is generally recognized as the most challenging problem in concurrent mapping and localization, and these algorithms are currently the only known approach that addresses this problem in a statistically sound way. The algorithms have also generated some of the largest maps ever built, in environments that are intrinsically difficult to map.

This chapter contains some statistical material which is slightly more advanced than the material in the previous chapters. In particular, it uses (without proof) Dempster's EM algorithm [], which is a well-known statistical technique for maximum likelihood estimation with missing data. To communicate the basic ideas first before diving into mathematical formalism, this chapter begins with an informal description of the basic ideas underlying all of the algorithms discussed in this chapter. In particular,

- Section 13.3 introduces the basic ideas using two simple, artificial examples.

- Section 13.4 states the basic EM algorithm for concurrent mapping and localization, and derives it mathematically.

- Section 13.5 discusses how implement EM. In particular, it discusses the implementation of a landmark-based mapping algorithm, which uses piecewise constant approximation of probability density functions, and points out strategies of reducing the computational complexity.

- Section 13.6 introduces a layered version of this algorithm, which relies on local maps as the basic entities for EM. As is discussed there, this layered version makes it possible to apply EM to occupancy grids-style metric maps.
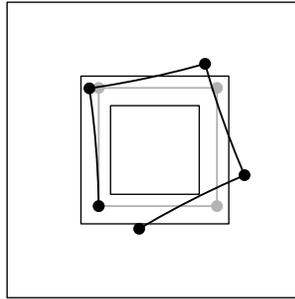
Figure 13.1 Environment (thin line); the robot's path (grey); and the measured path (thick line). The robot can perceive whether or not it is in one of the indistinguishable corners, within a certain maximum perceptual range. The dots indicate locations where the robot perceived a corner.

13.3   MAPPING WITH EM: THE BASIC IDEA

Let us begin by discussing the basic ideas and concepts in mapping with EM. As we will see, the general idea are quite intuitive and, like everything else in this book, critically linked to the notion of probabilities.

The goal of EM mapping is to find the most likely map given the data. Put mathematically, we are now interested in calculating

$$\operatorname*{argmax}_{m} P(m \mid d) \tag{13.13}$$

for arbitrary data sets and maps. Unfortunately, searching the space of maps exhaustively is impossible, since it is too large. EM therefore performs hill climbing in the space of all maps. More specifically, it starts out with a very poor map, then develops a better map, and so on, until it finally arrives at a local maximum in likelihood space.

In particular, and this is the basic "trick" of EM, EM iterates two different estimation steps

- a localization step, also called E-step for reasons given below, in which the robot localizes itself in its current best map, and

- a mapping stape, also called M-step, where a new, improved map is calculated based on the result of localization.
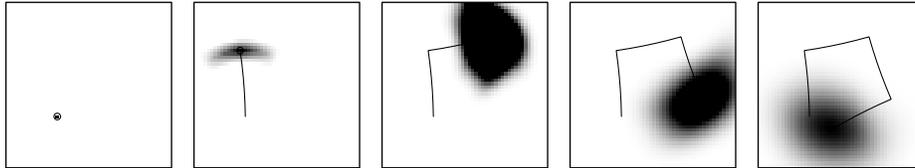
Figure 13.2 The initial alpha-values, using only odometry to compute posterior distributions over the robot's poses.

Clearly, this problem decomposition has the advantage that the steps individually are much easier than the concurrent In Chapter 7, we have already seen how to localize robots in a known map (the E-step). In Chapter 9, we have discussed techniques for building maps from known poses (M-step). The EM approach alternates localization and mapping, using techniques that bear similarity to the ones described before.

To illustrate EM, consider the rectangular environment shown in Figure 13.1. Let us for the moment assume that the robot has a sensor for detecting corners. Whenever it is near or in a corner, it can perceive its relative location, but it is unable to tell the corners apart. Notice that this environment is highly symmetric. The mapping problem clearly raises the data association problem of establishing correspondence between different detections of the same corner. Figure 13.1 also shows the robot's path (in gray). Dots along the path indicate when the robot's sensors detected a corner. Obviously, our robot tends to drift to right, relative to its odometry measurements.

The EM algorithm starts with a localization step without any map. Figure 13.2 shows this for the different points in time. Initially, pose defined to be $\langle x = 0, y = 0, \theta = 0 \rangle$, which is shown in the first (and leftmost) diagram in Figure 13.2. After the first motion segment to the adjacent corner, robot's belief shown in second diagram. This belief is purely based on odometry; since there is no map, no opportunity to constrain the belief in any way. The next three diagrams show robot's belief after the next three motion segment, illustrating that the robot's uncertainty increases over time.

The pose estimates are now used to generate a first, highly uncertain map. This map is shown in Figure 13.3. Here the grey-level corresponds to the likelihood that a x-y-location in the map is a corner: Black means the robot is certain there is a corner, white means the robot is certain there is none, and values in between indicate probabilities for cornerness between 0 (white) and 1 (black). The map in Figure 13.3 possesses several remarkable properties. First, it is highly uncertain, as indicated by the large regions that might possible be corners. The grey-ish circle at the bottom left corresponds to the first perception, where the robot knew its pose (by definition it was $\langle 0, 0, 0 \rangle$) and only saw a single corner within its circular perceptual field. The other
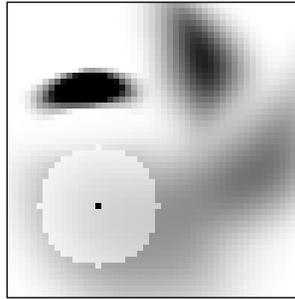
Figure 13.3   The initial map.

grey regions correspond to points in time where the robot perceived the same sensor reading, but was less certain about its position. Hence, the resulting map is less sharp.

The robot now re-runs the localization, but this time using the initial map. The map, while far from accurate, nevertheless leads to improved localization. The new localization run based on the initial map is shown in Figure 13.4. Comparison with the first localization results (Figure 13.2) shows that the margin of uncertainty has shrunk, illustrates the positive effect of the map on localization.

So far, only talked about *forward localization*, using the data leading up to time t to determine the robot's pose at time t. Strictly speaking, even data collected after time t carries information about the robot's pose at time t. Thus, some sort of *backwards localization* is called for that uses future data to revise past beliefs. Figure 13.5 shows the result of backwards localization. This figure is to be read from right to left. Since the robot's final pose is unknown, we start with a uniform distribution as indicated in the rightmost diagram in Figure 13.5. One time step earlier, the robot's distribution is given by the diagram next to the left. This is identical to global localization, just in reverse time order. The fact that this diagram contains a collection of circles indicates that after a single measurement followed by a motion command, the robot is probably on a fixed distance to one of the (approximately) known corner locations. From right to left, the diagrams in Figure 13.5 show the result of backwards localization all the way to the initial time step.

Strictly speaking, our approach requires backwards in all iterations, even the first, but in the absence of a map the resulting distributions are uniform, which explains why we did not address this issue in the first round of localization.
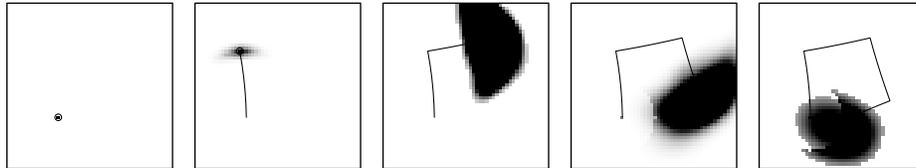
Figure 13.4 These alpha-values are based on odometry (as before) and the initial, highly uncertain map.
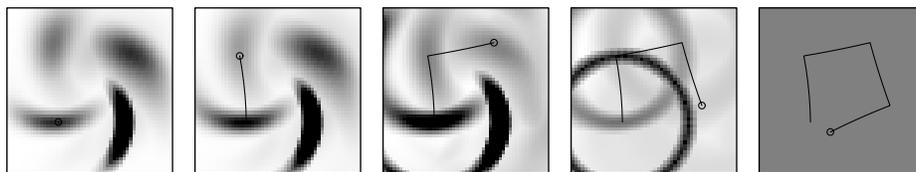
Figure 13.5 Backwards localization: The beta-values are computed form right to left, effectively localizing the robot backwards in time.
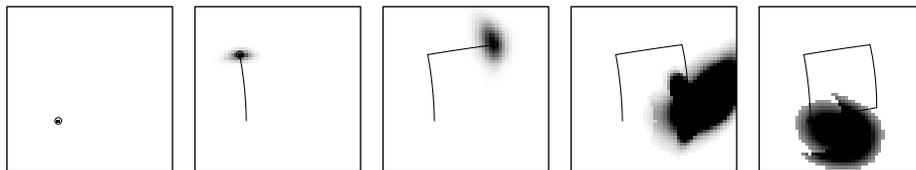
Figure 13.6 The combined results of forward and backward localization.
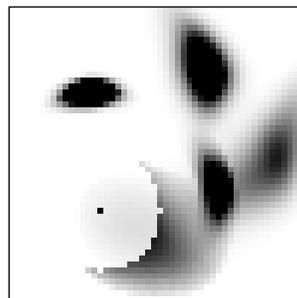
Figure 13.7 The second map.
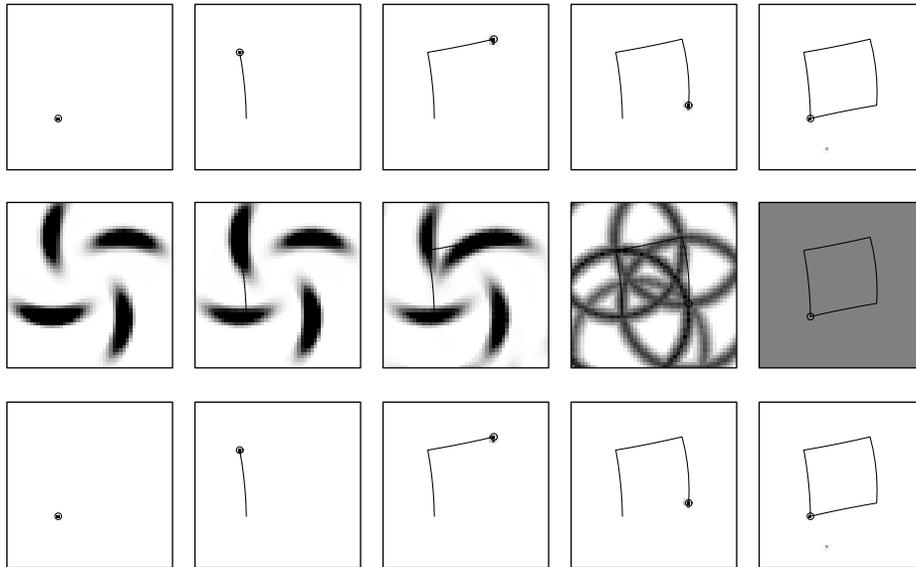
Figure 13.8   The combined results of forward and backward localization.
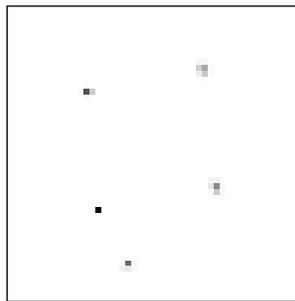
Figure 13.9   The final map.

The true posterior over the robot location at the various points in time is shown in Figure 13.6. Each diagram is the (normalized) *product* of the forward localization and the backwards localization results (c.f., Figure 13.4 and 13.5, respectively).

The reader should notice that the pose estimates, while far from being certain, are an improvement over the initial pose estimates when no map is used. The new, improved pose estimates are now used to generate a new, improved map, shown in Figure 13.7.
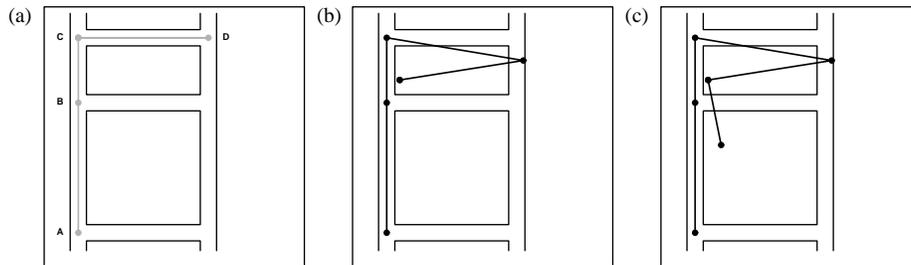
Figure 13.10   Test environment and data: (a) The environment possesses 4 indistinguish-able landmark locations, labeled A to D. The robot's path is: A-B-C-D-C-B. (b) First data set, as measured by robot's odometry. (c) Second data set, which contains one more step than the first.
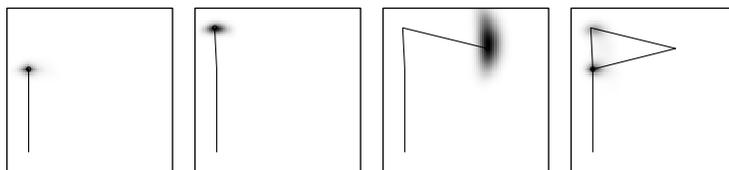
Figure 13.11   Data set 1, localization after three iterations of EM.
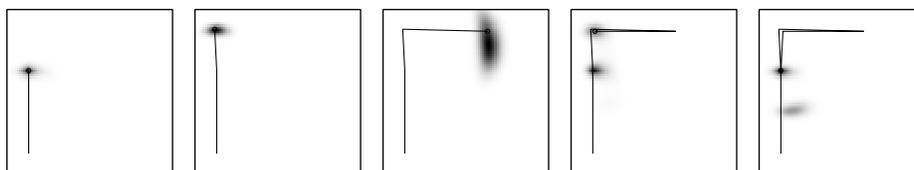
Figure 13.12   Data set 2, localization after three iterations of EM.

A comparison with the first map (Figure 13.3) shows that now the robot is more certain where the corners in its environment are.

The process of localization and mapping is iterated. After 10 iterations, the process nearly converged for our toy-like example. The resulting pose estimates are shown in the bottom row in Figure 13.8; the top and middle row show the result of forward and backward localization, respectively. At this point, the uncertain vanished almost entirely, even for the intermediate locations. At the last point in time, the robot is now certain that the current corner is identical with the first, as illustrated by the diagram in the bottom-right corner. Remarkable are the results of backward localization, which are spirals in x-y-θ space (and hence circles in the projective x-y space).

The corresponding map, shown in Figure 13.9, shows the location of four corners that is slightly rotated but otherwise consistent with the true arrangement. The final map also contains a spurious fifth corner, which is the result of "over-fitting" at the very border of the robot's perceptual range.

A second example is shown in Figures 13.10 to 13.12. This example illustrates how the algorithm effectively uses data to revise beliefs backwards in time. In this example, our algorithm first fails to localize the robot correctly, due to lack of data. As more data arises, the false beliefs are identified and corrected in favor of the correct ones.

Figure 13.10a shows the environment: A multi-corridor environment that possesses four "significant" places (labeled A to D). For the sake of the illustration, we assume that these places are all indistinguishable. The robot's path is A-B-C-D-C-B, that is, the robot moves up, turns right, then turns around and move half way back to where it started. Of course, the odometry is erroneous. Figures 13.10b&c depict the path, as obtained through the robot's odometry. In Figure 13.10b, the situation is depicted after the robot revisited location C. One step later, the robot is back at location B, as shown in Figure 13.10c.

The interesting feature of this example is that after returning back to location C (Figure 13.10b), the robot's pose is much close to location B. Since our robot actually cannot receive any corridors/walls, this suggests that maximum likelihood solution places the robot is at B, and not at C; this is a case where a false solution is actually more likely than the correct one. Figure 13.11 confirms our suspicion. It shows the robot's pose beliefs after three iterations of EM. The interesting diagram is the one on the right: Here the belief possesses two modes, one at the correct location and one at location B. Since the odometry is closer to B than to C, the likelihood of being at B is slightly larger. Thus, if one wanted to know the most likely path, EM would chose the one shown in this diagram.

The picture changes as the last data item is added, in which the robot moved from C to B, as shown in Figure 13.10c. This last step is easily explained if the robot is at C; starting at B, however, the new observation is less consistent. Applying EM to this new data set yields the beliefs shown in Figure 13.12. Here the most likely path is actually the correct one, even though the likelihood is only changed by a small amount. This example illustrates that EM is capable of using data to revise beliefs backwards in time. This is important specifically when mapping *cyclic* environments. Then first closing a cycle, the sensor data often is insufficient to determine the robot's pose correctly, requiring some sort of backwards-correction mechanisms that can fix errors in maps as new evidence arrives.

This completes the examples of EM mapping. We will now formally describe the EM mapping algorithm and derive it mathematically. Iterates localization and mapping. The forward pose estimates will be called α, the backward estimates β, and the product of both γ. Algorithm: maximizes the map likelihood. Pose estimated may not be interpreted as posterior, since evidence is used multiple times and hence robot is "over-confident." Our illustration also has implications for the choice of the representation. Even in this simple example,the intermediate pose estimates cover a wide range of distributions.

13.4   MAPPING WITH THE EM ALGORITHM

13.4.1   The EM Mapping Algorithm

Armed with the necessary mathematical derivation, we will now state the general EM mapping algorithm. Table 13.1 depicts the algorithm **EM_mapping()**. Here η denotes a generic normalizer that ensures that the desired probability distributions integrate (sum up) to 1.

The algorithm **EM_mapping()** is a batch algorithm. Its input is a data set d, from which it generates a map m. The algorithm **EM_mapping** is initialized in line 2 by assigning uniform distributions to each variable in the map. The main loop, which is iterated until convergence, implements the E-step (lines 4 to 11) and the M-step (lines 12 to 16). The E-step consists of three parts: a forward localization (lines 4 to 6), an inverse localization (lines 7 to 9), and a multiplication to obtain the full posterior beliefs over all poses (lines 10 and 11). The M-step computes, for each map element $\langle x, y \rangle$, the non-normalizes likelihood that the world's property at $\langle x, y \rangle$ is l (lines 13 and 14), and subsequently normalizes these probabilities (line 15). Finally, it compiles a single, global map (line 16). After convergence, the final map is returned (line 17). Empirically, 3 to 5 iterations often result in a map that is sufficiently close to the asymptotic result, so that the computation can be terminated. Notice that when stating the algorithm, **EM_mapping()** we omitted the index j. This is legitimate because the algorithm is incremental, ant it suffices to memorize the most recent map $m_{[j]}$ and beliefs $Bel_{[j]}(s^{(t)})$.

Make Figure: The L-shaped example environment, and maploc distributions

1:           **Algorithm EM_mapping(d):**

2:                *initialize m by* uniform-map

3:                *repeat until satisfied*
4:                     $\alpha^{(0)} = \mathrm{Dirac}(\langle 0,0,0 \rangle)$
5:                     *for $t = 1$ to T do*
6:                          $\alpha^{(t)} = \eta\, P(o^{(t)} \mid s^{(t)}, m) \int P(s^{(t)} \mid a^{(t-1)}, s^{(t-1)})\, \alpha^{(t-1)}\, ds^{(t)}$
7:                     $\beta^{(T)} = \mathrm{uniform}()$
8:                     *for $t = T - 1$ down to 0 do*
9:                          $\beta^{(t)} = \eta \int P(s^{(t)} \mid a^{(t)}, s^{(t+1)})\, P(o^{(t+1)} \mid s^{(t+1)}, m)\, \beta^{(t+1)}\, ds^{(t+1)}$
10:                    *for $t = 1$ to T do*
11:                         $Bel(s^{(t)}) = \alpha^{(t)}\, \beta^{(t)}$
12:                    *for all $\langle x, y \rangle$ do*
13:                         *for all $l \in L$ do*
14:                              $\bar{m}_{\langle x,y \rangle = l} = \sum_{t=0}^{T} \int P(o^{(t)} \mid s^{(t)}, m_{\langle x,y \rangle} = l)\cdot$
                                                                  $\cdot I_{\langle x,y \rangle \in \mathrm{range}(o^{(t)}, s^{(t)})}\, Bel(s^{(t)})\, ds^{(t)}$
15:                         $m_{\langle x,y \rangle = l} = \left(\sum_{l' \in L} \bar{m}_{\langle x,y \rangle = l'} \right)^{-1} \bar{m}_{\langle x,y \rangle = l}$
16:                    $m = \{ m_{\langle x,y \rangle} \}$

17:               *return m*

Table 13.1   The general EM mapping algorithm.

Figure **??** shows an illustrative example of the algorithm **EM_mapping()** in action. Here an imaginary robot, equipped with a sensor that produces noisy measurements of range and bearing to nearby landmarks, moves through an L-shaped environment. The environment possesses four landmarks. To make things interesting, let us assume the landmarks are indistinguishable to the robot, and the robot does not know how many there are. Figure **??** shows the robot's true trajectory, along with the measured one. Obviously, there is odometry error. When returning the landmark labeled 4, odometry suggests that the next landmark is closes to the one labeled 2; in reality however, the robot encountered landmark 3, and then moved back to landmark 2.

[[[Insert picture, complete the story]]]

Illustrates the importance of uncertainty. Also demonstrates that the robot can revise past beliefs as more evidence arrives. This is essential for "closing the loop."

13.4.2 The Map Likelihood Function

We begin our formal derivation with a statement of the likelihood function, whose maximization is our goal. As stated in the introduction of this chapter, our approach maximizes

$$P(m \mid d), \tag{13.14}$$

the likelihood of the map given the data. Our first exercise will be to express $P(m \mid d)$ in terms of familiar conditional probabilities, in particular the perceptual model $P(o \mid s, m)$ and the motion model $P(s' \mid a, s)$. In particular, we will find that

$$
\begin{aligned}
&P(m \mid d) \\
&= \ \eta \, P(m) \int \cdots \int \prod_{t=1}^{T} P(o^{(t)} \mid m, s^{(t)}) \prod_{t=1}^{T-1} P(s^{(t+1)} \mid a^{(t)}, s^{(t)}) \, ds^{(1)}, \ldots ds^{(T)}
\end{aligned} \tag{13.15}
$$

where η is a numerical constant: a normalizer that ensures that the left-hand side integrates to 1 over all m. Since we are only interested in maximization, not in computing the actual likelihood of a map, η is irrelevant.

The term $P(m)$ in (13.15) is the *prior* on maps. It allows one to express an a priori preference for certain maps over others. For example, if the environment is known to consist of parallel and orthogonal walls only—a commonly made assumption that can greatly simplify the mapping problem—one might exclude maps that do not obey that assumption. Throughout this text, however, we will consider the most general case and assume that $P(m)$ is uniformly distributed. Hence, $P(m)$ is a constant and can be ignored in the maximization, just like η. With these considerations, (13.15) shows that, at least in principle, finding the most likelihood can be done solely from the perceptual model $P(o \mid s, m)$, the motion model $P(s' \mid a, s)$, and the data d.

To see that (13.15) is correct, let us begin by applying Bayes rule to $P(m \mid d)$:

$$P(m \mid d) \ = \ \frac{P(d \mid m) \, P(m)}{P(d)} \tag{13.16}$$

Obviously, the denominator $P(d)$ is a constant factor that does not depend on m. To simplify the notation, we prefer to write

$$P(m \mid d) \quad = \quad \eta_1 \ P(m) \ P(d \mid m) \tag{13.17}$$

where η_1 is a normalizer that ensures consistency:

$$\eta_1 \quad = \quad \left(\sum_m P(d \mid m) \ P(m) \right)^{-1} \tag{13.18}$$

The rightmost term in Expression (13.17), $P(d \mid m)$, can be rewritten using the Theorem of Total Probability:

$$P(d \mid m) \quad = \quad \int \cdots \int P(d \mid m, s^{(0)}, \dots, s^{(T)})$$
$$P(s^{(0)}, \dots, s^{(T)} \mid m) \ ds^{(0)}, \dots, ds^{(T)} \tag{13.19}$$

Here we benefit from our careful definition of the motion model in Chapter **??**, which ensures that the rightmost term, $P(s^{(0)}, \dots, s^{(T)} \mid m)$, is a constant that does not depend on m. Hence, we obtain

$$P(d \mid m) \quad = \quad \eta_2 \int \cdots \int P(d \mid m, s^{(0)}, \dots, s^{(T)}) \ ds^{(0)}, \dots, ds^{(T)} \tag{13.20}$$

where η_2 is another normalizer. Further below, we will subsume η_1 and η_2 into a single constant, but for now let's keep them separate.

The key observation in our derivation is that given the map m and all T poses $s^{(0)}, \dots, s^{(T)}$, the Markov assumption renders all observations and motion commands conditionally independent. In other words, $P(d \mid m, s^{(0)}, \dots, s^{(T)})$ can be re-written as a product of "local" probabilities, conditioned on the appropriate state information:

$$P(d \mid m, s^{(0)}, \dots, s^{(T)}) \quad = \quad \prod_{t=0}^{T} P(o^{(t)} \mid s^{(t)}, m) \prod_{t=0}^{T-1} P(a^{(t)} \mid s^{(t+1)}, s^{(t)}) \tag{13.21}$$

The last term, $P(a^{(t)} \mid s^{(t+1)}, s^{(t)}, m)$, is simplified by first noticing that $a^{(t)}$ does not depend at all on the map m, hence

$$P(a^{(t)} \mid s^{(t+1)}, s^{(t)}, m) = P(a^{(t)} \mid s^{(t+1)}, s^{(t)}) \qquad (13.22)$$

and then further applying Bayes rule

$$P(a^{(t)} \mid s^{(t+1)}, s^{(t)}) = \frac{P(s^{(t+1)} \mid a^{(t)}, s^{(t)}) \, P(a^{(t)} \mid s^{(t)})}{P(s^{(t+1)} \mid s^{(t)})} \qquad (13.23)$$

Again, the denominator $P(s^{(t+1)} \mid s^{(t)})$ is a constant and so is the distribution $P(a^{(t)} \mid s^{(t)})$, since in the absence of any other information a single pose does not convey information about subsequent poses or motion commands. Hence, we can re-write (13.23) using the normalizer η_3:

$$P(a^{(t)} \mid s^{(t+1)}, s^{(t)}, m) = \eta_3 \, P(s^{(t+1)} \mid a^{(t)}, s^{(t)}) \qquad (13.24)$$

Substituting (13.24) into (13.21) yields

$$P(d \mid m, s^{(0)}, \ldots, s^{(T)}) = \eta_4 \prod_{t=0}^{T} P(o^{(t)} \mid s^{(t)}, m) \prod_{t=0}^{T-1} P(s^{(t+1)} \mid a^{(t)}, s^{(t)}) \qquad (13.25)$$

Notice that we have managed to express the desired probability in terms of familiar expressions: The perceptual model $P(o^{(t)} \mid s^{(t)}, m)$ and the motion model $P(s^{(t+1)} \mid a^{(t)}, s^{(t)})$.

We now substitute (13.25) into (13.20) and obtain

$$P(d \mid m) = \qquad (13.26)$$
$$\eta_2 \, \eta_4 \int \cdots \int \prod_{t=0}^{T} P(o^{(t)} \mid s^{(t)}, m) \prod_{t=0}^{T-1} P(s^{(t+1)} \mid a^{(t)}, s^{(t)}) \, ds^{(0)}, \ldots, ds^{(T)}$$

and back into (13.17):

$$P(m \mid d) = \qquad (13.27)$$

$$\eta\, P(m) \int \cdots \int \prod_{t=0}^{T} P(o^{(t)} \mid s^{(t)}, m) \prod_{t=0}^{T-1} P(s^{(t+1)} \mid a^{(t)}, s^{(t)})\, ds^{(0)}, \ldots, ds^{(T)}$$

Here we collapsed all normalizers into a single one ($\eta = \eta_1 \eta_2 \eta_4$), which yields the desired expression (13.15).

13.4.3 Efficient Maximum Likelihood Estimation

Having successfully derived the formula for map likelihood the next logical question is how to actually *maximize* it, thus, how to find the most likely map.

A simple-minded approach, which does *not* work, would be a generate-and-test approach:

> **Algorithm generate_and_test(d):**
>> *for all maps m do*
>>> *compute non-normalized likelihood $p \propto P(m \mid d)$*
>>> *if p larger than that of any previous map then*
>>>> $m^* = m$
>> *return m^**

Unfortunately, this algorithm fails for practical reasons. If each discrete map has 10^4 binary elements, which is much less than the number of grid cells for some of the maps shown further below, then there will be $2^{10,000}$ different maps. Evaluating a single one of them requires summation often over as 10^4 nested integrals or more ($T = 10^4$ data items may be easily collected in a few minutes), each of which can take another 10^6 values or so—this is clearly infeasible. The fact that the likelihood function is highly non-linear suggests that no practical algorithm exists that may ever be able to find it. We are certainly not aware of one.

The EM algorithm, which will be described in turn, implements hill-climbing in likelihood space. As noticed in the introduction, EM is an iterative algorithm. Let $j = 0, 1, \ldots$ denote the iteration counter. For simplification of the presentation, let us assume that $m_{[-1]}$ denotes the *empty map*, by which we mean a map where each $m_{\langle x, y \rangle}$ is uniformly distributed. EM iterates two steps:

1. **Localization (E-step):** The E-step computes the belief distribution over poses $s^{(0)}, \ldots, s^{(T)}$

$$Bel_{[j]}(s^{(t)}) \quad = \quad P(s^{(t)} \mid m_{[j-1]}, d) \qquad\qquad \forall t = 1, \ldots, T \quad (13.28)$$

conditioned on the map $m_{[j-1]}$ and the data d.

2. **Mapping (M-step):** The M-step calculates the most likely map

$$m_{[j]} \quad = \quad \underset{m}{\mathrm{argmax}}\, P(m \mid Bel_{[j]}(s^{(0)}), \ldots, Bel_{[j]}(s^{(T)}), d) \qquad (13.29)$$

$$= \quad \int \cdots \int \underset{m}{\mathrm{argmax}}\, P(m \mid s^{(0)}, \ldots, s^{(T)}, d)$$

$$Bel_{[j]}(s^{(0)}) \cdots Bel_{[j]}(s^{(T)})\, ds^{(0)}, \ldots, ds^{(T)} \qquad (13.30)$$

In the initial E-step, m is the empty map and the E-step calculates

$$Bel_{[j]}(s^{(t)}) \quad = \quad P(s^{(t)} \mid d) \qquad\qquad \forall t = 1, \ldots, T \qquad (13.31)$$

Since the most likely map can be difficult to compute even if the poses are known, further below we will introduce an additional independence assumption that allows us to decouple the computation in an efficient way. During the optimization, we will also consider probabilistic maps that contain a notion of uncertainty.

13.4.4   The E-step

Expression (13.28) is similar to localization (Chapter 7), but not identical. This is because for $t < T$, $Bel_{[j]}(s^{(t)})$ depends also on data collected at times $t' > t$, i.e., "future" data (from the perspective of time t). With appropriate assumptions, however, we will show that $Bel_{[j]}(s^{(t)})$ can be expressed as the normalized product of two terms:

$$Bel_{[j]}(s^{(t)}) \quad = \quad \eta\, P(s^{(t)} \mid m_{[j-1]}, d^{(0\ldots t)})\, P(s^{(t)} \mid m_{[j-1]}, d^{(t\ldots T)}) \quad (13.32)$$

This is convenient, as each of the terms on the right-hand side can be computed independently using Markov localization, as explained in the previous chapter.

To see, we recall

$$d \quad = \quad d^{(0...t)} \cup d^{(t...T)} \tag{13.33}$$

with

$$d^{(0...t)} \quad = \quad a^{(0)}, o^{(1)}, \ldots, a^{(t-1)}, o^{(t)} \tag{13.34}$$

$$d^{(t...T)} \quad = \quad a^{(t)}, o^{(t+1)}, \ldots, a^{(T-1)}, o^{(T)} \tag{13.35}$$

and note that

$$Bel_{[j]}(s^{(t)}) \quad = \quad P(s^{(t)} \mid m_{[j-1]}, d) \tag{13.36}$$

$$= \quad P(s^{(t)} \mid m_{[j-1]}, d^{(0...t)}, d^{(t...T)}) \tag{13.37}$$

Bayes rule gives us

$$Bel_{[j]}(s^{(t)}) \quad = \quad \frac{P(d^{(0...t)} \mid m_{[j-1]}, s^{(t)}, d^{(t...T)}) \, P(s^{(t)} \mid m_{[j-1]}, d^{(t...T)})}{P(d^{(0...t)} \mid m_{[j-1]}, d^{(t...T)})} \tag{13.38}$$

$$= \quad \eta_5 \, P(d^{(0...t)} \mid m_{[j-1]}, s^{(t)}, d^{(t...T)}) \, P(s^{(t)} \mid m_{[j-1]}, d^{(t...T)}) \tag{13.39}$$

$$= \quad \eta_5 \, P(d^{(0...t)} \mid m_{[j-1]}, s^{(t)}) \, P(s^{(t)} \mid m_{[j-1]}, d^{(t...T)}) \tag{13.40}$$

where η_5 is a normalizing constant. The last transformation exploits the Markov assumption: knowledge of th map $m_{[j-1]}$ and the pose $s^{(t)}$ at time t renders past data $d^{(0...t)}$ and future data $d^{(t...T)}$ independent. We will now apply Bayes rule again to the second term in (13.40), namely $P(d^{(0...t)} \mid m_{[j-1]}, s^{(t)})$, from which we obtain

$$P(d^{(0...t)} \mid m_{[j-1]}, s^{(t)}) \quad = \quad \frac{P(s^{(t)} \mid m_{[j-1]}, d^{(0...t)}) \, P(d^{(0...t)} \mid m_{[j-1]})}{P(s^{(t)} \mid m_{[j-1]})} \tag{13.41}$$

$$= \quad \eta_6 \, P(s^{(t)} \mid m_{[j-1]}, d^{(0...t)}) \, P(d^{(0...t)} \mid m_{[j-1]}) \tag{13.42}$$

Here η_6 is yet another constant. Substituting this back into (13.40) yields

$$Bel_{[j]}(s^{(t)}) \quad = \quad \eta_5 \eta_6 \, P(s^{(t)} \mid m_{[j-1]}, d^{(0...t)}) \tag{13.43}$$

$$P(d^{(0...t)} \mid m_{[j-1]}) \, P(s^{(t)} \mid m_{[j-1]}, d^{(t...T)}) \tag{13.44}$$

Since $P(d^{(0...t)} \mid m_{[j-1]})$ does not depend on $s^{(t)}$, it can be replaced by another constant, which gives us

$$Bel_{[j]}(s^{(t)}) \quad = \quad \eta_7 \, P(s^{(t)} \mid m_{[j-1]}, d^{(0...t)}) \, P(s^{(t)} \mid m_{[j-1]}, d^{(t...T)}) \quad (13.45)$$

Here η_7 is a constant that subsumes η_4, η_5, and $P(d^{(0...t)} \mid m_{[j-1]})$.

Henceforth, we will call

$$\alpha_{[j]}^{(t)} \quad = \quad P(s^{(t)} \mid m_{[j-1]}, d^{(0...t)}) \quad (13.46)$$

$$\beta_{[j]}^{(t)} \quad = \quad P(s^{(t)} \mid m_{[j-1]}, d^{(t...T)}) \quad (13.47)$$

which implies

$$Bel_{[j]}(s^{(t)}) \quad = \quad \eta \, \alpha_{[j]}^{(t)} \, \beta_{[j]}^{(t)} \quad (13.48)$$

The reader may notice that the computation of $\alpha_{[j]}^{(t)}$ is analogous to forward-localization as described in the previous chapter. The computation of $\beta_{[j]}^{(t)}$ can be viewed as "inverse localization," where data collected after time t is used to retroactively estimate the position at time t. The β-values add additional knowledge to the robot's position, typically not captured in localization. They are, however, essential for revising past belief based on sensor data that was received later in time, which is a necessary prerequisite of building large-scale maps.

Computation of the α-values

The computation of the α-values is equivalent to the probabilistic localization algorithm described in the previous chapter with known initial pose $s^{(0)} = \langle 0, 0, 0 \rangle$. For completeness, it is briefly restated here.

Initially, the robot is assumed to be at the center of the global reference frame. Thus, $\alpha_{[j]}^{(0)}$ (for any $j = 0, 1, \ldots$) is initialized by a Dirac distribution centered at $\langle 0, 0, 0 \rangle$:

$$\alpha_{[j]}^{(0)} \quad = \quad P(s^{(0)} \mid m_{[j-1]}, d^{(0...0)}) \quad (13.49)$$

$$= \quad P(s^{(0)}) \quad (13.50)$$

$$
= \quad \begin{cases} 1, & \text{if } s^{(0)} = \langle 0, 0, 0 \rangle \\ 0, & \text{if } s^{(0)} \neq \langle 0, 0, 0 \rangle \end{cases} \tag{13.51}
$$

All other $\alpha_{[j]}^{(t)}$ with $t = 1, \ldots, T$ are computed recursively by Bayes filters (c.f. Chapter **??**):

$$
\begin{aligned}
\alpha_{[j]}^{(t)} &= \quad P(s^{(t)} \mid m_{[j-1]}, d^{(0 \ldots t)}) \tag{13.52} \\
&= \quad P(s^{(t)} \mid m_{[j-1]}, o^{(t)}, a^{(t-1)}, d^{(0 \ldots t-1)}) \tag{13.53} \\
&= \quad \eta_6 \, P(o^{(t)} \mid s^{(t)}, m_{[j-1]}, a^{(t-1)}, d^{(0 \ldots t-1)}) \, P(s^{(t)} \mid m_{[j-1]}, a^{(t-1)}, d^{(0 \ldots t-1)}) \tag{13.54} \\
&= \quad \eta_6 \, P(o^{(t)} \mid s^{(t)}, m_{[j-1]}) \, P(s^{(t)} \mid m_{[j-1]}, a^{(t-1)}, d^{(0 \ldots t-1)}) \tag{13.55}
\end{aligned}
$$

where η_6 is again a probabilistic normalizer, and the rightmost term of (13.55) can be transformed to

$$
\begin{aligned}
& P(s^{(t)} \mid m_{[j-1]}, a^{(t-1)}, d^{(0 \ldots t-1)}) \tag{13.56} \\
&= \quad \int P(s^{(t)} \mid s^{(t-1)}, m_{[j-1]}, a^{(t-1)}, d^{(0 \ldots t-1)}) \, P(s^{(t-1)} \mid m_{[j-1]}, a^{(t-1)}, d^{(0 \ldots t-1)}) \tag{13.57} \\
&= \quad \int P(s^{(t)} \mid a^{(t-1)}, s^{(t-1)}) \, P(s^{(t-1)} \mid m_{[j-1]}, d^{(0 \ldots t-1)}) \, ds^{(t)} \tag{13.58} \\
&= \quad \int P(s^{(t)} \mid a^{(t-1)}, s^{(t-1)}) \, \alpha_{[j]}^{(t-1)} \, ds^{(t)} \tag{13.59}
\end{aligned}
$$

Substituting (13.59) into (13.55) yields the recursive rule for the computation of all $\alpha^{(t)}$ with boundary condition (13.51):

$$
\begin{aligned}
\alpha_{[j]}^{(0)} &= \quad \begin{cases} 1, & \text{if } s^{(0)} = (0, 0, 0) \\ 0, & \text{if } s^{(0)} \neq (0, 0, 0) \end{cases} \tag{13.60} \\
\alpha_{[j]}^{(t)} &= \quad \eta_6 \, P(o^{(t)} \mid s^{(t)}, m_{[j-1]}) \int P(s^{(t)} \mid a^{(t-1)}, s^{(t-1)}) \, \alpha_{[j]}^{(t-1)} \, ds^{(t)} \tag{13.61}
\end{aligned}
$$

for all $t = 1, \ldots, T$. Notice that (13.60) and (13.61) can be computed efficiently from the data d, the model $m_{[j-1]}$, using the motion model $P(s' \mid a, s)$ and the perceptual model $P(s \mid o, m)$.

Computation of the β-values

The computation of β-values is analogous, to that of the α-values, with the exception that it takes place backwards in time and that the boundary conditions differ. The values $\beta_{[j]}^{(T)}$ do not depend on any data. Hence, $\beta_{[j]}^{(T)}$, which expresses the probability that the robot's final position is s, is uniformly distributed:

$$\beta_{[j]}^{(T)} \;=\; \text{uniform} \tag{13.62}$$

All other β-values are computed recursively by applying Bayes filters backwards in time:

$$\beta_{[j]}^{(t)} \;=\; P(s^{(t)} \mid m_{[j-1]}, d^{(t...T)}) \tag{13.63}$$

$$= \int P(s^{(t)} \mid s^{(t+1)}, m_{[j-1]}, d^{(t...T)}) \, P(s^{(t+1)} \mid m_{[j-1]}, d^{(t...T)}) \, ds^{(t+1)} \tag{13.64}$$

$$= \int P(s^{(t)} \mid a^{(t)}, s^{(t+1)}) \, P(s^{(t+1)} \mid m_{[j-1]}, d^{(t+1...T)}, o^{(t+1)}) \, ds^{(t+1)} \tag{13.65}$$

By now, the reader should recognize that the first transformation that led to (13.64) is a result of applying the Theorem of Total Probability, and the second transformation exploits the Markov assumption with some terms reordered.

Following the same rationale as in the derivation of the α-values, we will now transform the last term in (13.65) using Bayes rule and then exploiting the Markov assumption, to obtain

$$P(s^{(t+1)} \mid m_{[j-1]}, d^{(t+1...T)}, o^{(t+1)}) \tag{13.66}$$

$$= \frac{P(o^{(t+1)} \mid s^{(t+1)}, m_{[j-1]}, d^{(t+1...T)}) \, P(s^{(t+1)} \mid m_{[j-1]}, d^{(t+1...T)})}{P(o^{(t+1)} \mid m_{[j-1]}, d^{(t+1...T)})} \tag{13.67}$$

$$= \eta_7 \, P(o^{(t+1)} \mid s^{(t+1)}, m_{[j-1]}, d^{(t+1...T)}) \, P(s^{(t+1)} \mid m_{[j-1]}, d^{(t+1...T)}) \tag{13.68}$$

$$= \eta_7 \, P(o^{(t+1)} \mid s^{(t+1)}, m_{[j-1]}) \, \beta_{[j]}^{(t+1)} \tag{13.69}$$

Substituting (13.69) into (13.65) gives us, together with the boundary condition (13.62):

$$\beta_{[j]}^{(T)} \;=\; \text{uniform} \tag{13.70}$$

$$\beta_{[j]}^{(t)} \;=\; \eta_7 \int P(s^{(t)} \mid a^{(t)}, s^{(t+1)}) \, P(o^{(t+1)} \mid s^{(t+1)}, m_{[j-1]}) \, \beta_{[j]}^{(t+1)} \, ds^{(t+1)} \tag{13.71}$$

Thus, the β values are best computed backwards in time, starting at $t = T$ and back to $T = 0$. For practical purposes, one might want to assume that the motion model is symmetric, i.e., $P(s' \mid a, s) = P(s \mid a, s')$.

First E-step

In the first computation of the E-step ($j = 0$), where no map is available, $P(o \mid s, m_{[-1]})$ is assumed to be distributed uniformly. This is equivalent to ignoring all observations $\{o^{(0)}, o^{(1)}, \ldots o^{(T)}\}$ in the computation of the α- and β-values. The resulting position estimates are only based on the motion commands $\{a^{(0)}, a^{(1)}, \ldots a^{(T-1)}\}$ in the data d. This completes the derivation of the E-step.

13.4.5   The M-step

Recall that the goal of the M-step is to compute the most likely map from the pose beliefs $Bel_{[j]}(s^{(T)}), d$:

$$m_{[j]} \;\; = \;\; \operatorname*{argmax}_{m} P(m \mid Bel_{[j]}(s^{(0)}), \ldots, Bel_{[j]}(s^{(T)}), d) \tag{13.72}$$

To facilitate the generation of the map, we decompose the mapping problem into a collection of local problems, that is, we independently generate local maximum likelihood maps for each $\langle x, y \rangle$-location

$$m_{\langle x,y \rangle, [j]} \;\; = \;\; \operatorname*{argmax}_{m_{\langle x,y \rangle}} P(m_{\langle x,y \rangle} \mid Bel_{[j]}(s^{(0)}), \ldots, Bel_{[j]}(s^{(T)}), d) \tag{13.73}$$

and construct the map $m_{[j]}$ by pasting together these pieces:

$$m_{[j]} \;\; = \;\; \bigcup_{\langle x,y \rangle} m_{\langle x,y \rangle, [j]} \tag{13.74}$$

Strictly speaking, for some sensor types the resulting map does *not* maximize likelihood. For example, a sonar reading of length o can be explained by a single obstacle within its cone; hence, different $\langle x, y \rangle$-locations cannot be mapped independently. However, the independent treatment of different locations is pervasive in the literature, as it transforms the high-dimensional mapping problem (often with more than 10^4 dimensions) into an equal number of independent one-dimensional mapping problems,

which can be solved efficiently. Finding computationally feasible solutions for (13.72) that circumvent this decomposition is an open problem with combinatorial character.

The locality of our decomposition makes it straightforward to compute the maximum likelihood expression (13.73). Recall that $l \in L$ denotes a property (e.g., occupied, unoccupied), and L denotes the set of all properties that the world can have at any location. The maximum likelihood map $m_{\langle x,y \rangle, [j]}$ under fixed position estimates $Bel_{[j]}(s^{(0)}), \ldots, Bel_{[j]}(s^{(T)}$ is computed according to the weighted likelihood ratio

$$
\begin{aligned}
m_{\langle x,y \rangle = l, [j]} &= P(m_{\langle x,y \rangle} = l \mid d, Bel_{[j]}(s^{(0)}), \ldots, Bel_{[j]}(s^{(T)})) \\
&= \frac{\text{Expected \# of times } l \text{ was observed at } \langle x, y \rangle}{\text{Expected \# of times } \textit{anything} \text{ was observed at } \langle x, y \rangle} \quad (13.75)
\end{aligned}
$$

The expectation is taken with respect to the beliefs $Bel_{[j]}(s^{(0)})$ to $Bel_{[j]}(s^{(T)})$. Expression (13.75) follows a frequentist approach, which equates the likelihood that location $\langle x, y \rangle$ has property l with the empirical frequency, weighted appropriately. Intuitively, we just "count" how often the robot saw l at $\langle x, y \rangle$ and divide it by the number of times it saw anything at $\langle x, y \rangle$. This ratio maximizes the likelihood of the data.

To make this more precise, let us now transform (13.75) into a mathematical expression. The enumerator of (13.75) is given by

$$
\sum_{t=0}^{T} \int P(m_{\langle x,y \rangle} = l \mid o^{(t)}, s^{(t)}) \, I_{\langle x,y \rangle \in \text{range}(o^{(t)}, s^{(t)})} \, Bel_{[j]}(s^{(t)}) \, ds^{(t)} \quad (13.76)
$$

This expression cumulates, with the appropriate weighting, the information we have about the map at $\langle x, y \rangle$.

The term $P(m_{\langle x,y \rangle} = l \mid o^{(t)}, s^{(t)})$ specifies the probability that the property of the map at $\langle x, y \rangle$ is l, judging from the t-th sensor measurement $o^{(t)}$ and under the assumption that the robot's pose was $s^{(t)}$. Of course, $o^{(t)}$ might not carry any information about $m_{\langle x,y \rangle}$. This is the case, for example, if the distance between $\langle x, y \rangle$ and $s^{(t)}$ exceeds the robot's perceptual range, or if the sensor measurement is blocked by an obstacle in-between. Hence, the probability is multiplied by an indicator variable $I_{\langle x,y \rangle \in \text{range}(o^{(t)}, s^{(t)})}$. Recall that the function $\text{range}(o, s)$ returns, the set of coordinates covered by the sensor measurement o taken at s. The indicator variable I checks

if the sensor measurement carries any information on the map at $\langle x, y \rangle$:

$$I_{\langle x,y \rangle \in \text{range}(o,s)} \quad = \quad \begin{cases} 1, & \text{if } \langle x, y \rangle \in \text{range}(o, s) \\ 0, & \text{if } \langle x, y \rangle \notin \text{range}(o, s) \end{cases} \tag{13.77}$$

Finally, expression (13.76) sums over all time steps $t = 1, \ldots, T$ and over all positions $s^{(t)}$, weighted by their likelihood $Bel_{[j]}(s^{(t)})$.

For the denominator of (13.75), we have sum (13.76) for *all* $l \in L$, which yields:

$$\sum_{l' \in L} \sum_{t=0}^{T} \int P(m_{\langle x,y \rangle} = l' \mid o^{(t)}, s^{(t)}) \, I_{\langle x,y \rangle \in \text{range}(o^{(t)}, s^{(t)})} \, Bel_{[j]}(s^{(t)}) \, ds^{(t)} \tag{13.78}$$

$$= \quad \sum_{t=0}^{T} \int \left(\sum_{l' \in L} P(m_{\langle x,y \rangle} = l' \mid o^{(t)}, s^{(t)}) \right) I_{\langle x,y \rangle \in \text{range}(o^{(t)}, s^{(t)})} \, Bel_{[j]}(s^{(t)}) \tag{13.79}$$

$$= \quad \sum_{t=0}^{T} \int I_{\langle x,y \rangle \in \text{range}(o^{(t)}, s^{(t)})} \, Bel_{[j]}(s^{(t)}) \, ds^{(t)} \tag{13.80}$$

since, trivially, $\sum_{l' \in L} P(l' \mid \cdot) = 1$. Dividing (13.76) by (13.80) yields the likelihood ratio

$$m_{\langle x,y \rangle = l, [j]} \quad = \quad \frac{\displaystyle\sum_{t=0}^{T} \int P(m_{\langle x,y \rangle} = l \mid o^{(t)}, s^{(t)}) \, I_{\langle x,y \rangle \in \text{range}(o^{(t)}, s^{(t)})} \, Bel_{[j]}(s^{(t)}) \, ds^{(t)}}{\displaystyle\sum_{t=0}^{T} \int I_{\langle x,y \rangle \in \text{range}(o^{(t)}, s^{(t)})} \, Bel_{[j]}(s^{(t)}) \, ds^{(t)}} \tag{13.81}$$

To complete the derivation, we need one last step. The enumerator of (13.81) is a function of a conditional density $P(m_{\langle x,y \rangle} \mid o, s)$. This density can be interpreted as an "inverse" perceptual model, since it determines the state of the world from a sensor measurement and a robot's pose. Inverse perceptual models will be addressed in depth elsewhere in this book. For the time being, it suffices to notice that our decomposition into small, local maps allows us to conveniently write

$$P(m_{\langle x,y \rangle} \mid o, s) \quad = \quad \frac{P(o, \mid s, m_{\langle x,y \rangle}) \, P(m_{\langle x,y \rangle} \mid s)}{P(o, \mid m_{\langle x,y \rangle})} \tag{13.82}$$

Both $P(m_{\langle x,y \rangle} \mid s)$ and $P(o, \mid m_{\langle x,y \rangle})$ can safely be approximated by a constant η_8, so that we have

$$P(m_{\langle x,y \rangle} \mid o, s) = \eta_8 \, P(o, \mid s, m_{\langle x,y \rangle}) \tag{13.83}$$

Notice that this expression is a version of the familiar perceptual model $P(o, \mid s, m)$.

Substituting (13.83) into (13.81) gives us the final equation that is the M-step:

$$
\begin{aligned}
m_{\langle x,y \rangle = l, [j]} &= P(m_{\langle x,y \rangle} = l \mid d, Bel_{[j]}(s^{(0)}), \dots, Bel_{[j]}(s^{(T)}) \\
&= \frac{\displaystyle\sum_{t=0}^{T} \int P(o^{(t)} \mid s^{(t)}, m_{\langle x,y \rangle} = l) \, I_{\langle x,y \rangle \in \mathrm{range}(o^{(t)}, s^{(t)})} \, Bel_{[j]}(s^{(t)}) \, ds^{(t)}}{\displaystyle\sum_{t=0}^{T} \int I_{\langle x,y \rangle \in \mathrm{range}(o^{(t)}, s^{(t)})} \, Bel_{[j]}(s^{(t)}) \, ds^{(t)}}
\end{aligned}
\tag{13.84}
$$

This completes the derivation of the M-step. While Equation (13.84) looks complex, it basically amounts to a relatively straightforward frequentist approach. It counts how often property l was observed for location $\langle x, y \rangle$, divided by the number anything was observed for that location. Each count is weighted by the probability that the robot was at a location s where it could observe something about $\langle x, y \rangle$. Frequency counts are maximum likelihood estimators. Thus, the M-step determines the most likely map at $\langle x, y \rangle$ from the position estimates computed in the E-step. By alternating both steps, the E-step and the M-step, both the localization estimates and the map are gradually improved.

13.4.6   Examples

Example:

13.5   GRID-BASED IMPLEMENTATION

So what is there to know about *implementing* the algorithm **EM_mapping**()? In principle, the algorithm can be married to any of the representations discussed in Chapter 2.5. As in the previous chapter, let us first consider the possibility of implementing

the algorithm **EM_mapping()** using grids. This involves approximating both Both the beliefs $Bel(s)$ and the map M using equally-spaced, metric grids. Consequently, integrals in Table 13.1 are replaced by (finite) sums over the corresponding grid cells.

Under certain conditions, grids are well-suited to implemented the algorithm **EM_mapping()**. Due to the inefficiency of grids, however can be extremely slow and memory-intensive. It can be sped up significantly by various modifications.

- **Data sub-sampling.** Instead of computing a pose belief $Bel(s^{(t)})$ for each point in time $t \in \{0, \dots, T\}$, in practice it often suffices to estimate the pose for a small subset thereof. This is because short-term control/odometry errors are usually small and therefore can be corrected by other and faster means. A simple example is to estimate beliefs $Bel(s)$ only after the robot progressed a certain distance (e.g., 10 meters), assuming that in between odometry is error-free. Alternatively, one might correct for in-between odometry errors by fast, linear interpolation. Data sub-sampling clearly alters the output of the algorithm, but it may be necessary given today's computing constraints.

- **Selective updating.** As described in Chapter **??**, selectively updating the probability distributions can greatly speed up the algorithm. Recall that in selective updating, only those grid cells are considered whose probability $Bel(s)$ is larger than a threshold (e.g., $0.001 \cdot \max_s Bel(s)$). With careful adjustment of the threshold, selective updating only minimally alters the result while speeding up computation by orders of magnitude.

- **Selective memorization.** Hand in hand with selective updating, the memory requirements can be lowered by selectively memorizing only a subset of all values in $Bel(s)$. A straightforward approach is to identify the smallest axes-parallel hypercube in the Grid that contains all cells whose likelihood exceeds the threshold used in selective updating. Alternatively, one can maintain an explicit list of such cells, and only store those whose probability exceeds the threshold.

- **Caching.** Certain quantities, such as the motion model $P(s' \mid a, s)$, have to computed in every iteration of the algorithm. To speed up the computation, one may pre-compute $P(s' \mid a, s)$ for all $a^{(t)}$ in the data set d and memorize this in a set of look-up tables.

- **Exploiting symmetry.** The cached densities $P(s' \mid a, s)$ are highly symmetric. For a fixed a, $P(s \mid a, s')$ might appear to be six-dimensional: three dimensions for $s = \langle x, y, \theta \rangle$ and three dimensions for $s' = \langle x', y', \theta' \rangle$. In practice, however, it suffices to memorize this density for a single value of s: $s = \langle 0, 0, 0 \rangle$. This is because

$$P(s' \mid a, s) \quad = \quad P(\langle x', y', \theta' \rangle \mid a, \langle x, y, \theta \rangle) \tag{13.85}$$

$$= P(\langle \hat{x}, \hat{y}, \hat{\theta} \rangle \mid a, \langle 0, 0, 0 \rangle) \qquad (13.86)$$

with

$$\hat{x} = (x' - x)\cos\theta + (y' - y)\sin\theta \qquad (13.87)$$
$$\hat{y} = (y' - y)\cos\theta - (x' - x)\sin\theta \qquad (13.88)$$
$$\hat{\theta} = \theta' - \theta \qquad (13.89)$$

Obviously, the resulting density $P(\langle \hat{x}, \hat{y}, \hat{\theta} \rangle \mid a, \langle 0, 0, 0 \rangle)$ is only three-dimensional. Further savings can be achieved exploiting symmetries in the motion model.

Empirically, the effect of these modifications will be gigantic! In one experiment, we found that they lowered memory requirements by a factor of more than 10^8 when compared to a simple-minded grid-based implementation of the algorithm **EM_mapping()**, with computational savings at a similar rate. So we recommend you try them out!

Make Figure: Landmark-based maps from Washington, Wean Hall.

Figure **??** shows examples....

Even though we chose landmarks as example, the reader should notice that our algorithm **EM_mapping()** is generally applicable to any type of sensors. Further below. in Chapter **??**, we will present a modified version of this algorithm which has been successfully applied to build maps with one of the most widely used sensors in mobile robotics: sonars.

13.6 LAYERED EM MAPPING

A pivotal problem with the algorithm **EM_mapping()**, as presented in Table 13.1 on page 366, is the fact for some sensors it may generate highly implausible maps during search. Technically, the source of the problem lies in the M-step, and in particular in the decomposition that estimates maximum likelihood map for each location $\langle x, y \rangle$ independently (c.f., Chapter 13.4.5).

Make Figure: Counterexample from Wolfram's ICML paper

[[[Actually, the 2-corridor map is a bad example, since it's based on occupancy grid maps, not the maximum likelihood estimate. Maybe we can generate a new counterexample]]]

To give a concrete example, suppose we apply **EM_mapping()** to a robot equipped with range finders (e.g., sonars), in the hope of building a consistent map of an office environment. If the robot's poses are known, the mapping algorithm might produce maps like the occupancy grid segment shown in Figure **??**. Clearly, this is a nice and crisp map of a corridor segment. The problem arises when the pose of the robot is *not* known well, as is typically the case in the M-step of **EM_mapping()**. For example, if $Bel(s)$ contains two distinct regions with high likelihood, the M-step might generate a map like the one shown in Figure **??**. It is worth noting that for each individual location $\langle x, y \rangle$, the map in Figure **??** correctly estimates the likelihood of occupancy. Nevertheless, the map as a whole misleads the E-step (localization): In some areas, it shows three walls instead of two, and in others the corridor appears to be twice as wide. Configurations like these can lead to erroneous results for the localization in the E-step, if the range-based localization algorithm described in Chapter 7 is applied.

Fortunately, the basic algorithm can be augmented in a way that reduces the damaging effects. We will now describe a *layered* version of the algorithm **EM_mapping()**, called **layered_EM_mapping()**. Technically, this algorithm does not fully sidestep the independence assumption, but it keeps track of certain dependencies, thereby remedying the problem in practice.

13.6.1   Layered Map Representations

Make Figure: ICML Figure 5

The key idea of the algorithm **layered_EM_mapping()** is to represent the map by a *collection of small, local maps*, instead of a monolithic, global map. Each local map is generated from a short data sub-sequence. Figure **??** shows some examples, where each local map is an occupancy grid generated from not more than 10 meters of robot motion. To establish correspondence between the coordinates systems of the local maps, and the global coordinate frame, each local map is annotated by the corresponding transformation. We will refer to this new representation as *layered*, to indicate that the global map is composed of a collection of small, local maps.

Layered maps can be learned with EM. However, our algorithm has to be modified appropriately to account for the fact that maps are now layered. EM with layered maps is graphically illustrated in Figure **??**: Instead of moving back and forth between

pose estimates and a flat map as was the case in **EM_mapping()**, the new algorithm **layered_EM_mapping()** first generated a set of local maps, and uses EM to estimate their location and orientation within the global coordinate frame. As we will see, the advantage of the layered approach is that within the local maps, dependencies between different $\langle x, y \rangle$ locations can be treated correctly. After the termination of EM, the local maps are fused into a single, global map. This approach inherits its name from the fact that two different map representations are used, a flat global map and a collection of local maps.

13.6.2   Local Maps

Local maps are occupancy grid maps acquired over a motion segment of bounded length. Purely to facilitate the notation, let us temporarily assume that the local map is generated from a *single* sensor measurement $o^{(t)}$. This allows us to write

$$m^{(t)} \tag{13.90}$$

for the local map generated from the sensor measurement $o^{(t)}$. The reader may notice that further below, after posing the main algorithm, we will integrate longer data sequences into a local map, which will reduce the computation and lead to more distinct local maps. However, for now we will assume a one-to-one correspondence between time steps t and local maps.

Each local map carries its own, local coordinate system. To map local map coordinates back into the global coordinate frame(and thereby establishing correspondence between the coordinate frames of different local maps), we have to annotate them by a coordinate transformation. Let us write

$$\sigma^{(t)} \tag{13.91}$$

for the coordinate transformation of the local map $m^{(t)}$. Just like robot poses, map poses are specified by three coordinates, $\sigma = \langle x, y, \theta \rangle$. Occasionally, we will refer to $\sigma^{(t)}$ as the *pose* of the t-th local map (poses of maps are essentially the same as robot poses).

In the vein of this book, our approach memorizes a belief distribution over $\sigma^{(t)}$ with each map. This distribution will be denoted

$$Bel(\sigma^{(t)}) \tag{13.92}$$

and is similar to the pose belief $Bel(s^{(t)})$

As we will see below, by representing maps in two layers we eliminate the problems arising from using a single, monolithic map. This is because local maps are not convolved with their pose distribution; instead, the pose distribution is represented explicitly.

13.6.3   The Perceptual Model For Layered Maps

Our layered map representation requires a new definition of the perceptual model $P(o \mid m, s)$. Our approach assumes that given a percept o taken at location s, any two local maps are conditionally independent:

$$P(m^{(t)}, m^{(t')} \mid o, s) \quad = \quad P(m^{(t)} \mid o, s) \, P(m^{(t')} \mid o, s) \tag{13.93}$$

for all $t \neq t'$. This assumption would be correct of none of the local maps ever overlapped. In regions of overlap it is incorrect, and might in turn lead to overly confident maps. Nevertheless, this assumption is essential to keep the computation manageable, and in practice it seems to work well.

The independence assumption allows us to extend our perceptual model to layered maps. Let us first apply Bayes rule to the familiar perceptual model:

$$P(o \mid s, m) \quad = \quad \frac{P(m \mid o, s) \, P(o \mid s)}{P(m \mid s)} \tag{13.94}$$

Notice that neither o nor m depend on s, hence

$$P(o \mid s, m) \quad = \quad \frac{P(m \mid o, s) \, P(o)}{P(m)} \tag{13.95}$$

Under the assumption that—in the absence of any other information—all observations and all maps are equally likely, $P(o)$ and $P(m)$ can both be subsumed into a constant factor, called η:

$$P(o \mid s, m) \;=\; \eta\, P(m \mid o, s) \tag{13.96}$$

Observing that $m = \{(m^{(0)}, \ldots, m^{(T)}\}$, and by virtue of our independence assumption (13.93), we obtain the nice product form:

$$P(m \mid o, s) \;=\; \eta\, P(m^{(0)}, \ldots, m^{(T)} \mid o, s) \tag{13.97}$$

$$\;=\; \prod_{t=1}^{T} P(m^{(t)} \mid o, s) \tag{13.98}$$

We will now expand using Total Probability and factoring out a collection of independences:

$$\prod_{t=1}^{T} P(m^{(t)} \mid o, s) \;=\; \eta \prod_{t=1}^{T} \int P(m^{(t)} \mid o, s, \sigma^{(t)})\, P(\sigma^{(t)} \mid o, s)\, d\sigma^{(t)} \tag{13.99}$$

$$\;=\; \eta \prod_{t=1}^{T} \int P(m^{(t)} \mid o, s, \sigma^{(t)})\, P(\sigma^{(t)})\, d\sigma^{(t)} \tag{13.100}$$

Bayes rule applied to the first term leads to

$$P(m^{(t)} \mid o, s, \sigma^{(t)}) \;=\; \frac{P(o \mid s, m^{(t)}, \sigma^{(t)})\, P(m^{(t)} \mid s, \sigma^{(t)})}{P(o \mid s, \sigma^{(t)})} \tag{13.101}$$

where, by the same logic as above, $P(m^{(t)} \mid s, \sigma^{(t)})$ and $P(o \mid s, \sigma^{(t)})$ can be subsumed into a (different) constant, which gives us:

$$P(o \mid s, m) \;=\; \eta_1 \prod_{t=1}^{T} \int P(o \mid s, m^{(t)}, \sigma^{(t)})\, P(\sigma^{(t)})\, d\sigma^{(t)} \tag{13.102}$$

Here η_1 is again a normalizer, but it is different from the η above. For convenience, we will replace the terms $P(\sigma^{(t)})$ by the beliefs $Bel(\sigma^{(t)})$ obtained while learning the

map, which leads to:

$$P(o \mid s, m) \quad = \quad \eta_1 \prod_{t=1}^{T} \int P(o \mid s, m^{(t)}, \sigma^{(t)}) \, Bel(\sigma^{(t)}) \, d\sigma^{(t)} \qquad (13.103)$$

The term $P(o \mid s, m^{(t)}, \sigma^{(t)})$ is computed using our familiar perceptual model, discussed in Chapter **??**:

$$P(o \mid s, m^{(t)}, \sigma^{(t)}) \quad = \quad P(o \mid \bar{\sigma}^{(t)}, m^{(t)}) \qquad (13.104)$$

where $\bar{\sigma}^{(t)} = \langle \bar{x}^{(t)}, \bar{y}^{(t)}, \bar{\theta}^{(t)} \rangle$ is given by the appropriate coordinate transformation:

$$\bar{x}^{(t)} \quad = \quad x \cos \theta + y \sin \theta - x^{(t)} \qquad (13.105)$$
$$\bar{y}^{(t)} \quad = \quad y \cos \theta - x \sin \theta - x^{(t)} \qquad (13.106)$$
$$\bar{\theta}^{(t)} \quad = \quad \theta - \theta^{(t)} \qquad (13.107)$$

Here we assumed that $s = \langle x, y, \theta \rangle$ and $\sigma^{(t)} = \langle x^{(t)}, y^{(t)}, \theta^{(t)} \rangle$.

Putting everything together, we obtain the *perceptual model for layered maps*:

$$P(o \mid s, m) \quad = \quad \eta_1 \prod_{t=1}^{T} \int P(o \mid m^{(t)}, \bar{\sigma}^{(t)}) \, Bel(\sigma^{(t)}) \, d\sigma^{(t)} \qquad (13.108)$$

The perceptual model (13.108) has some interesting properties. The probability of a sensor measurement o is computed by "localizing" it in each of the local maps (convolved by their pose beliefs), and them multiplying the results for all local maps. Thus, the sensor model is essentially the product of the familiar model defined over flat, monolithic maps. In practice, maps which cannot overlap with the robot's position can be safely eliminated, thereby speeding up the computation while not affecting the final result.

13.6.4   EM with Layered Maps

The E-Step

With the appropriate perceptual model in place, the E-step is analogous to the flat mapping case. We recall from (13.32) and (13.48) that $Bel_{[j]}(\sigma^{(t)})$ can be factorized conveniently (j is the iteration index) into

$$Bel_{[j]}(\sigma^{(t)}) \quad = \quad \eta \underbrace{P(\sigma^{(t)} \mid m_{[j-1]}, d^{(0...t)})}_{\alpha_{[j]}^{(t)}} \underbrace{P(\sigma^{(t)} \mid m_{[j-1]}, d^{(t...T)})}_{\beta_{[j]}^{(t)}} \quad (13.109)$$

for all $t = 1, \ldots, T$. Using our new perceptual model, which requires us to substitute $P(o \mid s, m)$ by $\prod_{t'=0}^{T} P(o \mid s, m^{(t')}, \bar{\sigma}^{(t')}) \, Bel_{[j-1]}(\sigma^{(t')})$ in the original equations for computing the the distributions of $\alpha_{[j]}^{(t)}$ and $\beta_{[j]}^{(t)}$, (13.60) and (13.61), we obtain:

$$\alpha_{[j]}^{(0)} \quad = \quad \left\{ \begin{array}{l} 1, \text{ if } s^{(0)} = (0,0,0) \\ 0, \text{ if } s^{(0)} \neq (0,0,0) \end{array} \right. \quad (13.110)$$

$$\alpha_{[j]}^{(t)} \quad = \quad \eta \left[\prod_{t'=1}^{T} P(o^{(t)} \mid s^{(t)}, m^{(t)}, \bar{\sigma}^{(t')}) \, Bel_{[j-1]}(s^{(t)}) \right]$$
$$\int P(s^{(t)} \mid a^{(t-1)}, s^{(t-1)}) \, \alpha_{[j]}^{(t-1)} \, ds^{(t)} \quad (13.111)$$

and

$$\beta_{[j]}^{(T)} \quad = \quad \text{uniform} \quad (13.112)$$

$$\beta_{[j]}^{(t)} \quad = \quad \eta_7 \int P(s^{(t)} \mid a^{(t)}, s^{(t+1)}) \left[\prod_{t'=1}^{T} \right.$$
$$\left. P(o^{(t+1)} \mid s^{(t+1)}, m^{(t)}, \bar{\sigma}^{(t')}) \, Bel(s^{(t)})(s^{(t)}) \right] \beta_{[j]}^{(t+1)} \, ds^{(t+1)} \quad (13.113)$$

for all $t = 1, \ldots, T$.

The M-Step with Deterministic Annealing

The M-step calculates the most likely poses of the local maps. In the most generic setting, the M-step calculates

$$s^{(t)} \quad = \quad \underset{s^{(t)}}{\operatorname{argmax}} \, Bel_{[j]}(s^{(t)}) \quad (13.114)$$

$$= \operatorname*{argmax}_{s^{(t)}} \alpha^{(t^{(t)})} \, \beta^{(t^{(t)})} \tag{13.115}$$

for all $t = 1, \ldots, T$. Such an approach would be the natural extension of plain EM to layered maps.

Unfortunately, this approach is problematic in practice. EM is a hill climbing method, which only converges to *local* maxima. If the odometric error is large, the initial map will be erroneous, and subsequent iterations of EM might not be able to recover.

The danger of getting stuck in a local maximum can be reduced significantly by a modified M-step. Instead of keeping track of the most likely pose of each map, our approach generates a distribution over poses that slowly converges to the most likely one. Thus, our approach generates a "soft" version of the maximum likelihood estimate. Over time, it gradually reduces the softness, until it finally generates the maximum likelihood pose. This approach is known as *deterministic annealing*.

In detail, the M-step generates a distribution over poses, denoted μ:

$$Bel_{[j]}(s^{(t)}) \;\; = \;\; \eta \, \left(\alpha^{(t^{(t)})} \beta^{(t^{(t)})} \right)^{\frac{1}{\sigma}} \tag{13.116}$$

Here η is a (different) normalizer that ensures that the probabilities integrate to 1. The parameter σ is a control parameter in $(0, 1]$ which, in analogy to the rich literature on annealing, will be referred to as *temperature*. When $\sigma = 1$, the full distribution over all poses is retained. When $\sigma = 0$, our approach is equivalent to the maximum likelihood assignment (13.115).

The temperature σ is slowly driven to zero—a process often referred to as *cooling*. In our approach, σ is initialized with 1 and annealed towards zero using an exponential cooling schedule. The effect annealing is to avoid early commitment to a single map. Instead, one can think of our approach as moving from density estimation (over the pose parameters in the map) to maximum likelihood estimation. The reader should notice that our approach is not the first to employ annealing to avoid local maxima in EM [14, 34].

Post-Processing

To come up with a single global map of the environment, the local maps $m^{(t)}$ are integrated based on their final maximum likelihood poses

$$\underset{s^{(t)}}{\operatorname{argmax}} \, Bel(s^{(t)}) \;\; = \;\; \underset{s^{(t)}}{\operatorname{argmax}} \, \alpha^{(t^{(t)})} \, \beta^{(t^{(t)})} \tag{13.117}$$

The local maps, along with their poses, are fed into the algorithm **occupancy_grid()** which then produces a single, global map.

13.6.5   The Layered EM Mapping Algorithm

Figure 13.2 shows the resulting algorithm, called **layered_EM_mapping()**.

Make Figure: Show ICML figures

13.6.6   Examples

More specifically, the data is separated into a stream of sub-datasets denoted

$$d \;\; = \;\; \bigcup_i d^{[i]} \tag{13.118}$$

during each of which the robot does not advance more than a pre-specified distance (e.g., 5 meters). The superscript is not to be confused with the superscript $^{(t)}$ or the subscript $_{[j]}$ in EM.

Let us denote the local maps

$$m^{[i]} \tag{13.119}$$

They are built under the assumption that the odometry error in each data segment $d^{[i]}$ is small enough to be neglected; thus, in practice one has to adjust the size of each $d^{[i]}$ in accordance to the quality of the robot's odometry.

1:        **Algorithm layered_EM_mapping(d):**

2:            *initialize m by* uniform-map

3:            $\sigma = 1.0$

4:            *for $t = 0$ to T do*

5:                $m^{(t)} = $ **occupancy_grid**($o^{(t)}$)

6:                $Bel(s^{(t)}) = \text{uniform}()$

7:            *repeat until satisfied*

8:                $\alpha^{(0)} = \text{Dirac}(\langle 0, 0, 0 \rangle)$

9:                *for $t = 1$ to T do*

10a: $\alpha^{(t)} = \eta \left[\prod_{t'=0}^{T} P(o^{(t)} \mid s^{(t)}, m^{(t')}, s^{(t')}) \, Bel(s^{(t')}) \right] \cdot$

10: $\cdot \int P(s^{(t)} \mid a^{(t-1)}, s^{(t-1)}) \, \alpha^{(t-1)} \, ds^{(t)}$

11:                $\beta^{(T)} = \text{uniform}()$

12:                *for $t = T - 1$ down to 0 do*

13a: $\beta^{(t)} = \eta \int P(s^{(t)} \mid a^{(t)}, s^{(t+1)}) \left[\prod_{t'=0}^{T} \right.$

13b: $\left. P(o^{(t+1)} \mid s^{(t+1)}, m^{(t')}, s^{(t')}) \, Bel(s^{(t')}) \right] \, \beta^{(t+1)} \, ds^{(t+1)}$

14:                *for $t = 1$ to T do*

15:                    $Bel(s^{(t)}) = \eta \, \left(\alpha^{(t)} \, \beta^{(t)} \right)^{\frac{1}{\sigma}}$

16:                $\sigma = .9\sigma$

17:            $\bar{m} = \{ m^{(0)}, \text{argmax}_s \, Bel(s^{(0)}), m^{(1)}, \ldots, m^{(T)}, \text{argmax}_s \, Bel(s^{(T)}) \}$

18a: $m = $ **occupancy_grid**($\{ m^{(0)}, \text{argmax}_{s^{(0)}} \, Bel(s^{(0)}), m^{(1)}, \ldots,$

18b: $m^{(T)}, \text{argmax}_{s^{(T)}} \, Bel(s^{(T)}) \}$)

19:            *return m*

Table 13.2   The layered EM map learning algorithm.

13.7   SUMMARY

- The EM algorithm builds maps by maximizing likelihood. It has been shown to build large-scale metric maps, thereby resolve ambiguities and exploiting all information in the data.

■

13.8   BIBLIOGRAPHICAL REMARKS

The problem of map acquisition has attracted a lot of attention in the last decade. The literature commonly distinguishes two types of maps, *robot-centered* and *world-centered* maps.

- Robot-centered maps memorize the sensor measurements a robot is expected receive at each location (or pose). In the general case, such maps are three-dimensional, parameterized by the robot's three-dimensional pose. In certain cases, however, two dimensions are sufficient. Example include robots whose sensor measurements are rotationally invariant (e.g., a "general brightness" sensor), or robots for which a sensor measurement taken with a specific heading direction θ is sufficient to compute sensor measurements under arbitrary heading directions θ (e.g., omni-cams, which cover a 360 degree field of view).

- In contrast, world-centered maps memorize the location of surrounding objects in world coordinates. For robots equipped with sensors that operate in a two-dimensional plane, such as most robots equipped with range finders, it suffices to memorize two-dimensional maps. Other sensors, such as cameras, typically require three-dimensional maps.

The advantage of robot-centered maps lies in the fact that they are almost universally applicable, no matter what sensors being used. Unlike world-centered maps, robot-centered maps do not require that correspondence is established between sensor input and the location of objects in the outside world. However, this advantage comes at a price. World-centered maps can usually be constructed from much less data than robot-centered maps. Why? Knowledge of the relative location of objects makes it possible to deduce how the world looks from nearby places. For example, picture a robot with a laser range finder located in a convex room, such that all walls are within the limits of the sensor's perceptual range. A single scan can be sufficient to determine the location of all walls; and geometry can be applied to predict sensor scans from other viewpoints. This advantage is not shared by robot-centered maps, since the transformation from one sensor view to another is, in the absence of geometry, far from trivial.

A second taxonomy, frequently discussed in the literature, distinguishes topological and metric approaches.

■   Topological algorithms represent maps as graphs, where edges correspond to places, and arcs to paths between them. For example, places might be locations with specific distinguishing features, such as intersections and T-junctions in office building, and arcs may correspond to specific behaviors or motion commands that enable the robot to move from one location to another, such as wall following. Recently, it has become popular to augment topological maps with metric information (relative distance, angle) to facilitate to disambiguation of similarly looking places.

■   Metric approaches embed maps into the plane. Locations in metric maps are memorized along with their global coordinates, typically expressed by x-y coordinates and, in certain case, orientations θ. At first glance, it appears that metric maps are more difficult to construct, since they have to be globally consistent. In practice, however, metric information has been found to be seminal for the disambiguation of alike-looking places, and the metric paradigm has clearly led to larger and more consistent maps.

Coincidentally, there is a strong correspondence in the literature between both taxonomies. Most topological approaches rely on robot-centered maps, and most metric approaches build world-centered maps.

14

FAST INCREMENTAL MAPPING ALGORITHMS

14.1 MOTIVATION

In the previous two chapters, two principled solutions to the concurrent mapping and localization were presented. Both solutions are statistically sound. They both incorporate uncertainty in robot poses and maps. However, both suffered important deficiencies. The Kalman filter approach cannot cope with ambiguities in perception, that is, it requires that the data association problem is solved. It is also limited to a relatively small number of features. Both limitations preclude its application to building high-resolution structural maps. The EM approach, on the other hand, provides a sound solution to the data association problem. However, it does not work in real-time, and all data has to be memorized. This is an important limitation. It precludes, for example, using this algorithm for real-time exploration and mapping of unknown environments.

This chapter discusses a family of algorithms that can generate maps in near real-time, while simultaneously coping with the data association problem. The core of the method is non-probabilistic. Instead of representing the uncertainty in map estimates, it only calculates a single map, which is obtained as the maximum likelihood solution to the incremental mapping problem. Clearly, the lack of a representation of uncertainty in the estimates causes problems. In particular, the core algorithm is incapable of mapping cyclic environments. Cycles are generally difficult to map, since the robot has to reconnect to a previously mapped area from a different direction. To overcome these problems, we augment the algorithm with a probabilistic estimator, a posterior estimator over robot poses. This estimator models the error of the maximum likelihood method in pose space. It makes it possible to handle problems with large cycles, while simultaneously generating maps in near real-time. The algorithms presented here have practical significance. For specific type of sensors, such as laser

range finders, these algorithm have been shown to produce maps of relatively large environments in real-time. They are also relatively easy to implement, and therefore have been popular in mobile robotics.

Towards the end of this chapter, we will discuss extensions to two important mapping problems that are mostly unexplored in robotics: The multi-robot mapping problem, in which a team of robots collaboratively acquire a map, and the 3D mapping problem, in which a robot seeks to generate a full three-dimensional map of its environment. As we will see, algorithms for multi-robot mapping and 3D mapping can be implemented with little effort on top of the algorithms discussed in this chaptr.

The chapter is organized as follows.

- Section 14.2 defines the incremental mapping problem and derives the basic algorithm for buiding maps incrementally using the maximum likelihood estimator.

- Section 14.3 discusses gradient descent strategies fo hill climbing in log likelihood space, with the goal of finding maximum likelihood maps in continuous search spaces. It provides concrete gradient descent algorithms for some of the perceptual models and motion models discussed in this book.

- Section 14.4 introduces the idea of simultaneously estimating a posterior over robot poses, and shows how to accommodate large residual errors when closing a cycle. It presents the main algorithm in this chapter, which combines incremental maximum likelihood estimation with a MCL-style posterior estimator.

- Section 14.5 discusses extensions of the basic algorithm to the collaborative multi-robot problem. The resulting algorithm is capable of fusing data from multiple robot platform whose initial pose relative to each other is unknown. However, they assume that there is a designated team leader; and other robots start off in the map built by the team leader.

- Finally, Section 14.6 presents an extension of the basic algorithm that enables a robot to build a three-dimensional map of the environment. These 3D maps combine structural and texture information, and lead to models similar to the ones used in video games. They are acquired using a robot equipped with two laser range finders and a panoramic camera.

14.2 INCREMENTAL LIKELIHOOD MAXIMIZATION

The rationale behind both algorithms is that of likelihood maximization. Instead of computing a posterior over maps—which is the central objective in the next chapter— the algorithms described here seek to compute the most likely map only, along with the most likely robot poses. This is formally denoted by:

$$\operatorname*{argmax}_{m,s^{(0)}\ldots,s^{(t)}} \ P(m, s^{(0)} \ldots, s^{(t)} \mid d) \tag{14.1}$$

Unfortunately, as we will see in the following two chapters, computing this expression is highly intractable. Thus, the approaches described in this chapter implement *incremental* maximum likelihood estimators. They compute a sequence of poses and maps

$$\{m^{(0)}, s^{(0)}\}, \{m^{(1)}, s^{(1)}\}, \{m^{(2)}, s^{(2)}\}, \ldots, \tag{14.2}$$

each by appending the most recent sensor measurement to the growing map. Notice that these estimates are not probabilistic! Instead of calculating a distribution over maps and poses, here we are only concerned with generating a single map, and a single pose. This restriction, which greatly reduces the computational burden when compared to a fully probabilistic approach, is the main source of brittleness of incremental maximum likelihood mapping. Further below, we will add a probabilistic pose estimator that will enable us to recover from certain estimation errors.

The first question we have to ask is: How can a robot find such an incremental sequence of maps and poses? Let us begin by restating the basic Bayes filter, which is at the heart of so many algorithms described in this book:

$$
\begin{aligned}
Bel(s^{(t)}) &= P(s^{(t)} \mid d) \\
&= \eta \, P(o^{(t)} \mid s^{(t)}, m) \int P(s^{(t)} \mid a^{(t-1)}, s^{(t-1)}, m) \, Bel(s^{(t-1)}) \, ds^{(t-1)}
\end{aligned}
\tag{14.3}
$$

In incremental concurrent mapping and localization, we seek to compute the belief over maps as well as the belief over poses. Thus, we have to compute the belief over maps and poses. Additionally, it is convenient to index the maps by time, since in the incremental approach we will generate a whole sequence of maps. Hence the Bayes

filter for concurrent mapping and localization looks as follows:

$$
\begin{aligned}
Bel(s^{(t)}, m^{(t)}) &= P(s^{(t)}, m^{(t)} \mid d) \\
&= \eta \, P(o^{(t)} \mid s^{(t)}, m^{(t)}) \int \int P(s^{(t)}, m^{(t)} \mid a^{(t-1)}, s^{(t-1)}, m^{(t-1)}) \\
&\qquad\qquad Bel(s^{(t-1)}, m^{(t-1)}) \, ds^{(t-1)} \, dm^{(t-1)}
\end{aligned}
\tag{14.4}
$$

This important equation extends Bayes filters to estimating maps. Unfortunately, Bayes filtering is inapplicable in practice. While Bayes filters can be calculated for localization alone with relative ease, this is not the case any longer in the concurrent mapping and localization problem. The reason for this is simple: tractability. While a robot pose is a three-dimensional quantity, a map is often described by many thousands of values.

Maximum likelihood estimation is a simpler problem than the full posterior computation. In maximum likelihood estimation, we seek to compute the maximum of the posterior probability:

$$
\langle s^{(t),*}, m^{(t),*} \rangle = \underset{s^{(t)}, m^{(t)}}{\operatorname{argmax}} \, Bel(s^{(t)}, m^{(t)})
\tag{14.5}
$$

Clearly, the maximum likelihood estimate carry a notion of its own uncertainty; however, as we will see in this and future chapters, computing the map that maximizes likelihood is a challenging problem. While the maximum likelihood estimator is generally easier to compute than the full posterior—after all, this is 'just' the mode of the posterior—even this problem is highly intractable. *Incremental* maximum likelihood estimators decompose the maximum likelihood estimation problem into a sequence of local, tractable problems. More specifically, they assume at each point in time t, one is readily a map and a pose estimate from the previous time step, that is, which incorporate all data up to time $t-1$. They then calculate the maximum likelihood map and pose at time t, an operation that can be performed efficiently.

Let us describe this a bit more formally. The incremental maximum likelihood estimator maximizes the following conditional belief:

$$
Bel(s^{(t)}, m^{(t)} \mid s^{(t-1)}, m^{(t-1)}) = P(s^{(t)}, m^{(t)} \mid d, s^{(t-1)}, m^{(t-1)})
\tag{14.6}
$$

which is the probability of the map and pose at time t given the data and the map and pose one time step earlier. The Markov assumption renders data gathered before time

$t-1$ conditionally independent of the map and pose at time t given the map and pose at time $t-1$. Hence, we can write

$$Bel(s^{(t)}, m^{(t)}|s^{(t-1)}, m^{(t-1)}) \quad = \quad P(s^{(t)}, m^{(t)} \mid o^{(t)}, a^{(t-1)}, s^{(t-1)}, m^{(t-1)}) \quad (14.7)$$

that is, only the data gathered between time $t-1$ and t have to be considered. Notice the resulting incremental estimator, which is being maximized here, is much simpler than the one in (14.4) above:

$$Bel(s^{(t)}, m^{(t)}|s^{(t-1)}, m^{(t-1)})$$
$$= \quad \eta \, P(o^{(t)} \mid s^{(t)}, m^{(t)}) \, P(s^{(t)}, m^{(t)} \mid a^{(t-1)}, s^{(t-1)}, m^{(t-1)}) \quad (14.8)$$

Why is this simpler? In comparison with (14.4), one does not have to integrate over maps and poses any longer. In fact, the incremental posterior probability (14.8) can be computed in closed form, making it extremely convenient for on-line estimation. However, the question remains as to how incremental maximum likelihood approaches obtain pose estimates and maps at each point in time, and why it suffices to keep track of a single pose and map only, instead of a distribution over poses and maps.

The first question is easily answered. Incremental maximum likelihood algorithms compute the map and pose pose at each time t by maximizing (14.8):

$$\langle s^{(t)}, m^{(t)} \rangle \quad = \quad \underset{\bar{s}^{(t)}, \bar{m}^{(t)}}{\operatorname{argmax}} \, Bel(\bar{s}^{(t)}, \bar{m}^{(t)}|s^{(t-1)}, m^{(t-1)}) \quad (14.9)$$

Even this calculation can be difficult, since the robot operates in a continuum and hence the space of all maps and poses cannot be searched efficiently. However, as we will see below, efficient hill climbing methods exist that, given a good starting point, tend to converge to a global optimum quickly. The resulting algorithm has the nice property that it can be applied in real time.

The second question is more difficult to answer. By retaining only the most likely map and pose at each point in time, the algorithms described in this chapter are brittle. In certain environments, the likelihood function decomposes nicely and the brittleness can be tolerated, In others, such as environments with large cyclic corridors, the decomposition is invalid and the resulting maps are not only incorrect, but as a whole score poorly in their overall likelihood—despite the fact that the stepwise likelihood has been maximized. We will explicitly discuss such cases below, and offer a fix by

1:          **Algorithm incremental_ML_mapping(o, a, s, m):**

2:              *set $s' = \mathrm{argmax}_{\bar{s}} \, P(\bar{s}|a, s)$*

3:              *repeat until satisfied*

4:                  $s' \longleftarrow s' + \kappa \nabla_{s'} \left[\log P(o|s', m) + \log P(s'|a, s) \right]$

5:              $m' \longleftarrow m$ *with* $\langle s', o \rangle$

6:              *return* $\langle m', s' \rangle$

Table 14.1   The basic incremental maximum likelihood algorithm for concurrent mapping and localization. It accepts as input a map and a robot pose, along with an action idem and a sensor easurement. It outputs an updated map and pose estimate. Notice that none of the representations are probabilistics, making this algorithm brittle in practice.

including a posterior estimator for the robot's pose, along with an algorithm that performs occasional adjustments. Unfortunately, additional brittleness is introduced by the hill-climbing nature of the maximum likelihood estimator. If the estimator gets stuck in a local minimum, the algorithms described here cannot recover. In practice, this severely limits the types and sizes of environments that can be mapped with the incremental maximum likelihood approach. Nevertheless, for accurate sensors relatively large and accurate maps can be built in real-time, giving this algorithm some practical importance.

14.3   MAXIMUM LIKELIHOOD AS GRADIENT DESCENT

14.3.1   Search in Pose Space

We will now consider the question as to how to maximize the incremental likelihood function (14.9). This incvolves search in the space of all poses and all maps at time t.

However, is we employ a fixed routine for map building (e.g., occupancy grid mapping), it suffices to search the space of all poses. This reduces the computation significantly, as the space of poses if three-dimensional, where the space of all maps is huge. To see, let us consider the sigutation where the robot pose is known. Following

our assumptions, the t-th map $m^{(t)}$ is generated from the previous map $m^{(t-1)}$ by a fixed, deterministic routing (e.g., occupancy grid mapping). Thus, the only parameter that remains to be optimized is the pose:

$$s^{(t)} = \underset{\bar{s}^{(t)}}{\mathrm{argmax}}\, Bel(\bar{s}^{(t)}, \bar{m}^{(t)}(\bar{s}^{(t)}, o^{(t)}, m^{(t-1)})|s^{(t-1)}, m^{(t-1)}) \quad (14.10)$$

Here $\bar{m}^{(t)}(\bar{s}^{(t)}, o^{(t)}, m^{(t-1)})$ denotes the map that is obtained by incrementally incorporating the observation $o^{(t)}$ to the map $m^{(t-1)}$ at the alleged pose $\bar{s}^{(t)}$. A little thought should convince the reader that one can safely replace this map by the most recent sensor measurement $o^{(t)}$ only (since the pose is known), giving us

$$s^{(t)} = \underset{\bar{s}^{(t)}}{\mathrm{argmax}}\, Bel(\bar{s}^{(t)}, o^{(t)}|s^{(t-1)}, m^{(t-1)}) \quad (14.11)$$

which, according to the definition of the incremental belief (14.8) can be spelled out as follows:

$$s^{(t)} = \underset{\bar{s}^{(t)}}{\mathrm{argmax}}\, P(o^{(t)} \mid \bar{s}^{(t)}, m^{(t-1)})\, P(\bar{s}^{(t)} \mid a^{(t-1)}, s^{(t-1)}, m^{(t-1)})$$

$$(14.12)$$

or in log-likelihood form:

$$s^{(t)} = \underset{\bar{s}^{(t)}}{\mathrm{argmax}}\, \log P(o^{(t)} \mid \bar{s}^{(t)}, m^{(t-1)}) + \log P(\bar{s}^{(t)} \mid a^{(t-1)}, s^{(t-1)}, m^{(t-1)})$$

$$(14.13)$$

To summarize, we have exploited the fact that growing the map is done by a fixed routine, which saves us the effort of searching the space of all possible maps. The resulting maximum likelihood expression only searches in the space of all poses.

The resulting expression (14.13) is now simple enough to be searched directly. In particular, we need to search the space of all poses at time t to find the pose that maximizes the incremental posterior. A common strategy for searching continuous spaces is gradient descent in log likelihood space. Luckily, all of the probabilities in (14.13) are differentiable. The gradient of the log of the probability on the right-hand side of (14.13) with respect to the desired pose is commonly written as

$$\nabla_{s^{(t)}}\, \log P(o^{(t)} \mid \bar{s}^{(t)}, m^{(t-1)}) + \log P(\bar{s}^{(t)} \mid a^{(t-1)}, s^{(t-1)}, m^{(t-1)}) \quad (14.14)$$

Since gradients are additive, we can decouple the gradient of the perceptual probability from that of the motion model:

$$\nabla_{s^{\bar{(t)}}} \log P(o^{(t)} \mid \bar{s}^{(t)}, m^{(t-1)}) + \nabla_{s^{\bar{(t)}}} \log P(\bar{s}^{(t)} \mid a^{(t-1)}, s^{(t-1)}, m^{(t-1)}) \quad (14.15)$$

Gradient descent incrementally changes the estimate $s^{\bar{(t)}}$ in the direction of the gradient. That is, we apply the following search algorithm

$$
\begin{aligned}
s^{\bar{(t)}} \quad \leftarrow \quad & s^{\bar{(t)}} + \kappa \; \left[\nabla_{s^{\bar{(t)}}} \; \log P(o^{(t)} \mid \bar{s}^{(t)}, m^{(t-1)}) + \right. \\
& \left. \nabla_{s^{\bar{(t)}}} \log P(\bar{s}^{(t)} \mid a^{(t-1)}, s^{(t-1)}, m^{(t-1)}) \right]
\end{aligned}
\quad (14.16)
$$

Here $\kappa > 0$ is a step size close to zero that is required for gradient descent. The starting point for gradient descent is obtained by simply applying the most recent action item (odometry reading) to the pose:

$$s^{(t)} \quad = \quad \underset{\bar{s}^{(t)}}{\operatorname{argmax}} \, P(\bar{s}^{(t)} \mid a^{(t-1)}, s^{(t-1)}) \quad (14.17)$$

which can be computed using the kinematic motion models discussed in Chapter 5 with noise parameters set to zero. Gradient descent then applies the update rule (14.16) repeatedly, until a convergence criterion is met.

14.3.2   Gradient Calculation

What remains to be discussed are the technical details of calculating the desired gradients (14.16). Before we give details for specific perceptual models and motion models, it is important to emphasize that the calculation of gradients of differentiable functions is a purely mechanical exercise, though sometimes tedious. Gradients of log likelihood functions often look complex; however, they are obtained by differentiating any of the motion models or perceptual models discussed in this book, using the well-known rules for differentiation. The models discussed in this book are all piecewise differentiable, hence gradients can be calculated for all of them.

Table 14.2 shows an algorithm for calculating the first derivative of the log-perceptual model

$$\nabla_{\bar{s}^{(t)}} \; \log P(o^{(t)} \mid \bar{s}^{(t)}, m^{(t-1)}) \quad (14.18)$$

1:       **Algorithm first_derivative_log_range_finder_model2(o, s, m):**

2:       $dx = dy = d\theta = 0$

3:       *for all $o_k \in o$ do*

4:         *if $o_k < o_{\max} - o_{\mathrm{small}}$*

5:           $x_{o_k} = x + x_k \cos\theta - y_k \sin\theta + o_k \cos(\theta + \theta_k)$

6:           $y_{o_k} = y + y_k \cos\theta + x_k \sin\theta + o_k \sin(\theta + \theta_k)$

7:           $\frac{\partial x_{o_k}}{\partial \theta} = -x_k \sin\theta - y_k \cos\theta - o_k \sin(\theta + \theta_k)$

8:           $\frac{\partial y_{o_k}}{\partial \theta} = -y_k \sin\theta + x_k \cos\theta + o_k \cos(\theta + \theta_k)$

9:           $\langle \bar{x}, \bar{y} \rangle = \mathrm{argmin}_{x',y'} \left\{ (x_{o_k} - x')^2 + (y_{o_k} - y')^2 \mid \langle x', y' \rangle \text{ occupied} \right\}$

10:           $dist^2 = (x_{o_k} - \bar{x})^2 + (y_{o_k} - \bar{y})^2$

11:           $\frac{\partial dist^2}{\partial x} = 2(x_{o_k} - \bar{x})$

12:           $\frac{\partial dist^2}{\partial y} = 2(y_{o_k} - \bar{y})$

13:           $\frac{\partial dist^2}{\partial \theta} = 2 \left[(x_{o_k} - \bar{x})\frac{\partial x_{o_k}}{\partial \theta} + (y_{o_k} - \bar{y})\frac{\partial y_{o_k}}{\partial \theta} \right]$

14:           $a = z_{\mathrm{hit}} \frac{1}{\sqrt{2\pi\sigma_{\mathrm{hit}}^2}}$

15:           $b = -\frac{1}{2}\frac{dist^2}{\sigma_{\mathrm{hit}}^2}$

16:           $c = \frac{1 - z_{\mathrm{hit}}}{o_{\max}}$

17:           $\frac{\partial b}{\partial x} = -\frac{1}{2\sigma_{\mathrm{hit}}^2}\frac{\partial dist^2}{\partial x}$

18:           $\frac{\partial b}{\partial y} = -\frac{1}{2\sigma_{\mathrm{hit}}^2}\frac{\partial dist^2}{\partial y}$

19:           $\frac{\partial b}{\partial \theta} = -\frac{1}{2\sigma_{\mathrm{hit}}^2}\frac{\partial dist^2}{\partial \theta}$

20:           $\log q = \log(ae^b + c)$

21:           $\frac{\partial \log q}{\partial x} = \frac{ae^b}{c + ae^b}\frac{\partial b}{\partial x}$

22:           $\frac{\partial \log q}{\partial y} = \frac{ae^b}{c + ae^b}\frac{\partial b}{\partial y}$

23:           $\frac{\partial \log q}{\partial \theta} = \frac{ae^b}{c + ae^b}\frac{\partial b}{\partial \theta}$

24:           $dx += \frac{\partial \log q}{\partial x}$

25:           $dy += \frac{\partial \log q}{\partial y}$

26:           $d\theta += \frac{\partial \log q}{\partial \theta}$

27:       *return $\langle dx, dy, d\theta \rangle$*

Table 14.2   Algorithm for computing the first derivative of the log probability computed by the range finder model in Table 6.3 on page 143.

1:     **Algorithm first_derivative_log_motion_model_odometry**(s', a, s)**:**

2:     $\delta_{\text{rot1}} = \text{atan2}(\bar{y}' - \bar{y}, \bar{x}' - \bar{x}) - \bar{\theta}$

3:     $\delta_{\text{trans}} = \sqrt{(\bar{x} - \bar{x}')^2 + (\bar{y} - \bar{y}')^s}$

4:     $\delta_{\text{rot2}} = \bar{\theta}' - \bar{\theta} - \delta_{\text{rot1}}$

5:     $\hat{\delta}_{\text{rot1}} = \text{atan2}(y' - y, x' - x) - \theta$

6:     $\hat{\delta}_{\text{trans}} = \sqrt{(x - x')^2 + (y - y')^2}$

7:     $\hat{\delta}_{\text{rot2}} = \theta' - \theta - \hat{\delta}_{\text{rot1}}$

8:     $\frac{\partial \hat{\delta}_{\text{rot1}}}{\partial x'} = ???; \frac{\partial \hat{\delta}_{\text{rot1}}}{\partial y'} = ???$

9:     $\frac{\partial \hat{\delta}_{\text{trans}}}{\partial x'} = \frac{x' - x}{\hat{\delta}_{\text{trans}}}; \frac{\partial \hat{\delta}_{\text{trans}}}{\partial y'} = \frac{y' - y}{\hat{\delta}_{\text{trans}}}$

10:    $\text{arg}_{11} = \delta_{\text{rot1}} - \hat{\delta}_{\text{rot1}}; \text{arg}_{12} = \alpha_1 \hat{\delta}_{\text{rot1}} + \alpha_2 \hat{\delta}_{\text{trans}}$

11:    $\text{arg}_{21} = \delta_{\text{trans}} - \hat{\delta}_{\text{trans}}; \text{arg}_{22} = \alpha_3 \hat{\delta}_{\text{trans}} + \alpha_4 (\hat{\delta}_{\text{rot1}} + \hat{\delta}_{\text{rot2}})$

12:    $\text{arg}_{31} = \delta_{\text{rot2}} - \hat{\delta}_{\text{rot2}}; \text{arg}_{32} = \alpha_1 \hat{\delta}_{\text{rot2}} + \alpha_2 \hat{\delta}_{\text{trans}}$

13:    $\frac{\partial \text{arg}_{11}}{\partial x'} = -\frac{\partial \hat{\delta}_{\text{rot1}}}{\partial x'}; \frac{\partial \text{arg}_{11}}{\partial y'} = -\frac{\partial \hat{\delta}_{\text{rot1}}}{\partial y'}; \frac{\partial \text{arg}_{11}}{\partial \theta'} = 0$

14:    $\frac{\partial \text{arg}_{12}}{\partial x'} = \alpha_1 \frac{\partial \hat{\delta}_{\text{rot1}}}{\partial x'} + \alpha_2 \frac{\partial \hat{\delta}_{\text{trans}}}{\partial x'}; \frac{\partial \text{arg}_{12}}{\partial y'} = \alpha_1 \frac{\partial \hat{\delta}_{\text{rot1}}}{\partial y'} + \alpha_2 \frac{\partial \hat{\delta}_{\text{trans}}}{\partial y'}; \frac{\partial \text{arg}_{12}}{\partial \theta'} = 0$

15:    $\frac{\partial \text{arg}_{21}}{\partial x'} = -\frac{\partial \hat{\delta}_{\text{trans}}}{\partial x'}; \frac{\partial \text{arg}_{21}}{\partial y'} = -\frac{\partial \hat{\delta}_{\text{trans}}}{\partial y'}; \frac{\partial \text{arg}_{21}}{\partial \theta'} = 0$

16:    $\frac{\partial \text{arg}_{22}}{\partial x'} = \alpha_3 \frac{\partial \hat{\delta}_{\text{trans}}}{\partial x'} + \alpha_4 \frac{\partial \hat{\delta}_{\text{rot1}}}{\partial x'}; \frac{\partial \text{arg}_{22}}{\partial y'} = \alpha_3 \frac{\partial \hat{\delta}_{\text{trans}}}{\partial y'} + \alpha_4 \frac{\partial \hat{\delta}_{\text{rot1}}}{\partial y'}; \frac{\partial \text{arg}_{22}}{\partial \theta'} = \alpha_4$

17:    $\frac{\partial \text{arg}_{31}}{\partial x'} = 0; \frac{\partial \text{arg}_{31}}{\partial y'} = 0; \frac{\partial \text{arg}_{31}}{\partial \theta'} = -1$

18:    $\frac{\partial \text{arg}_{32}}{\partial x'} = \alpha_2 \frac{\partial \hat{\delta}_{\text{trans}}}{\partial x'}; \frac{\partial \text{arg}_{32}}{\partial y'} = \alpha_2 \frac{\partial \hat{\delta}_{\text{trans}}}{\partial y'}; \frac{\partial \text{arg}_{32}}{\partial \theta'} = \alpha_1$

19:    *for* $i = 1$ *to* 3

20:        $p_i = \mathbf{prob}(\text{arg}_{i1}, \text{arg}_{i2})$

21:        $\frac{\partial p_i}{\partial x'} = \frac{\partial \mathbf{prob}(\text{arg}_{i1}, \text{arg}_{i2})}{\partial \text{arg}_{i1}} \frac{\partial \text{arg}_{i1}}{\partial x'} + \frac{\partial \mathbf{prob}(\text{arg}_{i1}, \text{arg}_{i2})}{\partial \text{arg}_{i2}} \frac{\partial \text{arg}_{i2}}{\partial x'}$

22:        $\frac{\partial p_i}{\partial y'} = \frac{\partial \mathbf{prob}(\text{arg}_{i1}, \text{arg}_{i2})}{\partial \text{arg}_{i1}} \frac{\partial \text{arg}_{i1}}{\partial y'} + \frac{\partial \mathbf{prob}(\text{arg}_{i1}, \text{arg}_{i2})}{\partial \text{arg}_{i2}} \frac{\partial \text{arg}_{i2}}{\partial y'}$

23:        $\frac{\partial p_i}{\partial \theta'} = \frac{\partial \mathbf{prob}(\text{arg}_{i1}, \text{arg}_{i2})}{\partial \text{arg}_{i1}} \frac{\partial \text{arg}_{i1}}{\partial \theta'} + \frac{\partial \mathbf{prob}(\text{arg}_{i1}, \text{arg}_{i2})}{\partial \text{arg}_{i2}} \frac{\partial \text{arg}_{i2}}{\partial \theta'}$

24:    $p = \sum_{i=1}^{3} \log p_i$

25:    $\frac{\partial p}{\partial x'} = \sum_{i=1}^{3} \frac{1}{p_i} \frac{\partial p_i}{\partial x'}; \frac{\partial p}{\partial y'} = \sum_{i=1}^{3} \frac{1}{p_i} \frac{\partial p_i}{\partial y'}; \frac{\partial p}{\partial \theta'} = \sum_{i=1}^{3} \frac{1}{p_i} \frac{\partial p_i}{\partial \theta'}$

26:    *return* $\langle \frac{\partial p}{\partial x'}, \frac{\partial p}{\partial y'}, \frac{\partial p}{\partial \theta'} \rangle$

Table 14.3 Algorithm for computing the first derivative of the logarithm of the motion model $\log P(s' \mid a, s)$ based on odometry information (see Table 5.5 on page 108).

1:        **Algorithm first_derivative_prob_normal_distribution**(d, b)**:**

2:        $\text{prob} = \frac{1}{\sqrt{2\pi b}} \, e^{-\frac{1}{2} \frac{d^2}{b}}$

3:        $\frac{\partial \text{prob}}{\partial d} = -\text{prob} \frac{d}{b}$

4:        $\frac{\partial \text{prob}}{\partial b} = \left(\frac{d^2}{2b^2} - \pi (2\pi b)^{-\frac{3}{2}} \right) \text{prob}$

5:        *return* $\langle \frac{\partial \text{prob}}{\partial d}, \frac{\partial \text{prob}}{\partial b} \rangle$

Table 14.4 Algorithm for computing the first derivative of the zero-centered normal distribution with variance b (see Table 5.2 on page 97).

for the perceptual model discussed in Chapter **??** (see Table 6.3 on page 143). The calculation of the gradient is fairly straightforward, and we arranged the algorithm in Table 14.2 in a way that should make it easy to understand the mechanics of calculating the desired derivative. Similarly, Table 14.3 depicts an algorithm for calculating the first derivative of the logarithm of the odometry-based motion model,

$$\nabla_{\bar{s}^{(t)}} \, \log P(\bar{s}^{(t)} \mid a^{(t-1)}, s^{(t-1)}) \tag{14.19}$$

which was originally discussed in Chapter **??** (see Table 14.3 on page 402). This algorithm requires the first derivative of the zero-centered noise model **prob**(), which is provided in Table 14.4 (zero-mean normal distribution). While this model is not exactly the one required in (14.16)—it does not take the model m into account—it is close (the extension is trivial). It demonstrates one more the mechanics of calculating a derivative of a complex function using the chain rule of differentiation. When implementing fast gradient descent, we strongly recommend to calculate the gradients by hand, instead of blindly copying the algorithms listed here.

14.3.3   Suggestions for the Implementation

Our algorithm for scan alignment is asymmetric. It only relies on measurements in o that fall into the free-space of the map m. What it not uses is the converse: if the map shows an occupied region in an area that, according to the scan s, should be free—this "mismatch" is not used for adjustment. Clearly, this is a deficiency of the basic perceptual model. If possible, we recommend to implement a symmetric version of the model, which penalizes both: measurements in s that fall into m's free-space, and regions in m that fall into s's free-space. The resulting perceptual model is slightly

more complex. Calculating the gradient with respect to the pose s' is analogous to the algorithm given here.

Additionally, the gradient calculation can be sped up significantly by caching specific information. If implemented as described above, we found that only approximately ten gradient calculations could be carried out per second, using 500Mhz Pentium II Computers running Linux. By far most of the time is consumed calculating the perceptual model and its first derivative, which spend most of the time with a single operation: The search for the nearest occupied location, which is performed in line 9 in the gradient algorithm shown in Table 14.2 (and also line 9 in the algorithm **range_finder_model2** shown in Table 6.3 on page 143). This search is expensive; furthermore, when changing the estimated pose $s^{(t)}$, the search has to be repeated.

Alternatively, one can first calculate the location of the nearest occupied location for a fine-grained grid, overlayed the x-y-space. Calculating this grid takes some start-up time, however, with it the search reduces to a table-lookup operation. The result of the search is then only approximate, as not all locations within a grid cell necessarily map to the same nearest neighbor. Using this technique with grid resolutions between 10 and 20 cm, we found that the resulting gradients were still well-suited for hill-climbing in likelihood space. Moreover, we found that the calculation of the gradient could be sped up by two orders of magnitude, making it possible to perform hundreds of iterations per second.

14.3.4   Examples

Figure 14.1 shows two example sequences (arranged vertically) of applying gradient descent for scan alignment. In both examples, the likelihood is maximized using the function **first_derivative_log_range_finder_model2** in Table 14.2. Both examples were recorded by a moving robot. We manually perturbed the second scan relative to the first by 10 cm along each axis, and 30 degrees rotational error.

Despite this large error, the routine reliably converges to the "right" alignment in approximately 100 iterations. Shown in Figure 14.1 are the initial alignments, and the alignments after 10, 50, and 100 iterations of gradient descent. These examples suggest that the likelihood function possesses few, if any, local minima in the proximity of the global optimum. In practice, we found that for scans recorded in sequence (e.g., with 3 Hertz), the gradient descent search routine practically *always* converges to the right maximum and aligns the scans with high accuracy.
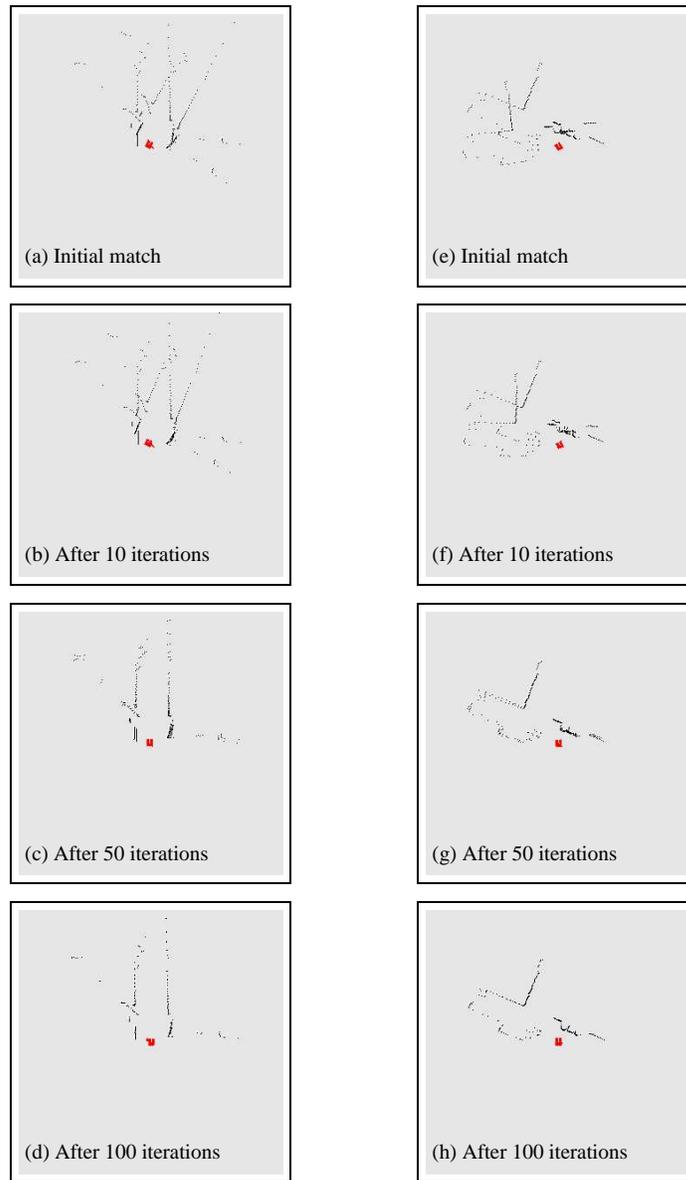
Figure 14.1 To examples of gradient descent for aligning scans (arranged vertically). In both cases, the initial translational error is 10 cm along each axis, and the rotational error is 30 degrees. The gradient descent algorithm safely recovers the maximum likelihood alignment.

Figure 14.2   A typical map obtained using the most simple incremental maximum likelihood approach. While the map is locally consistent, the approach fails to close the cycle and leads to inconsistent maps. This elucidates the limitations of stepwise likelihood maximization, and illustrates the difficulty of mapping cyclic environments.

14.3.5   Limitations

The basic limitation of stepwise likelihood maximization is that it can fail to generate globally optimal solutions. The problem is typically not observed when mapping a single hallway, or a simple non-cyclic environments—since here local consistency between adjacent measurements suffices to build a globally consistent map. However, in cyclic environments the robot has to establish correspondence to previously gathered data with potentially unbounded odometric error, and has to revise pose estimates backwards in time. Consequently, local consistency as achieved by the incremental maximum likelihood method is insufficient to achieve global consistency.

Figure 14.2 illustrates this problem using an example. It shows a map, acquired in a cyclic environment. The robot traversed a sequence of corridors, finally reaching its initial location. While along the way the maps appear locally consistent, it nevertheless accrues some localization error. As the robot closes the cycle, the accumulated localization error is too large to be accommodated locally, and the inability to correct maps backwards in time leads to a permanent mismatch shown in that figure. The resulting map happens to be still good enough for navigation. It is not difficult to imag-

ine, though, that larger cycles will lead to bigger mismatches that make it impossible to navigate.

In summary, the incremental maximum likelihood approach suffers two main limitations:

1. It is unable to cope with large odometry error.

2. It is unable to correct poses backwards in time.

Next we will describe an extension that addresses these two shortcomings, leading to an algorithm that has been found to work reliably in environments with cycles.

14.4　INCREMENTAL MAPPING WITH POSTERIOR ESTIMATION

To overcome the cycle mapping problem, it is necessary to revise past pose estimates backwards in time. This has implications for the way data is memorized. If, for example, we only maintained a single occupancy grid at each point in time, it would be impossible to modify the grid in response to an inconsistency that was discovered when closing a cycle. Thus, it is necessary to memorize past sensor measurements along with their estimated pose, so that pose estimates can be corrected retrospectively.

14.4.1　Detecting Cycles

Next, we need a mechanism for detecting cycles. This is realized by introducing a second estimator, which performs full posterior estimation over the robot's poses, concurrently to the map estimation. Posterior pose estimation is the same as localization. Robot pose estimators were already discussed extensively in the chapters on state estimation (Chapter 2) and localization (Chapter 7) . Let us briefly restate the basic filter equation:

$$Bel(x^{(t)}) \;\; = \;\; P(o^{(t)}|x^{(t)}, m^{(t)}) \int P(x^{(t)}|a^{(t-1)}, x^{(t-1)}, m^{(t-1)}) \, Bel(x^{(t-1)}) \, dx^{(t-1)} \quad (14.20)$$

Where $x^{(t)}$ is the pose of the robot at time t. An algorithm for posterior estimation over poses was given in Table 7.1 on page 163.

When implementing the localization algorithm, one has to take into account that it is run concurrently with an incremental map estimator, which already estimates robot poses. At the very least, it must be consistent with the maximum likelihood estimator in non-cyclic situations. Thus, the term $P(x^{(t)}|a^{(t-1)}, x^{(t-1)}, m)$ is not a model of robot motion. Instead, it is a model of the residual error induced by the maximum likelihood estimator. To be consistent, the mode of $P(x^{(t)}|a^{(t-1)}, x^{(t-1)}, m)$,

$$\underset{x^{(t)}}{\operatorname{argmax}} P(x^{(t)}|a^{(t-1)}, x^{(t-1)}, m) \tag{14.21}$$

must equivalent to the result of the maximum likelihood pose estimator (14.13). The uncertainty in the conditional reflects the posterior uncertainty of this estimate. Assuming that the uncertainty it normally distributed with zero mean, its inverse covariance matrix can be recovered from the second derivative of the log likelihood function in (14.13) (notice that the second derivative of the log of a multivariate normal distribution is always the inverse of its covariance). Similarly, one can use any of the motion models described in this book and "guess" the motion noise parameters by trial-and-error. In either case, $P(x^{(t)}|a^{(t-1)}, x^{(t-1)}, m)$ should be a much more focussed probability distribution than the original robot motion model, since it models the uncertainty *after* correcting the pose based on a sensor measurement.

Similarly, the term $P(o^{(t)}|x^{(t)}, m^{(t)})$ is not the perceptual probability. If one were to use the perceptual probability, the information from the most recent sensor scan would be used twice—once in the maximum likelihood estimator and once in the posterior estimate—and the robot could become falsely over-confident of its pose. Instead, we suggest to evaluate $P(o^{(t)}|x^{(t)}, m^{(t)})$ only with regards to sensor measurements that create conflicts when closing a cycle. Such past measurements can easily be identified, by keeping track of their time and their maximum likelihood location. As a result, the posterior $Bel(x^{(t)})$ grows as the robot moves into unexplored terrain. When reconnecting with a previously mapped area, its uncertainty will decrease as a result of factoring in the perceptual probability $P(o^{(t)}|x^{(t)}, m^{(t)})$.

14.4.2   Correcting Poses Backwards in Time

For the purpose of mapping, the robot's pose estimate is now replaced with the mode of the posterior $Bel(x^{(t)})$. In other words, instead of using (14.13) as the pose estimate, it is merely used to define the posterior $P(x^{(t)}|a^{(t-1)}, x^{(t-1)}, m)$. When the robot moves into unexplored terrain, $\operatorname{argmax}_{x^{(t)}} Bel(x^{(t)})$ will be identical to the estimate of the maximum likelihood estimator. However, when closing a cycle, both estimates may differ.

1:   **Algorithm incremental_ML_mapping_for_cycles(**$o, a, s, m, Bel(s)$**):**

2:   $\langle m', s' \rangle = $ **incremental_ML_mapping**(o, a, s, m)

3:   $Bel(s') = P(o, s') \int P(s'|a, s) \; Bel(s) \; ds$

4:   $s'' = argmax_{s'} \, Bel(s')$

5:   *if* $s'' \neq s'$

6:        *linearly distribute* $s'' - s'$ *along cycle*

7:        *refine past poses with algorithm* **incremental_ML_mapping**

8:   *return* $\langle m', s'', Bel(s') \rangle$

Table 14.5   Extension of the algorithm **incremental_ML_mapping** for mapping cyclic environments. The algorithm carries a posterior estimate over poses, along with the step-wise maximum likelihood map. When a cycle is closed, the maximum likelihood estimate s' and the mode of the posterior s'' will differ, initiating the backwards adjustment of robot poses.

To maintain consistent maps, these differences have to be propagated backwards in time, along the cycle. This is again a nested maximum likelihood estimation problem. However, care has to be applied with regards to computational efficiency. One way to implement the backwards correction efficiently is the following two-phase algorithm:

1. The deviation between the stepwise maximum likelihood estimate and the mode of the posterior is distributed *linearly* among all poses along the cycle. Calculating linear correction terms is extremely fast and places the robot poses somewhere close to their desired corrected estimates. However, the optimization problem is highly non-linear.

2. Subsequently, the incremental maximum likelihood estimator is applied backwards in time to maximize the probability of poses given their neighbors (in time). This algorithm is very similar to the one above, hence will not be described here in detail.

Table 14.5 sketches the resulting algorithm, which simultaneously maintains a maximum likelihood map and a full posterior estimate over poses. When a cycle is closed, the maximum likelihood pose estimate and the mode of the posterior will deviate,

which is detected in Step 5 of the algorithm. Steps 6 and 7 correspond to the mechanism for refining poses backwards in time.

14.4.3   Illustrations

Figure 14.3 shows an example of applying this algorithm in practice, generated from the same data that was used to generate the map shown in Figure 14.2. Here the posterior estimate is implemented using Monte Carlo localization (MCL). In the beginning, as the robot maps unknown terrain, the sample set spreads out, though it does not spread quite as much as a raw robot motion model would suggest. This is shown in Figure 14.3a, which shows the posterior samples along with the map obtained by the incremental maximum likelihood estimates.

Figure 14.3b shows the situation just before closing the cycles. As before, the robot has accrued significant pose estimation error in its maximum likelihood estimate. However, this error appears to be well captured in the posterior estimate of the pose, represented by the particle set. Figure 14.3c depicts the situation after closing the cycle. The posterior estimate is now very focussed, since it incorporates perceptual measurements relative to earlier parts of the map.

Despite its reliance on maximum likelihood estimates, the algorithm **incremental_ML_mapping_for_cycles** is very robust. Figure 14.4 shows the result of an extreme experiment, where this algorithm was applied to the mapping problem in the absence of any odometry (action) data. Consequently, the raw data is extremely different to interpret. Figure 14.4a shows the data set in the absence of odometry information. Clearly, overlaying scans produces an image that is impossible to interpret for the human eye. Our algorithm, applied to this data set, nevertheless produces a reasonably accurate map, as shown in Figure 14.4b. The only modification to the basic algorithm is the omission of the gradient $\nabla_{s'} \log P(s'|a, s)$, which is simply assumed to be zero due to the lack of action items a. The error accrued during mapping is approximately twice as large as with odometry information; however, as the cycle is closed, the residual error along the loop shrinks significantly and the resulting map is of the same quality as if odometry information were available. However, these results have to be taken with a grain of salt. Our approach will clearly fail in long, featureless corridors which lack the necessary structure for estimating poses from laser range data alone. All these results demonstrate, thus, is the robustness of the basic routine.

All the maps above have been obtained in real time on a 300Mhz Pentium II Laptop PC, with a robot moving at approximately 60 cm/sec (see Figure 14.5). However, we should notice that the current approach is unable to handle nested loops, and if the
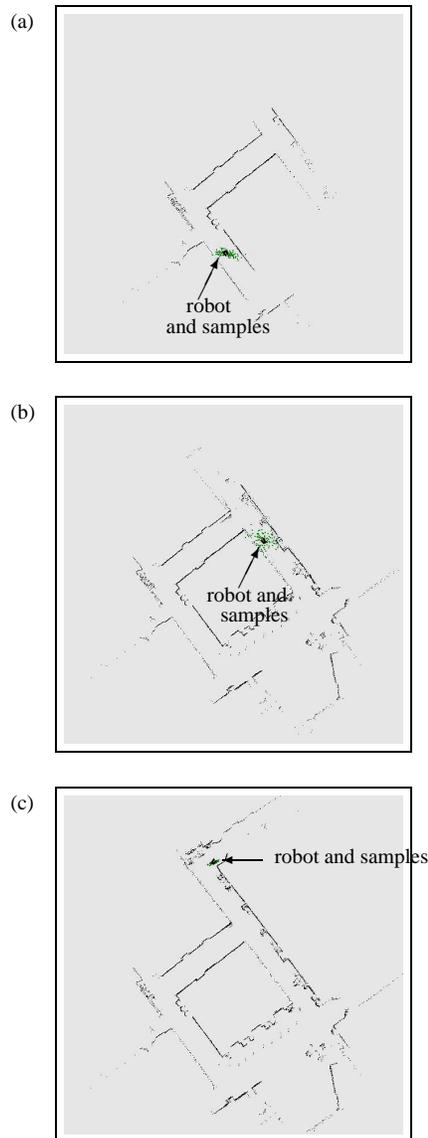
Figure 14.3 Mapping cyclic environments using particle filters for posterior pose estimation. The dots, centered around the robot, indicate the posterior belief which grows over time (a and b). When the cycle is closed as in (c), the map is revised accordingly and the posterior becomes small again.

Figure 14.4   Mapping without odometry. Left: Raw data, right: map, generated on-line in real-time.

loop is very large, it may be impossible to correct the pose estimates along the cycle in real time.

14.5   MULTI-ROBOT MAPPING

Estimating a full posterior over poses has a second advantage: It makes it possible to localize a robot in the map built by another. This is an essential step towards multi-robot robot mapping, which requires that robots determine their poses relative to each other. If one robot is known to be started in the map built of another, this estimation is equivalent to global localization, which can be carried out using MCL.

The basic algorithm for cooperative mobile robot mapping is very similar to **incremental_ML_mapping_for_cycles** stated in Table 14.5, with two modifications: First, the initial belief $Bel(s^{(0)})$ for the second robot is initialized by a uniform distribution over poses, instead of a point-mass distribution. Second, map measurements are only integrated when the robot feels confident enough that it knows its pose relative to the first. This is determined using the entropy of the posterior $Bel(s^{(t)})$: If the entropy $H[Bel(s^{(t)})]$ falls below a certain threshold, localization is assumed to be completed and the second robot integrates its sensor measurements into the map of the first one.

Figure 14.6 illustrates how a second robot localizes itself in the map built by another robot (called there: the *team leader*). The robots used here are Pioneer robots equipped with laser range finders, such as the ones shown in Figure 14.5. Figure 14.6a shows the belief after incorporating one sensor scan, represented by particles scattered almost uniformly through the map built by the team leader. A few time steps later, all

Figure 14.5 RWI Pioneer robots used for multi-robot mapping. These robots are equipped with a SICK laser range finder.

surviving particles are focused on the same pose. Now the robot knows with high probability its pose relative to the other robot's map.

We finally briefly outline an architecture for fusing maps from many robots, operating distributedly. If maps from many robots shall be integrated, one can build a hierarchical architecture. At the bottom of the hierarchy, each robot builds its own maps, estimating poses within its local coordinate frame. The corrected poses and measurements are than communicated to a higher level program, which uses the identical mapping approach, to develop a single map by localizing each robot relative to its coordinate frame. The integration then continues recursively up the hierarchy, until the final module integrates the data of its submodules. The resulting map is then communicated back (at high frequency) to all modules, including the robots, to aid the process of localization.

Such an architecture has been implemented and led to the results shown in Figure 14.7. This map was acquired in real-time using three robots, which locally corrected their pose estimates. A higher level module collected these pre-corrected data, and used the maximum likelihood estimator described above to build a single map for the robots. This map has been constructed in real-time, while the robot were in motion. This is necessary for robot that seek to systematically explore an unknown environment. More detail will be given in Chapter **??**, where we talk about probabilistic strategies for robot exploration.
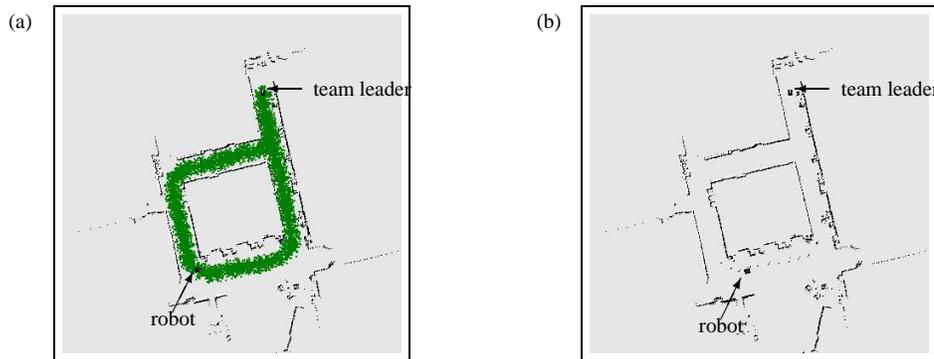
Figure 14.6 A second robot localizes itself in the map of the first, then contributes to building a single unified map. In (a), the initial uncertainty of the relative pose is expressed by a uniform sample in the existing map. The robot on the left found its pose in (b), and then maintains a sense of location in (c).
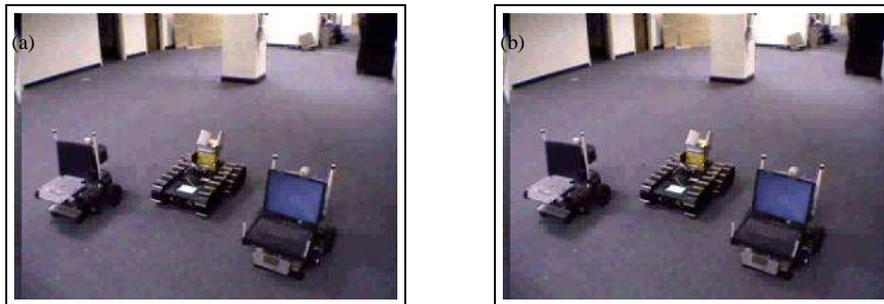
Figure 14.7 (a) Team of 3 robots, which produced the map shown in (b) in real-time. Also shown in (b) are the paths of the robots.

14.6   MAPPING IN 3D

Finally, we will be interested in developing full three-dimensional maps if building interiors. Such maps offer three key advantages over the two-dimensional maps studied throughout most of this book.

■   First, 3D maps are easier to acquire than 2D maps, at least from a perceptual point of view. By looking at the full 3D structure of a building interior, the problem of estimating correspondence is simpler. This is because places that might look alike in 2D often do not in 3D, reducing the danger of confusing them with each other.
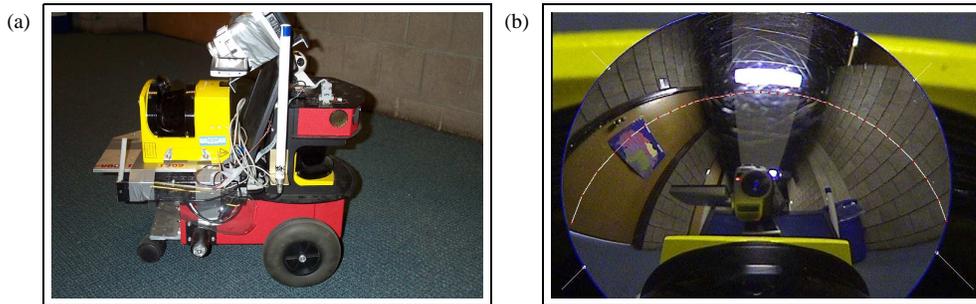
Figure 14.8 (a) Pioneer robot equipped with 2 laser range finders used for 3D mapping. (b) Panoramic image acquired by the robot. Marked here is the region in the image that corresponds to the vertical slice measured by the laser range finder.

- Second, 3D maps facilitate navigation. Many robot environments possess significant variation in occupancy in the vertical dimension. Modeling this can greatly reduce the danger of colliding with an obstacle.

- Third, many robot tasks require three-dimensional information, such as many tasks involving the localization and retrieval of objects or people.

- Finally, 3D maps carry much more information for a potential user of the maps. If one builds a map only for the sake of robot navigation, then our argument does not apply. However, if map are being acquired for use by a person (e.g., a rescue worker entering a building after an earthquake), 3D information can be absolutely critical.

Given these obvious advantages of 3D over 2D maps, it is surprising that the robotics community has paid so little attention to mapping in 3D.

The estimation of 3D maps is a challenging computational problem, which is amenable to many of the techniques described in this book. However, even with powerful 2D mapping one can obtain reasonable 3D maps. In the remainder of this chapter, we will briefly show results of a routine for acquiring 3D maps. At its center is a Pioneer robot equipped with two laser range finder and a panoramic camera. The robot is shown in Figure 14.8a. One of the laser range finders is directed forward, emitting a horizontal plateau of laser light. This laser range finder is used for concurrent mapping and localization using the algorithm described in this chapter. The second is pointed upwards, orthogonal to the robot's motion direction. This this laser the robot can scan the walls and the ceiling, gathering critical range information for building 3D maps. Finally, the panoramic camera is mounted adjacent to the upward laser scanner, en-
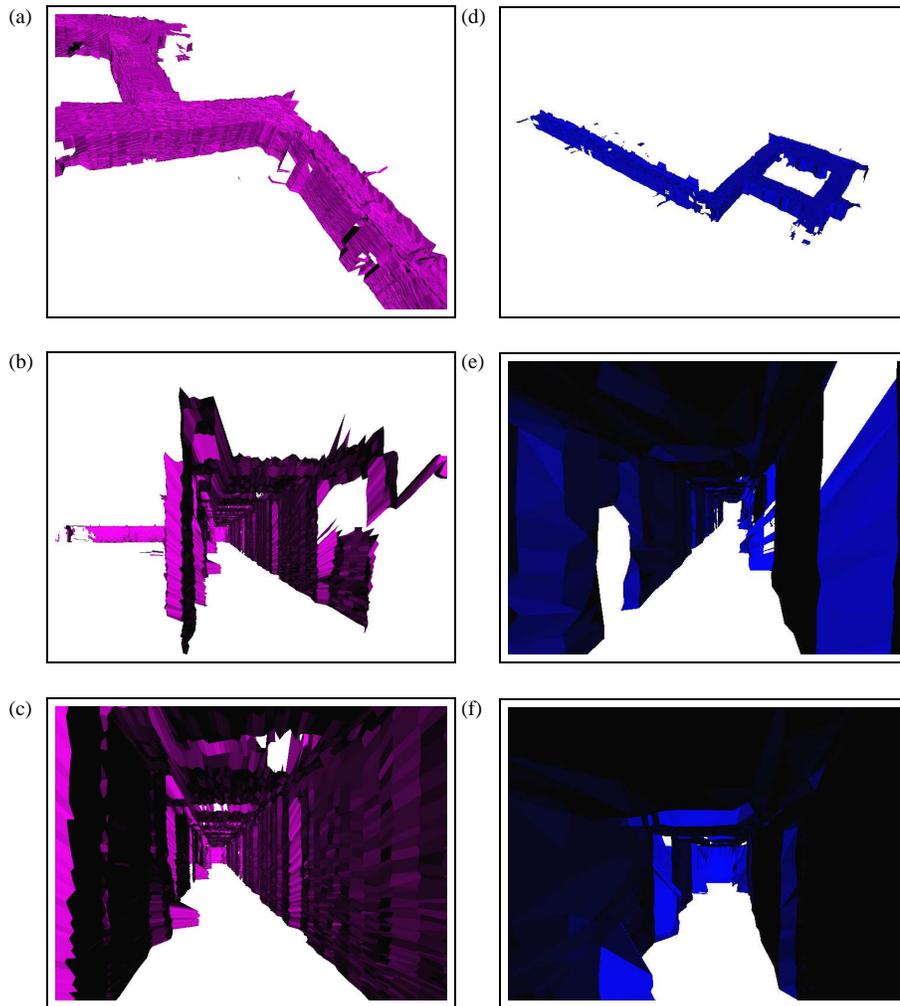
Figure 14.9 Views of the 3D structural map, for the high-res model (left column) and the reduced resolution model (right column).

abling to acquire color information that are easily matched to the corresponding range measurements. Figure 14.8b shows an example of a panoramic image. The line in the image corresponds to the range data acquired by the upwards-pointed laser.

Figure 14.9 shows images of a 3D structural map acquired in a multi-corridor indoor environment, after traversing it once. Localization is provided by the algorithm **in-**
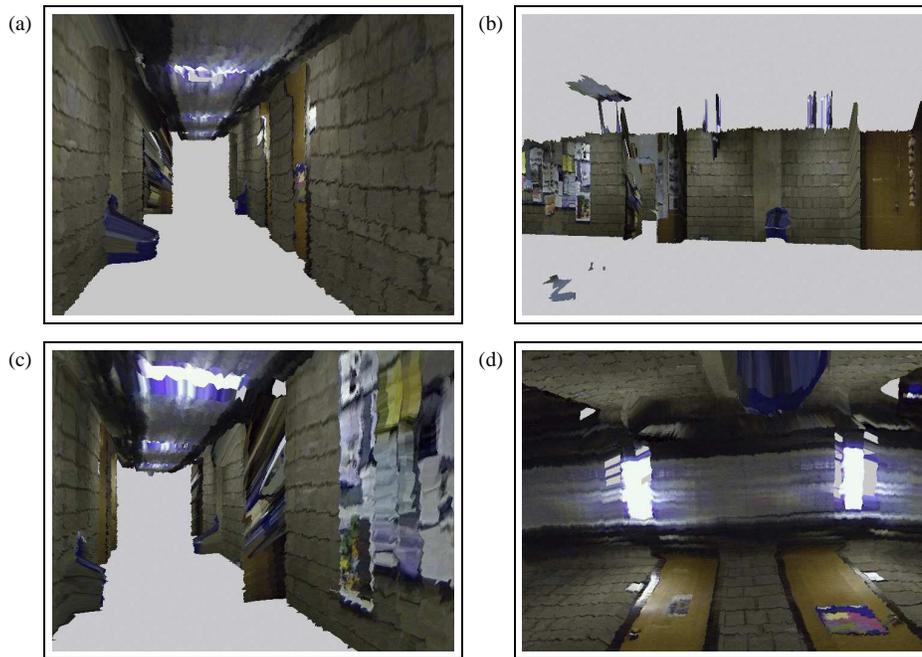
Figure 14.10 Snapshots of 3D texture maps, acquired using the modified Pioneer robot. These views are generated using a VRML viewer. Some of these views are similar to those one would see when entering the building. Others show the building from perspectives that are not available in the physical world.

cremental_ML_mapping_for_cycles. The vertical range measurements are then converted into polygones without additional pose estimation. The resulting map can be displayed in a VRML viewers, enabeling users to "fly through the building" virtually, without actually entering it. It also enables users to assume viewpoints that would be impossible in reality, such as the ones shown in Figure 14.9a and d.

A limitation of this map is its very large number of polygones, which increases linearly with robot operation. Simplifying polygonial maps has long been studied in the computer graphics literature, which concerns itself (among other things) with methods for fast rendering of complex polygonal maps. While the left column in Figure 14.9 shows views from the high-complexity map, the right column stems from a map which possesses only 5% of the polygones of the left map. The reduction of the polygonal map is achieved thorugh a routine adopted from the computer graphics literature [10].

The reduced map can be rendered in real-time without much of a noticeable loss of accuracy.

Finally, Figure 14.10 shows images obtained from a full 3D texture map. This map has been obtained by projecting the color information gathered by the panoramic camera onto the structural model. This map can equally be rendered in VRML. Again, the texture information is not used for localization. Instead, the location estimates of the structural components and the texture is directly obtained from the 2D map, exploiting knowledge of the location of the upward pointed laser and the panoramic camera relative to the robot's local coordinate system.

14.7   SUMMARY

This section discussed a collection of algorithms that blend together maximum likelihood estimation and posterior estimation over poses. In particular:

- We introduces a fast maximum likelihood algorithm for incrementally growing maps.

- We showed how to implement this algorithm using gradient descent, providing example programs for calculating derivatives in log likelihood space for a specific perceptual and motion model.

- We discussed the strengths and shortcomings of the approach. In particular, we showed that the incremental approach fails to find good maps in cyclic environments.

- We then introduced an extension which integrates a full posterior estimator over robot poses. We showed how to use these posterior estimates for resolving inconsistencies in cyclic environments.

- We illustrated, through examples, the robustness of the algorithm, while pointing out that at the same time, the algorithm is incapable of accommodating nested cycles.

- We discussed extensions to multi-robot mapping, pointing out that the posterior estimation enables one robot to localize itself in the map of another.

- Finally, we showed examples of three-dimensional maps generated using a robot with an upward pointed laser range finder and a panoramic camera.

14.8   BIBLIGRAPHICAL REMARKS

Shall we show images of Gutmann/Konolige? I'd opt for yes.

14.9   PROJECTS

1. Develop a 3D mapping algorithm that exploits the full 3D structure for pose and map estimation during mapping.

2. The multi-robot algorithm described in this chapter requires that each robot starts in the map built by another robot. Develop a multi-robot mapping algorithm that relaxes this assumption, that is, where robots can start at arbitrary poses and might only later traverse the same terrain.

3. Develop a 3D mapping algorithm that represent the enviroment by typical building components, such as walls, doors, tables, chairs, instead of sets of polygones.

15

MARKOV DEVISION PROCESSES

15.1 MOTIVATION

Thus far, the book has focused on robot perception. We have discussed a range of probabilistic algorithms that estimate quantities of interest from sensor data. However, the ultimate goal or any robot software is to choose the right actions. Accurate state estimation is only desirable insofar it facilitates the choice of action. This and the following chapter will therefore discuss probabilistic algorithms for action selection.

To motivate the study of probabilistic action selection algorithms, consider the following examples.

1. A a robotic manipulator grasps and assembles parts arriving in random configuration on a conveyer 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? Are there manipulation strategies that result in a well-defined configuration without sensing?

2. An underwater vehicle shall travel across the North Pole. Shall it take the shortest route, running risk of loosing orientation under the Ice? Or sould it avoid large featureless areas at the risk of using more time to reach its goal.

3. An robotic helicopter flies autonomously in a wooded area. The helicopter should stay clear of obstacles such as the ground so that even an unexpected wind gust can make it crash. But how far is far enough?

4. A team of robots explore an unknown planet. The problem is particularly hard if the robots do not know their initial pose relative to each other. 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? And what are the optimal ways to explore a planet with teams of robots?

These examples illustrate that action selection in many robotics tasks is closely tight to the notion of uncertainty. In some tasks, such as robot exploration, reducing uncertainty is the direct goal of action selection. Such tasks are known as *information gathering tasks*. Information gathering tasks will be studied in the next chapter. In other cases, a reducing uncertainty is merely a means to achieving some other goal, such as reliably arriving at a target location. Such tasks will be studied here.

From an algorithm design perspective, is convenient to distinguish two types of uncertainty: uncertainty in action, and uncertainty in perception.

1. **Deterministic versus stochastic action effects.** Classical robotics often assumes 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.

2. **Fully observable versus partially observable systems.** Classical robotics often assumes that sensors can measure the full state of the environment. As argued repeatedly throughout this book, this is an unrealistic assumption. The lack of perfect sensors has two ramifications: First robot control must be robust with respect to *current* uncertainty. Second, it must cope with future, *anticipated* uncertainty, and choose actions accordingly. An example of the latter was given above, where we discussed a robot which has the choice between a shorter path through a featureless area, and a longer one that reduces the danger of getting lost.

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 differ 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 a off-line optimization (planning) phase, in which they calculate a control policy. During execution, the control policy can be invoked very efficiently, and it can cope with a range of different situations. By no means is the choice of algorithms meant to suggest that this

is the only way to cope with uncertainty in robotics. Instead, it reflects the style of algorithms that are currently in use.

The majority of algorithms discussed in this chapter assume finite state and action spaces. Continuous spaces are approximated using grid-style representations. The chapter is organized as follows.

- Section 15.2 discusses in depth the role of the two types of uncertainty and lays out their implications on algorithm design.

- Section 15.3 introduces value iteration, a popular planning algorithm. Value iteration, as introduced there, addresses 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).

- Section 16.2 discusses a more general planning algorithm that addresses both types of uncertainty: Uncertainty in action and uncertainty in perception. This algorithm adopts the idea of value iteration, but applies it to a belief space representation. The framework underlying this algorithm is called Partially Observable Markov Decision Processes (POMDPs). The POMDP algorithm can anticipate uncertainty, actively gather information, and explore optimally in pursuit of an arbitrary performance goal, at the expense of increased computational complexity.

- Finally, Section 16.5 discusses augmented MDP algorithms, which are crossovers between MDP and POMDP algorithms. Augmented MDP algorithms consider uncertainty, but abstract away detail and hence are computationally much more efficient than POMDP solutions.

In the following chapter, ... we will discuss a more general algorithms that addresses both types of uncertainty: Uncertainty in action and uncertainty in perception. The framework underlying these algorithm is called Partially Observable Markov Decision Processes (POMDPs). The POMDP algorithm can anticipate uncertainty, actively gather information, and explore optimally in pursuit of an arbitrary performance goal, at the expense of increased computational complexity.

```
+----------------------------------------------+
|                                              |
|   +---+   +----------+   +----------+   +---+  |
|   |   |   |          |   |          |   |   | +-+
+---+   |Goal----------+Rob+----------+Pit|   | | |
|   |   |   +----------+   +----------+   |   +-+ |
|   |   |   |          |   |          |   |   |   |
|   +---+   +----------+   +----------+   +---+   |
|                                              |
+----------------------------------------------+
```
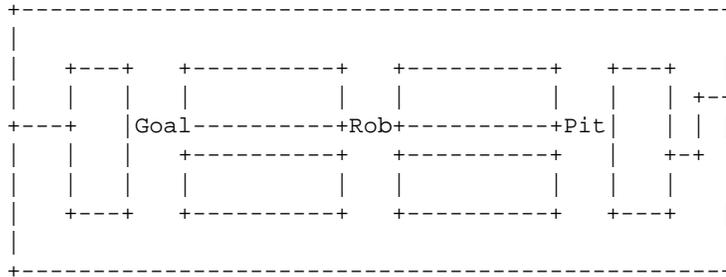
Figure 15.1   Near-symmetric environment with narrow and wide corridors. The robot's initial location is known, but not its pose.

15.2   UNCERTAINTY IN ACTION SELECTION

Make Figure: Classical Planning

Make Figure: MDP planning

Make Figure: POMDP planning

Figure 15.1 shows a toy-like environment that illustrates the different types of uncertainty. Shown there is a mobile robot in a corridor-like environment. The environment is highly symmetric; the only distinguishing feature are two walls at its far ends. At symmetric locations, the environment possesses a small number distinct places, one labeled the goal location (G), another which contains a pit (P), and one that contains a unique landmark that can help the robot finding out where it is. The robot's task is to advance to the goal location as quickly as possible while avoiding falling into the pit. Let us suppose the goal and the pit are perceptually indistinguishable—unless the robot actually enters the region, running risk to fall into the pit. Finally, we notice that there are multiple paths to either the goal, one that is short but narrow, and two others that are longer but wider. Clearly, this environment is less complex than natural robotic environments; however, it is complex enough to distinguish different types of uncertainty.

In the classical robot planning paradigm, there is no uncertainty. The robot knows its initial pose and it knows the location of the goal. Furthermore, actions are executed in the physical world as commanded. In such a situation, it suffices to plan off-line

a single sequence of actions, which is then executed at run-time. There is no need to sense. Figure **??** shows an example of such a plan. Obviously, in the absence of errors in the robot actuator, the narrow shorter path is superior to the longer, wider one. Hence, a planner that strives for optimality would choose the former path over the latter. Many existing planning algorithm are only concerned with finding *a* path, regardless of optimality. Thus, they might choose either path. What happens if the plan is executed? Of course, physical robot hardware is inaccurate. A robot blindly following the narrow hallway might run danger of colliding with the walls. Furthermore, a blindly executing robot might miss the goal location because to 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 its alternative.

A paradigm that encompasses uncertainty in robot motion is known as *Markov decision processes*, or *MDPs*. MDPs assume that the state of the environment can be fully measured at all times. In other words, the perceptual model $P(o|s)$ is deterministic and bijective. However, is allows for stochastic action effects, that is, the action model $P(s'|s, a)$ 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, either because of its actions, or because of other environment dynamics. One way to cope with the resulting uncertainty is to generate a *policy* for action selection defined for all states that the robot might encounter. Such mappings from states to actions are also known as *universal plans* or *navigation functions*. An example of a policy is shown in Figure **??**. 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. Additionally, this framework opens the opportunity to guide the action selection process based on future, anticipated uncertainty. Consider, for example, a highly inaccurate robot which, if placed in the narrow corridor, is likely to collide with a wall. Planning algorithms that consider motion uncertainty can assess this risk at planning time and might choose the wider, safer path.

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 o are noisy projections of the state s. Hence, the state can only be estimated up to a certain degree. To illustrate this, consider the situation depicted in Figure **??** under the assumption that the robot does not know its initial orientation. Clearly, the symmetry of the environment makes it difficult to disambiguate the robot's pose.

By moving directly towards the projected goal state the robot faces a 50% chance of falling into the pit—which would tell it where it is but result in mission failure. The optimal plan, thus, is to move to the small area in the upper right (or lower left), which enables the robot to disambiguate its pose. After that, the robot can then safely move to its goal location. Thus, the robot has to actively gather information while suffering a detour. This is an example of the most interesting scenario in probabilistic robotics: The robot's sensors pose intrinsic limitations as to what the robot knows. Similar situations occur in locate-and-retrieve tasks, planetary exploration, and many other robot tasks.

How can one devise an algorithm for action selection that can cope with this type of uncertainty? 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 partially observable environment cannot be solved by considering all possible environments and averaging the solution.

Instead, the key idea is to generate plans in *belief space* (sometimes referred to *information space*). The belief space comprises the space of all belief distributions b that the robot might encounter. The belief space for our example is shown in Figure **??**. It reflects what the robot knows about the state of the world. The center diagram corresponds to the case where the robot is ignorant of its heading direction, as indicated by the two question marks. As the robot enters any of the locations that reveal where there goal is, it will make a transition to one of the two diagrams at the border. Both of those correspond to cases where the robot pose is fully known: The robot faces north in the left diagram, and it faces south in the right one. Since the a priori chance of each location is the same, the robot will experience a random transition with a 50% chance of ending up in either state of knowledge.

The belief state is rich enough to solve the planning problem. In our toy example, the number of different belief states happens to be finite. This is usually not the case. In worlds with finitely many states the belief space is usually continuous, but of finite dimensionality. If the state space is continuous, the belief space is usually infinitely-dimensional.

This example illustrates a fundamental property that arises from the robot's inability to sense the state of the world—one whose importance for robotics has often been under-appreciated. In particular, in uncertain worlds a robot planning algorithm must consider the state of knowledge (the belief state). 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 optimally gathers information, in that it only seeks new information to the extent that it is actually beneficial to the expected utility of the robot's action. This is a key advantage of the probabilistic approach to robotics. However, it comes at the price of an increased complexity of the planning problem.

15.3 VALUE ITERATION

We will now describe a first algorithm for action selection under uncertainty. The algorithm is a version of various flooding-type algorithms that recursively calculate the utility of each action relative to a payoff function. Value iteration, as discussed in this section, addresses only the first type of uncertainty: It devises control policies that can cope with the stochasticity of the physical world. It does not address the uncertainty arising from perceptual limitations, Instead, we will assume that the state of the world is fully observable at any point in time.

15.3.1 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 been picked up by a robot arm), or they might express conditions over longer periods of time (e.g., a robot balances a pole). Some action selection algorithms carry explicit notions of goals; others use follow implicit goals. The algorithms discussed in this book all use explicit descriptions of goals. This enables them to pursue different goals. In contrast, control algorithm with implicit goals are usually unable to generate actions for more than just one goal.

In robotics, one is often concerned with reaching specific goal configurations, 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, jerk, or the number of collisions with surrounding obstacles. 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 (also known as *utility*,

cost). The payoff, denoted c, is a function of the state. For example, a simple payoff function is the following:

$$c(s) \quad = \quad \begin{cases} +100 & \text{if robot reaches goal} \\ -1 & \text{otherwise} \end{cases} \qquad (15.1)$$

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 is has not reached that configuration.

Why using 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 our treatment of both quantities will be entirely analogous 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 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 programs are usually referred to as control policies, denoted π:

$$\pi : d^{(0...t)} \longrightarrow a^{(t)} \qquad (15.2)$$

A policy π is a function that maps date into actions. Taking as general a view as possible, a policy might be an elaborate planning algorithm, or it might be a fast, reactive algorithm that bases its decision on the most recent data item only. The policy π may deterministic or non-deterministic, and it might only be partially defined in the space of all data sets $d^{(0...t)}$.

An interesting concept in the context of creating control policies is the *planning horizon*. Sometimes, it suffices to choose an 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 action 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

$$C^T = E \left[\sum_{\tau=1}^{T} \gamma^\tau c_{t+\tau} \right] \tag{15.3}$$

Here the expectation $E[\,]$ is taken over future momentary payoff values $c_{t+\tau}$ that the robot might accrue between time t and time $t + T$. The individual payoffs $c_{t+\tau}$ are multiplied by an exponential factor γ^τ, called the *discount factor*, which 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 (15.3). 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 below, bears resemblance to the value of money, which also looses value over time exponentially due to inflation.

We notice that C^T is a sum of T time steps. T is called the *planning horizon*, or simply: *horizon*. We distinguish three important cases:

1. $T = 1$. This is 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 solutions are currently the best known solutions that can be computed in reasonable time. Obviously, greedy optimization is invariant with respect to the discount factor γ, as long as $\gamma > 0$.

2. T is finite (but larger than 1). This case is known as the *finite-horizon case*. Typically, the payoff is not discounted over time, that is, $\gamma = 1$. 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 action is a function of time horizon. Near the far end of the time horizon, for example, the optimal action might differ substantially from the optimal action 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.

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's 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 amount of money, rendering the expected cumulative payoff C^T insufficient to select the better program.

Under the assumption that each individual c is bounded in magnitude (that is, $|c| < c_{\max}$ for some value $c_{\max}$), discounting guarantees that C^∞ is finite— despite the fact that the sum has infinitely many terms. In our example, we have

$$
\begin{aligned}
C^\infty &= c + \gamma c + \gamma^2 c + \gamma^3 c + \dots & (15.4)\\
&= \frac{c}{1-\gamma} & (15.5)
\end{aligned}
$$

where c is either \$1, or \$100. Using, for example, a discount factor of $\gamma = 0.99$, we find that our first program gives us a discounted return of $C = \$100$, whereas the second results in $C = \$10,000$. More generally, C^∞ is finite γ as long as it is smaller than 1. An popular alternative to discounting involve maximizing the *average* payoff instead of the total payoff. Algorithms for maximizing average payoff will not be studied in this book.

We will now introduce a few, useful variations on the basic notation. In particular, sometimes we would like to refer to the cumulative payoff C^T conditioned on the state at time t being s. This will be written as follows:

$$
C^T(s) = E\left[\sum_{\tau=1}^{T}\gamma^\tau c_{t+\tau}|s^{(t)} = s\right] \tag{15.6}
$$

The cumulative payoff C^T is a function the robot's policy for action selection. Sometimes, it is beneficial to make this dependence explicit:

$$
C^T_\pi = E\left[\sum_{\tau=1}^{T}\gamma^\tau c_{t+\tau}|a^{(t)} = \pi(d^{(0\dots t)})\right] \tag{15.7}
$$

This notation enables us to compare two control policies π and π', and determine which one is better. Simply compare C_π^T to $C_{\pi'}^T$ and pick the algorithm with higher expected discounted future payoff.

Finally, we notice that the expected cumulative payoff is often referred to as *value*, to contrast them with the immediate state payoff denoted c.

15.3.2 Finding Control Policies in Fully Observable Domains

Traditionally, the robotics planning literature has investigated the planning problem predominately in *deterministic* and *fully observable* worlds. Here the assumption is that $P(s' \mid a, s)$ and $P(o \mid s)$ both are point mass distributions—which is a fancy way of saying that state transitions and observations are deterministic. Moreover, the measurement function $P(o \mid s)$ is usually assumed to be bijective, which implies that the state s can be determined from the observation o. In such cases, a perfectly acceptable plan is a fixed sequence of actions that does not involve sensing during plan execution. Naturally, this case plays no role in the probabilistic setting, as it does not accommodate the inherent uncertainty in robotics.

More realistic is the case where $P(s' \mid a, s)$ is stochastic, but the state s is fully observable. Two reasons make it worthwhile to study this special case in some depth. First, it encompasses some of the uncertainty in robotics, making sensing an integral part of plan execution. Second, it prepares us for the more general case of partial observability in that it allows us to introduce the idea of value iteration. The framework for action selection in stochastic environments with fully observable state is known as *Markov decision processes*, abbreviated as *MDPs*.

The problem that arises in MDPs is that of determining a policy π that maximizes the expected future payoff C^T. Since the state is fully observable, it suffices to condition the policy on the current state s, instead of the entire data history as in (15.2). Thus, we assume the policy is of the form

$$\pi : s \longrightarrow a \tag{15.8}$$

Why the policy can be conditioned on the most recent state only should be immediately obvious. Formally, this follows directly from the Markov assumption defined in Chapter 2.4.4 of this book.

Let us now derive a basic algorithm for constructing such a policy known as *value iteration*. Let us begin with defining the optimal policy for a planning horizon of $T = 1$, that is, we are only interested in a policy that maximizes the immediate next payoff. The optimal policy is denoted $\pi^1(s)$ and is obtained by maximizing the expected payoff over all actions a:

$$\pi^1(s) \quad = \quad \underset{a}{\mathrm{argmax}} \int c(s') \, P(s'|a,s) \, ds' \tag{15.9}$$

Thus, an optimal action is that that maximizes the immediate next payoff in expectation, and the policy that chooses such an action is optimal.

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:

$$C^1(s) \quad = \quad \underset{a}{\max} \, \gamma \int c(s') \, P(s'|a,s) \, ds' \tag{15.10}$$

The definition of the optimal policy with horizon 1 and the corresponding value function enables us to determine the optimal policy for the planning horizon $T = 2$. In particular, the optimal policy for $T = 2$ selects the action that maximizes the one-step optimal value $C^1(s)$:

$$\pi^2(s) \quad = \quad \underset{a}{\mathrm{argmax}} \int [c(s') + C^1(s')] \, P(s'|a,s) \, ds' \tag{15.11}$$

It should be immediately obvious why this policy is optimal. The value of this policy conditioned on the state s is given by the following discounted expression:

$$C^2(s) \quad = \quad \underset{a}{\max} \int [c(s') + C^1(s')] \, P(s'|a,s) \, ds' \tag{15.12}$$

Notice that 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 value function, can be obtained recursively from the optimal policy and value function $T - 1$.

This is indeed the case. We can recursively construct the optimal policy and the corresponding value function for the planning horizon T from the optimal value function for $T - 1$, via the following equation:

$$\pi^T(s) \quad = \quad \underset{a}{\mathrm{argmax}} \int [c(s') + C^{T-1}(s')] \; P(s'|a, s) \; ds' \tag{15.13}$$

The resulting policy $\pi^T(s)$ is optimal for the planning horizon T. Analogously, its value function is defined through the following recursion:

$$C^T(s) \quad = \quad \underset{a}{\max} \int [c(s') + C^{T-1}(s')] \; P(s'|a, s) \; ds' \tag{15.14}$$

Equations (15.13) and (15.14) give us a recursive definition of the optimal policy and optimal value function for any finite horizon T.

In the infinite horizon case, the optimal value function reaches an an equilibrium:

$$C^\infty(s) \quad = \quad \underset{a}{\max} \int [c(s') + C^\infty(s')] \; P(s'|a, s) \; ds' \tag{15.15}$$

This invariance is known as *Bellman equation*. Without proof, we notice that every value function C which satisfies the following recursion:

$$C(s) \quad = \quad \underset{a}{\max} \int [c(s') + C(s')] \; P(s'|a, s) \; ds' \tag{15.16}$$

is optimal, in the sense that the policy that is greedy with respect to C

$$\pi(s) \quad = \quad \underset{a}{\mathrm{argmax}} \int [c(s') + C(s')] \; P(s'|a, s) \; ds' \tag{15.17}$$

maximizes the infinite-horizon payoff.

15.3.3   Value Iteration

This consideration leads to the definition of value iteration, a popular and decades-old algorithm for calculating the optimal infinite-horizon value function. Value iteration is

a practical algorithm for computing the optimal policy fully observable problems with finite state and action spaces. It does this by successively approximating the optimal value functions, as defined in (15.16).

Let us denote the approximation by $\hat{C}$ Initially, the approximation is set to zero (or, alternatively, some large negative value):

$$\hat{C} \quad \longleftarrow \quad 0 \qquad\qquad\qquad\qquad\qquad (15.18)$$

Value iteration then successively updates the approximation via the following recursive rule, which computes the value of a state s from the best expected value one time step later:

$$\hat{C}(s) \quad \longleftarrow \quad \max_a \int [c(s') + \hat{C}(s')] \, P(s'|a, s) \, ds' \qquad\qquad (15.19)$$

Notice that 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 observed after much smaller number of iterations.

At any point in time, the value function $\hat{C}(s)$ defines the policy of greedily maximizing $\hat{C}(s)$:

$$\pi(s) \quad = \quad \underset{a}{\mathrm{argmax}} \int [c(s') + \hat{C}(s')] \, P(s'|a, s) \, ds' \qquad\qquad (15.20)$$

After convergence of value iteration, the policy that is greedy with respect to the final value function is optimal.

Table 15.1 outlines the basic value iteration algorithm **MDP_value_iteration**. The value function is initialized in lines 2 and 3. Line 4 through 6 implement the recursive calculation of the value function. If the state space is discrete, the integral is replaced by a finite sum. Once value iteration converges, the resulting value function $\hat{C}$ induces the optimal policy.

1:         **Algorithm MDP_value_iteration():**

2:           *for all s do*

3:              $\hat{C}(s) = 0$

4:           *repeat until convergence*

5:            *for all s*

6:               $\hat{C}(s) = \max_a \int [c(s') + \hat{C}(s')]\ P(s'|a, s);ds'$

7:           *return $\hat{C}$*

Table 15.1   The value iteration algorithm for MDPs with fi nite state and action spaces.

15.3.4   Illustration

Figure 15.2 illustrates value iteration in the context of a robotic path planning problem. Shown there is a 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 15.2.

The value function is shown in grey, where the brighter a location, the higher its value. Following the color gradient leads to the respective goal location, as indicated by the path shown in Figure 15.2. 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 15.2 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 x-y-space only, basically ignoring the cost of rotation. It is also quite common to ignore the robot's velocity, despite the fact that it clearly constrains where a robot can move. Obviously, path planners that plan in x-y-space are unable to factor the cost of rotation into the value function, and they cannot deal with robot dynamics (velocities). It is therefore common practice to combine such path planners with fast, reactive collision avoidance modules that
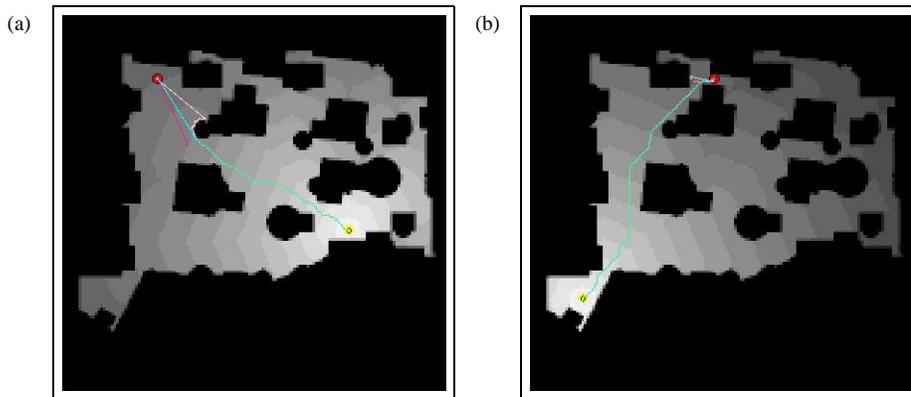
Figure 15.2 Example of value iteration over state spaces in robot motion. Obstacles are shown in black. The value function is indicated by the grayly shaded area. Greedy action selection with respect to the value function lead to optimal control, assuming that the robot's pose is observable. Also shown in the diagrams are example paths obtained by following the greedy policy.

generate motor velocities while obeying dynamic constraints. A path planner which 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.

16

PARTIALLY OBSERVABLE MARKOV DECISION PROCESSES

16.1   MOTIVATION

Let us now shift focus to the partially observable problem. The algorithms discussed thus far only address uncertainty in action effects, but they assume that the state of the world can be determined with certainty. For fully observable Markov decision processes, we devised a value iteration algoithm for controlling robots in stochastic domains. We will now be interested in the more general case where the state is not fully observable. Lack of observability means that the robot can only estimate a posterior distribution over possible world state, which we refer to as the belief b. This setting is known as *Partially Observable Markov Decision Processes*, or *POMDPs*.

The central question addressed in the remainder of this chapter is the following: Can we devise planning and control algorithms for POMDPs? The answer is positive, but with caveats. Algorithms for finding the optimal policy only exist for finite worlds, where the state space, the action space, the space of observations, and the planmning horizon T are all finite. Unfortunately, these exact methods are computationally extremely complex. For the more interesting continous case, the best known algorihms are all approximative.

All algorithms studied on this chapter build on the value iteration approach discussed previously. Let us restate Equation (15.14), which is the central update equation in value iteration in MDPs:

$$C^T(s) \;\; = \;\; \max_a \int [c(s') + C^{T-1}(s')] \, P(s'|a, s) \, ds' \tag{16.1}$$

In POMDPs, we apply the very same idea. Howewver, the state s it not observable. All we have is the belief state b, which is a posterior distributon over states. POMDPs, thus, compute a value function over belief spaces:

$$C^T(b) \quad = \quad \max_a \int [c(b') + C^{T-1}(b')] \ P(b'|a, b) \ db' \tag{16.2}$$

and use this value function to generate control:

$$\pi^T(b) \quad = \quad \operatorname*{argmax}_a \int [c(b') + C^{T-1}(b')] \ P(b'|a, b) \ db' \tag{16.3}$$

Unfortunately, calculating value functions is more complicated in belief space than it is in state space. A belief is a probability distribution; thus, values C^T in POMDPs are functions over probability distributions. This is problematic. If the state space is finite, the belief space is continuous, since it is the space of all distributions over the state space. The situation is even more delicate for continuous state spaces, where the belief space is an infinitely-dimensional continuum. Furthermore, Equations (16.2) and (16.3) integrate over all beliefs b'. Given the complex nature of the belief space, it is not at all obvious that the integration can be carried out exactly, or that effective approximations can be found.

Luckily, exact solutions exist for the interesting special case of finite worlds...

This chapter is organized as follows.

- Section 16.2 discusses a more general planning algorithm that addresses both types of uncertainty: Uncertainty in action and uncertainty in perception. This algorithm adopts the idea of value iteration, but applies it to a belief space representation. The framework underlying this algorithm is called Partially Observable Markov Decision Processes (POMDPs). The POMDP algorithm can anticipate uncertainty, actively gather information, and explore optimally in pursuit of an arbitrary performance goal, at the expense of increased computational complexity.

- Finally, Section 16.5 discusses augmented MDP algorithms, which are crossovers between MDP and POMDP algorithms. Augmented MDP algorithms consider uncertainty, but abstract away detail and hence are computationally much more efficient than POMDP solutions.

16.2 FINITE ENVIRONMENTS

We begin our consideration with the important special case of finite worlds. Here we assume that we have a finite state space, finitely many actions a at each state, a finite number of different observations o, and a finite planning horizon T. Under these conditions, the optimal value function can be calculated exactly, as can the optimal policy. This is not at all obvious. Even if the state space is finite, there are in fact infinitely many possible beliefs. However, in this and the following section we establish a well-known result that the optimal value function is convex and composed of finitely many linear pieces. Thus, even though the value function is defined over a continuum, it can be represented on a digital computer—up to the accuracy of floating point numbers.

16.2.1 An Illustrative Example

An example of a finite world is shown in Figure 16.1. This specific example is extremely simple and artificial, and its sole role is to familiarize the reader with the key issues of POMDPs before discussion the general solution. We notice that the world in Figure 16.1 possesses four states, labeled s_1 through s_4, two actions, a_1 and a_2, and two observations, labeled o_1 and o_2. The initial state is drawn at random from the two top states, s_1 and s_2. The robot is now given a choice: executing action a_1, which will with high probability (but not always) teleport it to the respective other state. Alternatively, it can execute action a_2, which results in a transition in one of the two bottom states, s_3 or s_4 with the probabilities as indicated. In s_3 the robot will receive a large positive payoff, whereas in s_4 the payoff is negative. Both of those states are terminal states, that is, once entered the task is over. Thus, the robot's goal is to execute action a_2 when in state s_1. If the robot knew what state it was in, performing the task would be extremely easy: Simply apply action a_1 until the state is s_1, then apply action a_2. However, the robot does not know its state. Instead, it can perceive the two observations o_1 and o_2. Unfortunately, both observations are possible at both of these states, though the probability of observing one versus the other is different, depending on what state the robot is in. Since the robot does not know whether its initial state is s_1 or s_2, it must carefully keep track of past observations to calculate its belief. So the key policy questions are: When should the robot try action a_2? How confident must it be, and how long will it take to try this action? These and other questions will be answered further below.

Our goal in this section is to develop an algorithm for computing the optimal value function exactly for finite worlds with finite horizon T. Let us denote the planning horizon by T and the states by $s_1, \ldots, s_n$. Exploiting the finiteness of the state space,
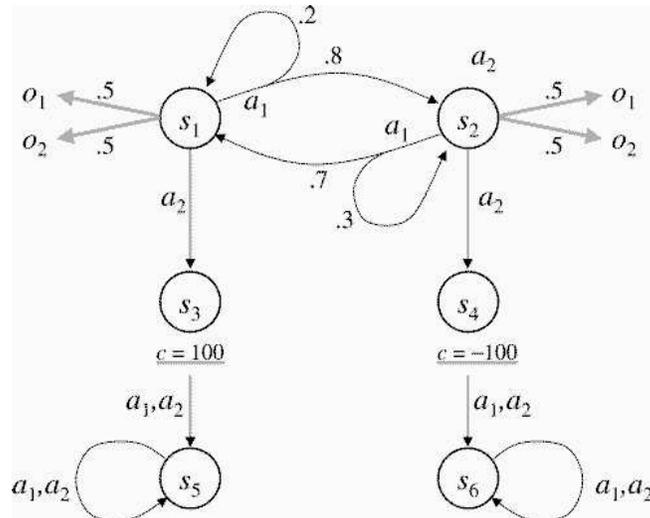
```
four states
states s1, s2 are initial states
action a1: change states with with prob 0.9 (if in s1)  0.8 (if in s2)
action s2: go to s3, s4. 90% chance that we'll go to the other state
payoff: 100 in s3, -100 in s4
observations: o1 .7 in s1, 0.4 in s2
observations: o2 .3 in s1, 0.6 in s2
```

Figure 16.1 Finite State Environment, used to illustrate value iteration in belief space.

we notice that the belief $b(s)$ is given by n probabilities

$$p_i \quad = \quad b(s = s_i) \tag{16.4}$$

with $i = 1, \ldots, n$. The belief must satisfy the conditions

$$p_i \quad \geq \quad 0$$
$$\sum_{i=1}^{n} p_i \quad = \quad 1 \tag{16.5}$$

Because of the last condition, $b(s)$ can be specified by $n - 1$ parameters $p_1, \ldots, p_{n-1}$ instead of n parameters. The remaining probability p_n can be calculated as follows:

$$p_n \quad = \quad 1 - \sum_{i=1}^{n} p_i \tag{16.6}$$

Thus, if the state space is finite and of size n, a belief is a $(n - 1)$-dimensional vector.

In our example shown in Figure 16.1, it might appear that the belief is specified by three numerical probability values, since there are four states. However, the action a_2 separates (with certainty) the states $\{s_1, s_2\}$ from the states $\{s_3, s_4\}$. Thus, the only uncertainty that the robot may encounter is a confusion between states s_1 and s_2, and a confusion between states s_3 and s_4. Both can be represented by a single numerical probability value, thus, the only continuous component of the belief state is one-dimensional. This makes this example convenient for plotting value functions.

Let us now focus on the question as to whether we can calculate the optimal value function and the optimal policy exactly in finite domains. If not, we might be forced to approximate it. At first, one might conclude that calculating the optimal value function is impossible, due to the fact that the belief space is continuous. However, we observe that for finite worlds, the value function has a special shape: It is composed of finitely many linear pieces. This makes it possible to calculate the optimal value function in finite time. We also notice that the value function is convex and continuous—these latter two properties also apply to optimal value functions over continuous state spaces and infinite planning horizons.

We begin our consideration with the immediate payoff of a belief state b. Recall that the payoff of b is given by the expectation of the payoff c under the probability distribution b:

$$c(b) \quad = \quad \int c(s)\, b(s)\, ds \tag{16.7}$$

Using the fact that b is uniquely specified by the probabilities $p_1, \ldots, p_n$, we can write

$$c(b) \quad = \quad c(p_1, \ldots, p_n) \quad = \quad \sum_{i=1}^{n} c(s_i)\, p_i \tag{16.8}$$

which is indeed linear in $p_1, \ldots, p_n$.
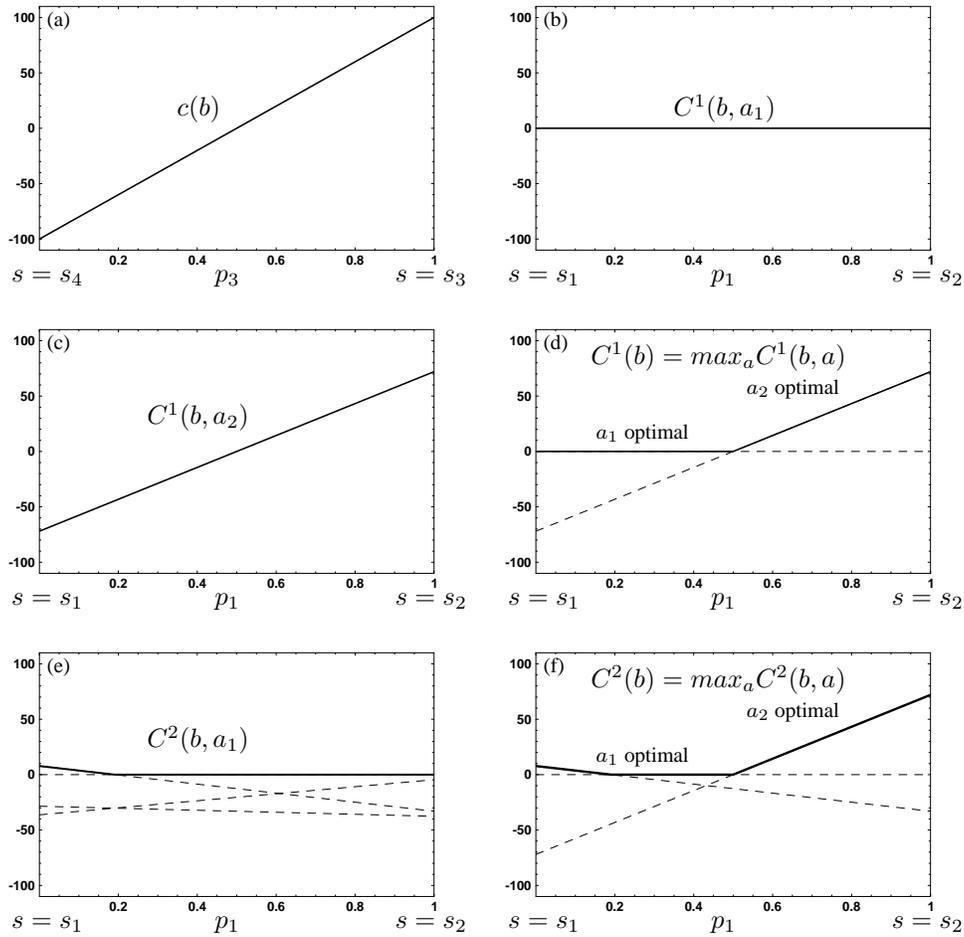
Figure 16.2 Expected payoff $c(b)$ as a function of the belief parameter p_3, assuming that the robot is either in state s_3 or s_4.

It is interesting to plot the function $c(b)$ for belief distributions over the two states s_3 and s_4 in our example, the only states with non-zero payoff. The payoff in state s_3 is 100, whereas the payoff in state s_4 is -100. Figure 16.2a shows the function $c(b)$ for the subspace of the belief defined by $\langle 0, 0, p_3, 1 - p_3 \rangle$ which is a belief space that places all probability in the states s_3 and s_4. Here the expectation $c(b)$ is plotted as a function of the probability p_3. Obviously, if $p_3 = 0$, the environment state is s_4, and the payoff will be $c(s_4) = -100$. On the other hand, if $p_3 = 1$, the environment's

state is s_3 and the payoff will be $c(s_3) = 100$. In between, the expectation is linear, leading to the graph shown in Figure 16.2a.

This consideration enables us to calculate the value function for planning horizon $T = 1$. From now on, we will only consider the subset of the belief space that places all probability on the two states s_1 and s_2. This belief space is parameterized by a single parameter, p_1, since $p_2 = 1 - p_1$ and $p_3 = p_4 = 0$. The value function $C^1(b, a_1)$ is constant zero for action a_1:

$$C^1(b, a_1) \quad = \quad 0 \tag{16.9}$$

since whatever the true state of the robot, action a_1 will not lead it to a state that makes it receive non-zero payoff. This value function $C^1(b, a_1)$ if graphed in Figure 16.2b. The picture becomes more interesting for action a_2. If the state of the environment is s_1, this action will lead with 90% chance to state s_3, where the robot will receive a payoff of 100. With 10% probability it will end up in state s_4, where its payoff will be -100. Thus, the expected payoff in state s_1 is $0.9 \cdot 100 + 0.1 \cdot (-100) = 80$. By an analogous argument, the expected payoff in state s_2 is -80. In between, the expectation is linear, yielding the value function

$$C^1(b, a_2) \quad = \quad \gamma(80p_1 - 80p_2) \quad = \quad 72p_1 - 72p_2 \tag{16.10}$$

Here we use the discount factor $\gamma = 0.9$. This function is shown in Figure 16.2c, for beliefs of the type $\langle p_1, 1 - p_1, 0, 0 \rangle$.

So what is the right action selection policy? Following the rationale of maximizing expected payoff, the best action depends on our current belief, assuming that it accurate reflects our knowledge about the real world. If the probability $p_1 \geq 0.5$, the optimal action will be a_2, since we expect positive payoff. For values smaller than 0.5, the optimal action will be a_1, since it avoids the negative expected payoff associated with action a_2. The corresponding value function is the maximum of the action-specific value functions:

$$
\begin{aligned}
C^1(b) \quad &= \quad \max_a C^1(b, a) \\
&= \quad \max\{0 \ ; \ 72p_1 - 72p_2\}
\end{aligned}
\tag{16.11}
$$

This value function and the corresponding policy for action selection is illustrated by the solid graph in Figure 16.2d, which maximizes the two linear components indicated

by the dashed lines. We notice that this value function is not linear any longer. Instead, it is piecewise linear and convex. The non-linearity arises from the fact that different actions are optimal for different parts of the belief space.

From the value function in Figure 16.2d, can we conclude that for beliefs $p_1 < 0.5$ there is no way to reap payoffs larger than 0? Of course, the answer is no. The value function $C^1(b)$ is only optimal for the horizon $T = 1$. For larger horizons, it is possible to first execute action a_1, followed by action a_2. Executing action a_1 has two beneficial effects: First, it helps us estimate the current state better due to the fact that the we can sense, and second, with high probability it changes the state from s_1 to s_2 or vice versa. Thus a good policy might be to execute action a_1 until we are reasonably confident that the state of the robot is s_1. Then, the robot should execute action a_2.

To make this more formal, let us derive the optimal value function for horizon $T = 2$. Suppose we execute action a_1. Then two things can happen: We either observe o_1 or o_2. Let us first assume we observe o_1. Then the new belief state can be computed using the Bayes filter:

$$
\begin{aligned}
p_1' &= \eta_1 \, P(o_1'|s_1') \sum_{i=1}^{2} P(s_1'|a_1, s_i) \, p_i \\
&= \eta_1 \, 0.7(0.1p_1 + 0.8p_2) \\
&= \eta_1 \, (0.07p_1 + 0.56p_2)
\end{aligned}
\tag{16.12}
$$

where η_1 is the normalizer in Bayes rule. Similarly, we obtain for the posterior probability of being in s_2:

$$
\begin{aligned}
p_2' &= \eta_1 \, P(o_1'|s_2') \sum_{i=1}^{2} P(s_2'|a_1, s_i) \, p_i \\
&= \eta_1 \, 0.4(0.9p_1 + 0.2p_2) \\
&= \eta_1 \, (0.36p_1 + 0.08p_2)
\end{aligned}
\tag{16.13}
$$

Since we know that $p_1' + p_2' = 1$—after all, when executing action a_1 the state will either be s_1 or s_2—we obtain for η_1:

$$
\eta_1 \;=\; \frac{1}{0.07p_1 + 0.56p_2 + 0.36p_1 + 0.08p_2} \;=\; \frac{1}{0.43p_1 + 0.64p_2}
\tag{16.14}
$$

We also notice that the variable η_1 is a useful probability: It is the probability of observing o_1 after executing action a_1 (regardless of the posterior state).

We now have expression characterizing the posterior belief given that we execute action a_1 and observe o_1. So what is the value of this belief state? The answer is obtained by plugging the new belief into the value function $C^1(b)$ as defined in Equation (16.11), and discounting the result by γ:

$$
\begin{aligned}
C^2(b, a_1, o_1) &= \gamma \ \max\{0 \ ; \ 72p'_1 - 72p'_2\} \\
&= 0.9 \ \max\{0 \ ; \ 72 \ \eta_1 \ (0.36p_1 + 0.08p_2) - 72 \ \eta_1 \ (0.43p_1 + 0.64p_2)\} \\
&= 0.9 \ \max\{0 \ ; \ 72 \ \eta_1 \ (0.36p_1 + 0.08p_2 - 0.43p_1 - 0.64p_2)\} \\
&= 0.9 \ \max\{0 \ ; \ 72 \ \eta_1 \ (-0.07p_1 - 0.56p_2)\} \\
&= 0.9 \ \max\{0 \ ; \ \eta_1 \ (-5.04p_1 - 40.32p_2)\} \qquad \text{(16.15)}
\end{aligned}
$$

We now move the factor η_1 out of the maximization and move 0.9 inside, and obtain:

$$
C^2(b, a_1, o_1) \quad = \quad \eta_1 \ \max\{0 \ ; \ -4.563p_1 - 36.288p_2\} \qquad \text{(16.16)}
$$

which is a piecewise linear, convex function in the parameters of the belief b.

The derivation for observing o_2 after executing action a_1 is completely analogous. In particular, we obtain the posterior

$$
\begin{aligned}
p'_1 &= \eta_2 \ 0.3(0.1p_1 + 0.8p_2) &= \eta_2 \ (0.03p_1 + 0.24p_2) \\
p'_2 &= \eta_2 \ 0.6(0.9p_1 + 0.2p_2) &= \eta_2 \ (0.54p_1 + 0.12p_2) \qquad \text{(16.17)}
\end{aligned}
$$

with the normalizer

$$
\eta_2 \quad = \quad \frac{1}{0.57p_1 + 0.36p_2} \qquad \text{(16.18)}
$$

The corresponding value is

$$
\begin{aligned}
C^2(b, a_1, o_1) &= \gamma \ \max\{0 \ ; \ 72p'_1 - 72p'_2\} \\
&= \eta_2 \ \max\{0 \ ; \ -33.048p_1 + 7.776p_2\} \qquad \text{(16.19)}
\end{aligned}
$$

As an aside, we also notice that $\eta_1^{-1} + \eta_2^{-1} = 1$. This follows directly from the fact that the normalizer in Bayes filters is the observation probability

$$\eta_i = \frac{1}{P(o_i'|a_1, b)} \tag{16.20}$$

and the fact that there are exactly two possible observations in our example, o_1 and o_2.

Let us now calculate the value $C^2(b, a_1)$, which is the expected value upon executing action a_1. Clearly, the value is a mixture of the values $C^2(b, a_1, o_1)$ and $C^2(b, a_1, o_2)$, weighted by the probabilities of actually observing o_1 and o_2, respectively. Put into mathematical notation, we have

$$C^2(b, a_1) = \sum_{i=1}^{2} C^2(b, a_1, o_i) P(o_i|a_1, b) \tag{16.21}$$

The terms $C^2(b, a_1, o_i)$ were already defined above. The probability $P(o_i|a_1, b)$ of observing o_i after executing action a_1 is η_i^{-1}. Thus, we have all the ingredients for calculating the desired value $C^2(b, a_1)$:

$$
\begin{aligned}
C^2(b, a_1) &= \eta_1^{-1}\, \eta_1\, \max\{0\ ;\ -4.563p_1 - 36.288p_2\} \\
&\qquad + \eta_2^{-1}\, \eta_2\, \max\{0\ ;\ -33.048p_1 + 7.776p_2\} \tag{16.22} \\
&= \max\{0\ ;\ -4.563p_1 - 36.288p_2\} + \max\{0\ ;\ -33.048p_1 + 7.776p_2\}
\end{aligned}
$$

This expression can be re-expressed as the maximum of four linear functions:

$$
\begin{aligned}
C^2(b, a_1) &= \max\{0\ ;\ -4.563p_1 - 36.288p_2\ ; \tag{16.23} \\
&\qquad -33.048p_1 + 7.776p_2\ ;\ -37.611p_1 - 28.512p_2\}
\end{aligned}
$$

Figure 16.2e shows those four linear functions for the belief space $\langle p_1, 1 - p_2, 0, 0 \rangle$. The value function $C^2(b, a_1)$ is the maximum of those four linear functions. As is easy to be seen, two of these functions are sufficient to define the maximum; the other two are smaller over the entire spectrum. This enables us to rewrite $C^2(b, a_1)$ as the maximum of only two terms, instead of four:

$$C^2(b, a_1) = \max\{0\ ;\ -33.048p_1 + 7.776p_2\} \tag{16.24}$$

Finally, we have to determine the value $C^2(b)$, which is the maximum of the following two terms:

$$C^2(b) \quad = \quad \max\{C^2(b, a_1), C^2(b, a_2)\} \tag{16.25}$$

As is easily verified, the second term in the maximization, $C^2(b, a_2)$, is exactly the same as above for planning horizon $T = 1$:

$$C^2(b, a_2) \quad = \quad C^1(b, a_2) \quad = \quad 72p_1 - 72p_2 \tag{16.26}$$

Hence we obtain from (16.24) and (16.26):

$$C^2(b) \quad = \quad \max\{0 \; ; \; -33.048p_1 + 7.776p_2 \; ; \; 72p_1 - 72p_2\} \tag{16.27}$$

This function is the optimal value function for planning horizon T. Figure **??** graphs $C^2(b)$ and its components for the belief subspace $\langle p_1, 1 - p_2, 0, 0 \rangle$. As is easy to be seen, the function is piecewise linear and convex. In particular, it consists of three linear pieces. For beliefs under the two leftmost pieces (i.e., $p_1 < 0.5$), a_1 is the optimal action. For $p_1 = 0.5$, both actions are equally good. Beliefs that correspond to the rightmost linear piece, that is, beliefs with $p_1 > 0.5$, have the optimal action a_2. If a_1 is the optimal function, the next action depends on the initial point in belief space and on the observation.

The value function $C^2(b)$ is only optimal for the horizon 2. However, our analysis illustrates several important points.

First, the optimal value function for any finite horizon is continuous, piecewise linear, and convex. Each linear piece corresponds to a different action choice at some point in the future, or to a different observation that can be made. The convexity of the value function indicates the rather intuitive observation, namely that knowing is always superior to not knowing. Given two belief states b and b', the mixed value of the belief states is larger or equal to the value of the mixed belief state, for some mixing parameter β with $0 \le \beta \le 1$:

$$\beta C(b) + (1 - \beta)C(b') \quad \ge \quad C(\beta b + (1 - \beta)b') \tag{16.28}$$

Second, the number of linear pieces can grow tremendously, specifically if one does not pay attention to linear functions becoming obsolete. In our toy example, two

out of four linear constraints defining $C^2(b, a_1)$ were not needed. The 'trick' of efficiently implementing POMDPs lies in identifying obsolete linear functions as early as possible, so that no computation is wasted when calculating the value function. Unfortunately, even if we carefully eliminate all unneeded linear constraints, the number of linear functions can still grow extremely rapidly. This poses intrinsic scaling limitations on the exact value iteration solution for finite POMDPs.

16.2.2   Value Iteration in Belief Space

The previous section showed, by example, how to calculate value functions in finite worlds. Let us now return to the general problem of value iteration in belief space. In particular, in the introduction to this chapter we stated the basic update equation for the value function, which we briefly restate here:

$$C^T(b) = \max_a \int [c(b') + C^{T-1}(b')] \ P(b'|a, b) \ db' \qquad (16.29)$$

In this and the following sections, we will develop a general algorithm for value iteration in belief space that can be implemented on a digital computer. We begin by noticing that Equation (16.2) suggests an integration over all beliefs, where each belief is a probability distribution. If the state space is finite and the number of states is n, the space of all probability distributions is a continuum with dimension $n - 1$. To see, we notice that $n - 1$ numerical values are required to specify a probability distribution over n discrete events (the n-th parameter can be omitted since probabilities add up to 1). If the state space is continuous, the belief space possesses infinitely many dimensions. Thus, integrating over all belief appears to be computationally daunting a task. However, we can avoid this integration by reformulating the problem and integrating over observations instead.

Let us examine the conditional probability $P(b'|a, b)$, which specifies a distribution over posterior beliefs b' given a belief b and an action a. If only b and a are known, the posterior belief b' is not unique, and $P(b'|a, b)$ is a true probability distribution over beliefs. However, if if we also knew the measurement o' after executing action a, the posterior b' is unique and $P(b'|a, b)$ is a degenerate point-mass distribution. Why is this so? The answer is provided by the Bayes filter. From the belief b before action execution, the action a, and the subsequent observation o', the Bayes filter calculates a single, posterior belief b' which is the single, correct belief. Thus, we conclude that if only we knew o', the integration over all beliefs in (16.2) would be obsolete.

This insight can be exploited by re-expressing

$$P(b'|a,b) = \int P(b'|a,b,o') \, P(o'|a,b) \, do' \tag{16.30}$$

where $P(b'|a,b,o')$ is a point-mass distribution focussed on the single belief calculated by the Bayes filer. Plugging this integral into the definition of value iteration (16.2), we obtain

$$C^T(b) = \max_a \int \int [c(b') + C^{T-1}(b')] \, P(b'|a,b,o') \, db' \, P(o'|a,b) \, do' \tag{16.31}$$

The inner integral

$$\int [c(b') + C^{T-1}(b')] \, P(b'|a,b,o') \, db' \tag{16.32}$$

contains only one non-zero term. This is the term where b' is the distribution calculated from b, a, and o' using Bayes filters. Let us call this distribution $B(b,a,o')$, that is,

$$
\begin{aligned}
B(b,a,o')(s') &= P(s'|o',a,b) \\
&= \frac{P(o'|s',a,b) \, P(s'|a,b)}{P(o'|a,b)} \\
&= \frac{1}{P(o'|a,b)} \, P(o'|s') \int P(s'|a,b,s) \, P(s|a,b) \, ds \\
&= \frac{1}{P(o'|a,b)} \, P(o'|s') \int P(s'|a,s) \, b(s) \, ds \tag{16.33}
\end{aligned}
$$

The reader should recognize the familiar Bayes filter derivation that was extensively discussed in Chapter 2, this time with the normalizer made explicit. We notice that the normalizer, $P(o'|a,b)$, is a factor in the value update equation (16.31). Hence, substituting $B(b,a,o')$ into (16.31) eliminates this term, which leads to the recursive description of the value iteration algorithm:

$$C^T(b) = \max_a \int [c(B(b,a,o')) + C^{T-1}(B(b,a,o'))] \, P(o'|a,b) \, do' \tag{16.34}$$

This form is more convenient than the one in (16.2), since it only requires integration over all possible measurements o', instead of all possible belief distributions b'. This transformation is a key insight for all value iteration algorithms derived in this chapter. In fact, it was used implicitly in the example above, where a new value function was obtained by mixing together finitely many piecewise linear functions in Equation (16.22).

Below, it will be convenient to split the maximization over actions from the integration. Hence, we notice that that (16.34) can be rewritten as the following two equations:

$$C^T(b, a) = \int [c(B(b, a, o')) + C^{T-1}(B(b, a, o'))] \, P(o'|a, b) \, do' \quad (16.35)$$

$$C^T(b) = \max_a C^T(b, a) \quad (16.36)$$

Here $C^T(b, a)$ is the horizon T value function over the belief b assuming that the immediate next action is a.

16.2.3   Calculating the Value Function

In our example, the optimal value function was piecewise linear and convex. Is this specific to this example, or is this a general characteristic of value functions in finite worlds? Luckily, it turns out that the latter is the case: All optimal value functions in finite POMDPs with finite horizon are piecewise linear and convex. Piecewise linearity means that value function C^T is represented by a collection of linear functions. If C^T was linear, it could be represented by set of coefficients $C_1^T, \ldots, C_n^T$:

$$C^T(b) = C^T(p_1, \ldots, p_n) = \sum_{i=1}^{n} C_i^T p_i \quad (16.37)$$

where as usual, $p_1, \ldots, p_n$ are the parameters of a belief distribution. As in our example, a piecewise linear and convex value function $C^T(b)$ can be represented by the maximum of a collection of K linear functions

$$C^T(b) = \max_k \sum_{i=1}^{n} C_{k,i}^T p_i \quad (16.38)$$

where $C_{k,1}^T, \ldots, C_{k,n}^T$ denote the parameters of the k-th linear function, and K denotes the number of linear pieces. The reader should quickly convince herself that the maximum of a finite set of linear functions is indeed a convex piecewise linear, convex function.

In value iteration, the initial value function is given by

$$C^0 = 0 \tag{16.39}$$

We notice that this value function is linear, hence by definition it is also piecewise linear and convex. This assignment establishes the (trivial) baseline of the recursive value iteration algorithm.

We will now derive a recursive equation for calculating the value function $C^T(b)$. This equation assumes that the value function one time step earlier, $C^{T-1}(b)$, is represented by a piecewise linear function as specified above. As part of the derivation, we will show that under the assumption that $C^{T-1}(b)$ is piecewise linear and convex, $C^T(b)$ is also piecewise linear and convex. Induction over the planning horizon T then proves that all value functions with finite horizon are indeed piecewise linear and convex.

Equations (16.35) and (16.36) state the following:

$$C^T(b,a) = \int \left[c(B(b,a,o')) + C^{T-1}(B(b,a,o')) \right] P(o'|a,b)\, do' \tag{16.40}$$

$$C^T(b) = \max_a C^T(b,a) \tag{16.41}$$

In finite spaces, all integrals can be replaced by finite sums, and we obtain:

$$C^T(b,a) = \gamma \sum_{o'} \left[c(B(b,a,o')) + C^{T-1}(B(b,a,o')) \right] P(o'|a,b) \tag{16.42}$$

$$C^T(b) = \max_a C^T(b,a) \tag{16.43}$$

The belief $B(b,a,o')$ is obtained using the following expression, which 'translates' Equation (16.33) to finite spaces by replacing the integral by a sum:

$$B(b,a,o')(s') = \frac{1}{P(o'|a,b)}\, P(o'|s') \sum_s P(s'|a,s)\, b(s) \tag{16.44}$$

If the belief b is represented by the parameters $\{p_1, \ldots, p_n\}$, and the belief $B(b, a, o')$ by $\{p'_1, \ldots, p'_n\}$, the i-th parameter of the belief b' is computed as follows:

$$p'_i \;\; = \;\; \frac{1}{P(o'|a,b)} \; P(o'|s'_i) \sum_{j=1}^{n} P(s'_i|a, s_j) \; p_j \qquad (16.45)$$

The reader may recognize the discrete Bayes filter that was already discussed in length in Chapter 4.1, where introduced the basic filtering theory.

To compute the value function $C^T(b, a)$, we will now derive more practical expressions for the terms $c(B(b, a, o'))$ and $C^{T-1}(B(b, a, o'))$, starting with the one in (16.42). With our parameterization $B(b, a, o') = \{p'_1, \ldots, p'_n\}$, we obtain for the term $c(B(b, a, o'))$:

$$c(B(b, a, o')) \;\; = \;\; c(p'_1, \ldots, p'_n) \;\; = \;\; \sum_{i=1}^{n} c_i p'_i \qquad (16.46)$$

Substituting (16.45) into this expression gives us

$$\begin{aligned} c(B(b, a, o')) \;\; &= \;\; \sum_{i=1}^{n} c_i \frac{1}{P(o'|a,b)} \; P(o'|s'_i) \sum_{j=1}^{n} P(s'_i|a, s_j) \; p_j \\ &= \;\; \frac{1}{P(o'|a,b)} \sum_{i=1}^{n} c_i \; P(o'|s'_i) \sum_{j=1}^{n} P(s'_i|a, s_j) \; p_j \qquad (16.47) \end{aligned}$$

The latter transformation is legal since the term $P(o'|a, b)$ does not depend on i. This form still contains an expression that is difficult to compute: $P(o'|a, b)$. However, the beauty of the finite case is that this expression cancels out, as we will see very soon. Hence, it does not have to be considered any further at this point.

The derivation of the term $C^{T-1}(B(b, a, o'))$ in (16.42) is similar to that of $c(B(b, a, o'))$ above. In particular, we have to replace the the immediate payoff function c by the the value function C^{T-1}, which is only piecewise linear and convex. This gives us

$$C^{T-1}(B(b, a, o')) \;\; = \;\; C^{T-1}(p'_1, \ldots, p'_n) \;\; = \;\; \max_k \sum_{i=1}^{n} C^{T-1}_{k,i} p'_i \qquad (16.48)$$

for sets of linear parameters function $C_{k,i}^{T-1}$. As above, we now substitute the definition of the posterior belief B (see (16.45)) into this expression, and obtain:

$$= \max_k \left[\sum_{i=1}^{n} C_{k,i}^{T-1} \frac{1}{P(o'|a,b)} \; P(o'|s_i') \; \sum_{j=1}^{n} P(s_i'|a,s_j) \; p_j \right] \quad (16.49)$$

The term $P(o'|a,b)^{-1}$ can be moved out of the summation and the maximization, since it does not depend on i or k:

$$= \frac{1}{P(o'|a,b)} \max_k \left[\sum_{i=1}^{n} C_{k,i}^{T-1} P(o'|s_i') \; \sum_{j=1}^{n} P(s_i'|a,s_j) \; p_j \right] \quad (16.50)$$

It may not be immediately obvious that this expression is also piecewise linear and convex. However, reordering the terms inside the summation gives us the following expression:

$$= \frac{1}{P(o'|a,b)} \max_k \left[\sum_{j=1}^{n} p_j \left(\sum_{i=1}^{n} C_{k,i}^{T-1} \; P(o'|s_i') \; P(s_i'|a,s_j) \right) \right] \quad (16.51)$$

In this form, it is easy to verify that that for any fixed value of k, the argument of the maximization is indeed linear in the parameters p_j. The maximum of those linear pieces is, thus, piecewise linear and convex. Moreover, the number of linear pieces is finite.

Let us now return to the main problem addressed in this section, namely the problem computing $C^T(b,a)$, the value function for horizon T. We briefly restate Equation (16.42):

$$C^T(b,a) = \gamma \sum_{o'} \left[c(B(b,a,o')) + C^{T-1}(B(b,a,o')) \right] \; P(o'|a,b) \quad (16.52)$$

Let us first calculate the sum $c(B(b,a,o')) + C^{T-1}(B(b,a,o'))$, using Equations (16.47) and (16.50):

$$c(B(b,a,o')) + C^{T-1}(B(b,a,o'))$$

$$= \frac{1}{P(o'|a,b)} \sum_{i=1}^{n} c_i \, P(o'|s_i') \sum_{j=1}^{n} P(s_i'|a,s_j) \, p_j$$

$$+ \frac{1}{P(o'|a,b)} \max_{k} \left[\sum_{i=1}^{n} C_{k,i}^{T-1} P(o'|s_i') \sum_{j=1}^{n} P(s_i'|a,s_j) \, p_j \right] \quad (16.53)$$

With some reordering, we obtain

$$c(B(b,a,o')) + C^{T-1}(B(b,a,o'))$$

$$= \frac{1}{P(o'|a,b)} \max_{k} \left[\sum_{i=1}^{n} (c_i + C_{k,i}^{T-1}) P(o'|s_i') \sum_{j=1}^{n} P(s_i'|a,s_j) \, p_j \right] \quad (16.54)$$

Substituting this expression into Equation (16.52) gives us:

$$C^T(b,a) = \gamma \sum_{o'} \left\{ \frac{1}{P(o'|a,b)} \max_{k} \left[\sum_{i=1}^{n} (c_i + C_{k,i}^{T-1}) \right. \right.$$

$$\left. \left. P(o'|s_i') \sum_{j=1}^{n} P(s_i'|a,s_j) \, p_j \right] \right\} P(o'|a,b) \quad (16.55)$$

We notice that the term $P(o'|a,b)$ appears in enumerator and in the denominator of this expression, thus cancels out:

$$C^T(b,a) = \gamma \sum_{o'} \max_{k} \left[\sum_{i=1}^{n} (c_i + C_{k,i}^{T-1}) P(o'|s_i') \sum_{j=1}^{n} P(s_i'|a,s_j) \, p_j \right] (16.56)$$

This cancellation is an essential characteristic of the POMDP solution in finite worlds, which accounts for a simpler update equation when compared to the general solution described in the previous section. The desired value function is then obtained by maximizing $C^T(b,a)$ over all actions a:

$$C^T(b) = \max_{a} C^T(b,a) \quad (16.57)$$

As is easily verified, $C^T(b)$ is indeed piecewise linear and convex in the belief parameters $p_1, \ldots, p_n$. In particular, for each fixed k the term inside the large brackets in (16.56) is linear. The maximum of those K linear functions is a piecewise linear function that consist of at most K linear pieces. The sum of piecewise linear, convex functions is again piecewise linear and convex. Hence the summation over all observations o' produces again a piecewise linear convex function. Finally, the maximization over all actions in Equation (16.57) again results in a piecewise linear and convex function. Thus, we conclude that $C^T(b)$ is piecewise linear and convex for any finite horizon T.

16.2.4 Linear Programming Solution

How can we turn the mathematical insight into an algorithm that can be carried out on a digital computer in finite time? As in our example above, the key insight is that solutions to equations like the one above can be calculated using linear programming. Linear programming provides solutions for optimization problems under linear constraints.

Suppose we know that

$$C = \max_{a:1 \leq a \leq m} x(a) \tag{16.58}$$

for some fictitious set of values $x(a)$, and some positive integer m. A solution to this equation can be obtained by a linear program which possesses a collection of m linear constraints, one for each possible value of a:

$$\Phi = \{C \geq x(a)\} \tag{16.59}$$

The value of C is then obtained by minimizing C under these constraints, which is a standard linear programming problem. Now suppose we are given the slightly more complicated expression

$$C = \sum_i^n \max_{a:1 \leq a \leq m} x(a, i) \tag{16.60}$$

for some fictitious values $x(a, i)$ and some positive integer n. How can we obtain the solution to this problem by a linear program? Clearly, for each i there might be

a different a that maximizes the inner term $x(a, i)$. The solution to this problem is therefore attained for a specific set of values for a, one for each element in the sum. Let us for a moment assume we knew those values of a. If we denote this set by $a(1), \ldots, a(n)$, we have

$$C \;\; = \;\; \sum_{i}^{n} x(a(i), i) \tag{16.61}$$

which gives us the single linear constraint

$$\Phi \;\; = \;\; \{C \geq \sum_{i}^{n} x(a(i), i)\} \tag{16.62}$$

Unfortunately, we do not know the values of $a(i)$ where C attains its maximum. The linear programming solution for (16.60) contains therefore one constraint for *any* such sequence $a(i)$:

$$\bigcup_{a(i):1 \leq a(i) \leq m, 1 \leq i \leq n} \{C \geq \sum_{i}^{n} x(a(i), i)\} \tag{16.63}$$

The total number of constraints is m^n, since there are n free variables $a(i)$ which each can take on m different values. This constraint set ensures that the maximizing constraint(s) is always included. Minimizing C under these constraints then generates the solution of (16.60).

We also notice that it is common practice to divide the set of constraints into *active* and *passive* constraints, depending on whether they actively constrain the solution for C. In this particular example, active constraints are those where the maximum $\max_a x(a)$ is attained; whereas passive constraints correspond to smaller $x(a)$-values. If we are interested in the value of a that maximizes an expression (the 'argmax'), we simply choose an a whose corresponding constraint is active. Active constraint are easily identified using any of the existing linear programming algorithms.

These insights enable us to provide a concrete algorithm for calculating the optimal value function and the optimal policy. Initially, for horizon $T = 1$, the value function

is defined by (16.39), which we briefly restate here:

$$C^1 \;=\; \gamma \sum_{i=1}^{n} c(s_i)\, p_i \tag{16.64}$$

The corresponding constraint set contains only a single constraint:

$$\Phi^1 \;=\; \{C^1 \le \gamma \sum_{i=1}^{n} c(s_i)\, p_i\} \tag{16.65}$$

Minimizing C^1 under the constrain set Φ^1 gives the desired solution.

Let us now consider how to construct the constraint set Φ^T from the set of constraints Φ^{T-1}. According to Equations (16.56) and (16.57), the desired value function is

$$C^T(b) \;=\; \gamma \max_a \sum_{o'} \max_k \left[\sum_{i=1}^{n} (c_i + C_{k,i}^{T-1}) P(o'|s_i') \sum_{j=1}^{n} P(s_i'|a, s_j)\, p_j \right] \tag{16.66}$$

which is similar in shape to the linear constraints discussed above (Equation (16.60)). For each action a, we therefore have to generate to following set of constraints:

$$\bigcup_{a} \;\; \bigcup_{k(o'):1 \le k(o') \le |\Phi^{T-1}|} \left\{ C^T(b) \le \gamma \sum_{o'} \right.$$
$$\left. \sum_{i=1}^{n} (c_i + C_{k(o'),i}^{T-1}) P(o'|s_i') \sum_{j=1}^{n} P(s_i'|a, s_j)\, p_j \right\} \tag{16.67}$$

which are $|O|^{|\Phi^{T-1}|}$ linear constraints. Here $|O|$ denotes the number of observations, and $|\Phi^{T-1}|$ is the number of constraint set obtained for horizon $T-1$. The total number of constraints is multiplicative in the number of actions $|A|$, thus our simple solution obtains

$$|\Phi^T| \;=\; |A| \cdot |O|^{|\Phi^{T-1}|} \tag{16.68}$$

1:          **Algorithm finite_world_POMDP(T):**

2:              $\Phi^1 = \{C^1 \leq \gamma \sum_{i=1}^{n} c(s_i) \, p_i\}$

3:          *For each t from 2 to T*
4:              $\Phi^t = \emptyset$
5:              $l = 0$
6:              *For each action a*
7:                  *For each $k(1), \ldots, k(|O|)$ with $1 \leq k(o') \leq |\Phi^{t-1}|, 1 \leq o' \leq |O|$*
8:                      $C_{l,j}^T = \gamma \sum_{o'} \sum_{i=1}^{n} (c_i + C_{k(o'),i}^{T-1}) P(o'|s_i') \sum_{j=1}^{n} P(s_i'|a, s_j)$
9:                      $\Phi^t = \Phi^t \cup \{\langle a, C \leq \sum_{j=1}^{n} C_{l,j}^T p_j \rangle\}$
10:                     $l = l + 1$

11:         *return Φ^T*

Table 16.1   The POMDP algorithm for discrete worlds. This algorithm represents the optimal value function by a set of constraints, which are calculated recursively.

constraints for horizon T, with the boundary condition $|\Phi^1| = 1$.

Table 16.1 depicts the value iteration algorithm for finite POMDP.

16.3   GENERAL POMDPS

The previous section derived a value iteration algorithm for POMDPs in finite worlds. In particular, this algorithm requires finite numbers of states, observations, and actions, and it also requires a finite horizon T.

[...]

There are still terms in (16.35) that require further consideration. First, the immediate payoff function c has thus far only been defined over states. Here we need to calculate it for belief distributions. The payoff of a belief b is simply the expectation of the

per-state payoff $c(s)$ under the belief $b(s)$:

$$c(b) \;\; = \;\; E_b[c] \;\; = \;\; \int c(s)\, b(s)\, ds \qquad (16.69)$$

It might seem odd that the payoff depends on the robot's belief. However, the robot's belief represents the true posterior over world states, which justifies the expectation.

Second, we need to derive an expression for the probability $P(o'|a, b)$, which is the distribution of observations o' one might receive after executing b in a state distributed according to b. Deriving this expression is a straightforward mathematical exercise. In particular, let us first integrate over the state s' at which the observation o' is made:

$$P(o'|a, b) \;\; = \;\; \int P(o'|a, b, s')\, P(s'|a, b)\, ds' \qquad (16.70)$$

Thus, s' refers to the state after action execution. As usual, we exploit the Markov property to simplify this expression

$$= \;\; \int P(o'|s')\, P(s'|a, b)\, ds' \qquad (16.71)$$

If we now integrate over states s before action execution:

$$= \;\; \int P(o'|s') \int P(s'|a, b, s)\, P(s|a, b)\, ds\, ds' \qquad (16.72)$$

we obtain the simplified term

$$= \;\; \int P(o'|s') \int P(s'|a, s)\, b(s)\, ds\, ds' \qquad (16.73)$$

Armed with the necessary expression, we can now return to the original question of how to perform value iteration in belief space and give a concrete algorithm. In particular, let us substitute (16.73) back into (16.34), which leads to the following recursive

definition of the value function in belief space:

$$
\begin{aligned}
C^T(b) \quad = \quad \max_a \int \Bigg(\bigg[C^{T-1}(B(b,a,o')) + \int c(s')\, B(b,a,o')(s')\, ds' \bigg] \\
\int P(o'|s') \int P(s'|a,s)\, b(s)\, ds\, ds' \Bigg)\, do' \qquad (16.74)
\end{aligned}
$$

This recursive definition defines the optimal value function for any finite horizon T. The optimal value function for infinite horizons is chacterized by Bellman's equation:

$$
\begin{aligned}
C^\infty(b) \quad = \quad \max_a \int \Bigg(\bigg[C^\infty(B(b,a,o')) + \int c(s')\, B(b,a,o')(s')\, ds' \bigg] \\
\int P(o'|s') \int P(s'|a,s)\, b(s)\, ds\, ds' \Bigg)\, do' \qquad (16.75)
\end{aligned}
$$

As in the MDP case, value iteration approximates $C^\infty(b)$ using a function $\hat{C}$ by repetitively applying this update equation. The central value iteration algorithm is therefore as follows:

$$
\begin{aligned}
\hat{C}(b) \quad \longleftarrow \quad \max_a \int \Bigg(\bigg[\hat{C}(B(b,a,o')) + \int c(s')\, B(b,a,o')(s')\, ds' \bigg] \\
\int P(o'|s') \int P(s'|a,s)\, b(s)\, ds\, ds' \Bigg)\, do' \qquad (16.76)
\end{aligned}
$$

This update equation generalizes value iteration to belief spaces. In particular, it provides a way to backup values among belief spaces using the familiar motion model and the familiar perceptual model. The update equation is quite intuitive: The value of a belief b is obtained by maximizing over all actions a. To determine the value of executing action a under the belief b, Equation (16.76) suggests to consider all observations o'. One such an observation is fixed, the posterior belief is calculated via Bayes filtering, and the corresponding value is calculated. Thus, what is missing is a calculation of the probability of measuring o'. In Equation (16.76), this probabilist is obtained by integrating over all states s, all subsequent states s', and weighing these state combinations by the corresponding probabilities under the belief b and the motion model $P(s'|a,s)$.

1:      **Algorithm POMDP(T):**

2:          *for all b do*
3:              $\hat{C}(b) = 0$

4:          *repeat until convergence*
5:              *for all b do*
6:                  $C^* = -\infty$
7:                  *for all actions a do*
8:                      $$C_a = \int \left(\left[\hat{C}(B(b, a, o')) + \int c(s') \; B(b, a, o')(s') \; ds' \right] \right.$$
                        $$\left. \int P(o'|s') \; \int P(s'|a, s) \; b(s) \; ds \; ds' \right) \; do'$$
9:                  *if $C_a > C^*$*
10:                     $C^* = C_a$

11:             $\hat{C}(b) = C^*$

12:         *return $\hat{C}$*

Table 16.2 The general POMDP algorithm with finite horizon T, where the function $B(b, a, o')$ is as specified in the text. This version leaves open as to how the value function $\hat{C}$ is represented, and how the integrals in Step 8 are calculated.

16.3.1   The General POMDP Algorithm

Table 16.2 depicts a general algorithm for value iteration in belief space. This algorithm is only an in-principle algorithm that cannot be implemented on a digital computer, since it requires integration over infinite spaces, and lacks a specification as to how the value function $\hat{C}$ is represented. The algorithm accepts the planning horizon T as input, which is assumed to be finite. As in the state-based version of value iteration, the value function is initialized by 0 (lines 2 and 3 in Table 16.2). Lines 6 through 11 identify the best action for a belief state b relative to the value function $\hat{C}$. The central update, in line 8, is equivalent to the argument of the maximization in Equation (16.76).

The most important remaining question concerns the nature of $\hat{C}$. To perform value iteration in belief space, we need a way to represent a value function over beliefs. Beliefs are probability distributions. Thus, we need a method for assigning values to probability distributions. This, unfortunately, is a difficult problem that lacks a general solution. As noticed above, belief spaces over continuous state spaces possess infinitely many dimensions, suggesting that the value function is defined over an infinitely-dimensional space. Even for finite state spaces, the value function is defined over a continuum, making it questionable as to whether the optimal value function can be represented on a digital computer. However, it turns out that the value function can be calculated exactly for the important special case of *finite* problems, in particular, for problems with finite state, action, and observation spaces, and with finite planning horizon. An exact such algorithm will be give in the next section.

16.4 A MONTE CARLO APPROXIMATION

Let us now study a concrete algorithm that approximates the general POMDP algorithm, and that has shown promise in practical applications. The basic idea is to approximate the belief state using particles, using the particule filter algorithm for calculating beliefs. Particule filters were already extensively discussed in various chapters of this book. They were mathematically derived in Chapter 4.2.1.

16.4.1 Monte Carlo Backups

For the sake of completeness, let briefly review the basic update equations. Initially, N random samples are drawn from the initial belief distribution $b^{(0)}$. Then, at time t, a new set of weighted particles is generated.

Table 16.3 shows a variant of the particle filter algorithm that accepts an action a, an observation o', and a belief b as input. It then transforms the belief b into a new belief. This algorithm implements the function $B(b, a, o')$, assuming that belief states are represented by weighted sets of particles. It is trivially obtained from the algorithm **particle_filter** in Table 4.3 on page 4.3 of this book. See Chapter 4.2.1 for a more detailed explanation.

Armed with $B(b, a, o')$, the update equation can be implemented using a straight-foward Monte Carlo algorihm. Recall that value function in belief space is defined by

1:       **Algorithm particle_filter_2(b, a, o'):**

2:           $b' = \emptyset$

3:           *do N times:*
4:               *sample $\langle s, p \rangle$ from b according to $p_1, \ldots, p_N$ in b*
5:               *sample $s' \sim P(s'|a, s)$*
6:               *$w' = P(o'|s')$*
7:               *add $\langle s', w' \rangle$ to b'*

8:           *normalize all weights $w' \in b'$*
9:           *return b'*

Table 16.3   A variant of the particle filter algorithm, which accepts an action a and an observation o' as input.

the following recursive equation:

$$C^T(b) \;=\; \max_a \int \left(\left[C^{T-1}(B(b, a, o')) + \int c(s')\, B(b, a, o')(s')\, ds' \right] \right. \\ \left. \int P(o'|s') \int P(s'|a, s)\, b(s)\, ds\, ds' \right) do' \qquad (16.77)$$

Inner projection loop:

1:       **Algorithm MCPOMDP_inner_loop(b):**

2:           $C^T(b) = -\infty$
3:           *for all a do*
4:               $C^T(b, a) = 0$
5:               *do N times*
6:                   *Sample $s \sim b$*
7:                   *Sample $s' \sim P(s'|a, s)$*

8:                      *Sample $o' \sim P(o'|s')$*
9:                      $b' = $ **particle_filter_2**(b, a, o')
10:                     $C^T(b) = C^T(b) + \frac{1}{N}\gamma[C^{T-1}(b') + c(s')]$
11:                 *if $C^T(b, a) > C^T(b)$*
12:                     $C^T(b) = C^T(b, a)$
13:             *return $C^T(b)$*

Outer McPOMDP loop:

1:          **Algorithm MCPOMDP_outer_loop(b):**

2:              *for all b do*
3:                  $\hat{C}(b) = 0$

4:              *repeat until convergence*
5:                  *for all b*
6:                      $\hat{C}(b) = $ **MCPOMDP_inner_loop**(b)
7:                  *return $\hat{C}$*

1:          **Algorithm MCPOMDP_alterative_outer_loop(b):**

2:              *for all b do*
3:                  $\hat{C}(b) = 0$

4:              *repeat until convergence*
5:                  $b = P(s_0)$
6:                  $s \sim P(s_0)$

7:                  *do until trial over*
8:                      $\hat{C}(b) = $ **MCPOMDP_inner_loop**(b)
9:                      *if* **rand**$(0, 1) > 0.1$
10:                         $a = \text{argmax}_a \, \hat{C}(b, a)$
11:                     *else*
12:                         *select a at random*

13:                    *draw* $s' \sim P(s'|a, s)$

14:                    $s = s'$

15:            *return* $\hat{C}$

16.4.1.1   Learning Value Functions

Following the rich literature on reinforcement learning [16, 38], our approach solves the POMDP problem by value iteration in belief space. More specifically, our approach recursively learns a value function Q over belief states and action, by *backing up* values from subsequent belief states:

$$Q(\theta_t, a_t) \quad \longleftarrow \quad E\left[R(o_{t+1}) + \gamma \max_{\bar{a}} Q(\theta_{t+1}, \bar{a})\right] \tag{16.78}$$

Leaving open (for a moment) how Q is represented, it is easy to be seen how the algorithm **particle_projection** can be applied to compute a Monte Carlo approximation of the right hand-side expression: Given a belief state θ_t and an action a_t, **particle_projection** computes a sample of $R(o_{t+1})$ and θ_{t+1}, from which the expected value on the right hand side of (16.78) can be approximated.

It has been shown [3] that if both sides of (16.78) are equal, the *greedy* policy

$$\sigma^Q(\theta) \quad = \quad \operatorname*{argmax}_{\bar{a}} Q(\theta, \bar{a}) \tag{16.79}$$

is *optimal*, i.e., $\sigma^* = \sigma^Q$. Furthermore, it has been shown (for the discrete case!) that repetitive application of (16.78) leads to an optimal value function and, thus, to the optimal policy [44, 8].

Our approach essentially performs model-based reinforcement learning in belief space using approximate sample-based representations. This makes it possible to apply a rich bag of tricks found in the literature on MDPs. In our experiments below, we use on-line reinforcement learning with counter-based exploration and experience replay [21] to determine the order in which belief states are updated.

16.4.1.2   Nearest Neighbor

We now return to the issue how to represent Q. Since we are operating in real-valued spaces, some sort of function approximation method is called for. However, recall

(a)
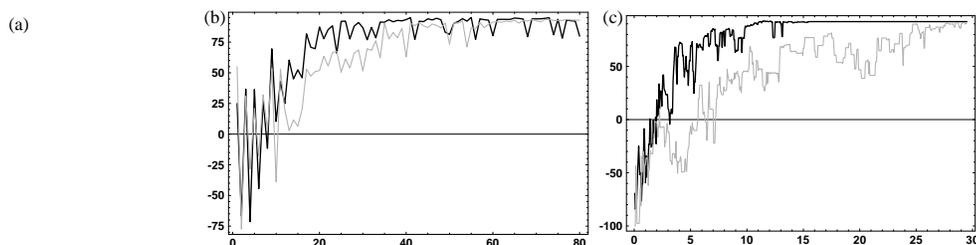
Figure 16.3 (a) The environment, schematically. (b) Average performance (reward) as a function of training episodes. The black graph corresponds to the smaller environment (25 steps min), the grey graph to the larger environment (50 steps min). (c) Same results, plotted as a function of number of backups (in thousands).

that Q accepts a probability distribution (a sample set) as an input. This makes most existing function approximators (e.g., neural networks) inapplicable.

In our current implementation, nearest neighbor [24] is applied to represent Q. More specifically, our algorithm maintains a set of sample sets θ (belief states) annotated by an action a and a Q-value $Q(\theta, a)$. When a new belief state θ' is encountered, its Q-value is obtained by finding the k nearest neighbors in the database, and linearly averaging their Q-values. If there aren't sufficiently many neighbors (within a pre-specified maximum distance), θ' is added to the database; hence, the database grows over time.

Our approach uses KL divergence (relative entropy) as a distance function[1]. Technically, the KL-divergence between two continuous distributions is well-defined. When applied to sample sets, however, it cannot be computed. Hence, when evaluating the distance between two different sample sets, our approach maps them into continuous-valued densities using Gaussian kernels, and uses Monte Carlo sampling to approximate the KL divergence between them. This algorithm is fairly generic an extension of nearest neighbors to function approximation in density space, where densities are represented by samples. Space limitations preclude us from providing further detail (see [24, 33]).

16.4.2   Experimental Results

Preliminary results have been obtained in a world shown in two domains, one synthetic and one using a simulator of a RWI B21 robot.

---

[1] Strictly speaking, KL divergence is not a distance metric, but this is ignored here.
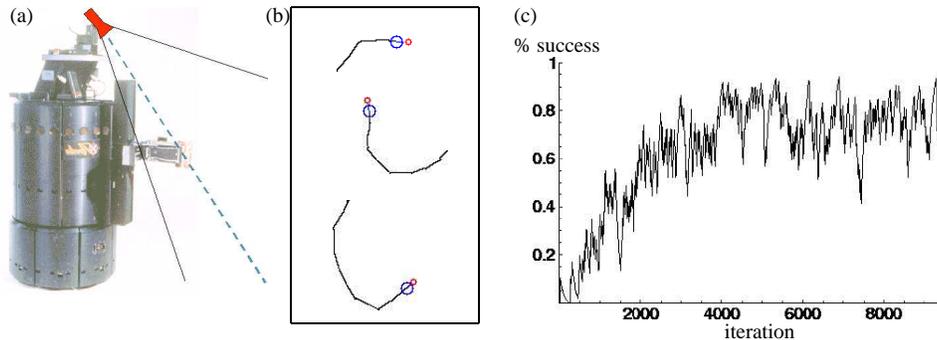
Figure 16.4 Find and fetch task: (a) The mobile robot with gripper and camera, holding the target object (experiments are carried out in simulation!), (b) three successful runs (trajectory projected into 2D), and (c) success rate as a function of number of planning steps.

In the synthetic environment (Figure 16.3a), the agents starts at the lower left corner. Its objective is to reach "heaven" which is either at the upper left corner or the lower right corner. The opposite location is "hell." The agent does not know the location of heaven, but it can ask a "priest" who is located in the upper right corner. Thus, an optimal solution requires the agent to go first to the priest, and then head to heaven. The state space contains a real-valued (coordinates of the agent) and discrete (location of heaven) component. Both are unobservable: In addition to not knowing the location of heaven, the agent also cannot sense its (real-valued) coordinates. 5% random motion noise is injected at each move. When an agent hits a boundary, it is penalized, but it is also told which boundary it hit (which makes it possible to infer its coordinates along one axis). However, notice that the *initial* coordinates of the agent are known.

The optimal solution takes approximately 25 steps; thus, a successful POMDP planner must be capable of looking 25 steps ahead. We will use the term "successful policy" to refer to a policy that always leads to heaven, even if the path is suboptimal. For a policy to be successful, the agent must have learned to first move to the priest (information gathering), and then proceed to the right target location.

Figures 16.3b&c show performance results, averaged over 13 experiments. The solid (black) curve in both diagrams plots the average cumulative reward J as a function of the number of training episodes (Figure 16.3b), and as a function of the number of backups (Figure 16.3c). A successful policy was consistently found after 17 episodes (or 6,150 backups), in all 13 experiments. In our current implementation, 6,150 backups require approximately 29 minutes on a Pentium PC. In some experiments, a successful policy was identified in 6 episodes (less than 1,500 backups or 7 minutes). After a successful policy is found, further learning gradually optimizes the path. To

investigate scaling, we doubled the size of the environment (quadrupling the size of the state space), making the optimal solution 50 steps long. The results are depicted by the gray curves in Figures 16.3b&c. Here a successful policy is consistently found after 33 episodes (10,250 backups, 58 minutes). In some runs, a successful policy is identified after only 14 episodes.

We also applied MC-POMDPs to a robotic *locate-and-retrieve task*. Here a robot (Figure 16.4a) is to find and grasp an object somewhere in its vicinity (at floor *or* table height). The robot's task is to grasp the object using its gripper. It is rewarded for successfully grasping the object, and penalized for unsuccessful grasps or for moving too far away from the object. The state space is continuous in x and y coordinates, and discrete in the object's height.

The robot uses a mono-camera system for object detection; hence, viewing the object from a single location is insufficient for its 3D localization. Moreover, initially the object might not be in sight of the robot's camera, so that the robot must look around first. In our simulation, we assume 30% general detection error (false-positive and false-negative), with additional Gaussian noise if the object is detected correctly. The robot's actions include turns (by a variable angle), translations (by a variable distance), and grasps (at one of two legal heights). Robot control is erroneous with a variance of 20% (in x-y-space) and 5% (in rotational space). Typical belief states range from uniformly distributed sample sets (initial belief) to samples narrowly focused on a specific x-y-z location.

Figure 16.4c shows the rate of successful grasps as a function of iterations (actions). While initially, the robot fails to grasp the object, after approximately 4,000 iterations its performance surpasses 80%. Here the planning time is in the order of 2 hours. However, the robot fails to reach 100%. This is in part because certain initial configurations make it impossible to succeed (e.g., when the object is too close to the maximum allowed distance), in part because the robot occasionally misses the object by a few centimeters. Figure 16.4b depicts three successful example trajectories. In all three, the robot initially searches the object, then moves towards it and grasps it successfully.

16.5 AUGMENTED MARKOV DECISION PROCESSES

So far, we have studied two frameworks and algorithms for action selection under uncertainty: MDP and POMDP algorithms. Both frameworks accommodate non-

deterministic action outcomes. Only the POMDP algorithms can cope with uncertainty in perception, whereas MDP algorithms assume that the state is fully observable. On the other hand, MDP algorithms are polynomial in the number of states, whereas POMDP algorithms are doubly exponential.

Both MDP and POMDP are in fact extreme ends of a spectrum of possible probabilistic algorithms. MDP algorithms ignore the issue of uncertainty in a robot's belief entirely. In contrast, POMDPs consider all possible belief functions, even though many of them might be extremely unlikely in practice. This raises the question of the middle ground: Are there probabilistic algorithms that can accommodate some of the robot's uncertainty, yet are computationally more efficient than the full POMDP solution?

The answer is yes. In many robotics domains, the range of belief distributions the robot might encounter is a small subspace of the space of all belief distributions that can be defined over the state space. This insight enables us to devise probabilistic planning algorithms that are significantly more efficient than solutions to the full POMDP problem, which still coping with robot beliefs in adequate ways.

16.5.1   The Augmented State Space

The augmented MDP algorithm (AMDP) applies to situations where the belief state can be summarized by a lower-dimensional statistic. Values and actions are calculated from this statistic, instead of the full belief b. The smaller the statistic, the more efficient the resulting algorithm.

In many situations a good choice of the statistic is the tuple

$$\bar{b} \;=\; \langle \operatorname*{argmax}_{s} b(s); H[b] \rangle \tag{16.80}$$

where $\operatorname{argmax}_s b(s)$ is the most likely state under the belief distribution b, and

$$H[b] \;=\; -\int b(s) \ln b(s) \, ds \tag{16.81}$$

is the *entropy* of the belief distribution. Calculating a value function parameterized on $\bar{b}$, instead of b, is similar to the MDP approach assuming that the most likely state is the actual state of the world. It differs, however, by a one-dimensional statistic that

characterizes the amount of uncertainty in the state estimate. The entire uncertainty—usually an infinitely dimensional quantity—is thus summarized by a one-dimensional quantity: the entropy. The representation is mathematically justified if $\bar{b}$ is a sufficient statistic of b with regards to the estimation of value, that is:

$$C(b) \quad = \quad C(\bar{b}) \tag{16.82}$$

for all b the robot may encounter. In practice, this assumption will rarely hold true. However, the resulting value function might still be good enough for a sensible choice of action. Alternatively, one might consider different statistics, such as moments of the belief distribution (mean, variance, ...), modes, and so on.

16.5.2   Value Iteration in AMDPs

Value iteration can easily be applied to the augmented MDP model. Let f be the function that extracts the statistic from b, that is

$$\bar{b} \quad = \quad f(b) \tag{16.83}$$

for arbitrary beliefs b. Then the POMDP value iteration equation (16.76) can be rewritten as

$$\hat{C}(f(b)) \quad = \quad \max_a \int \left[\hat{C}(f(B(b, a, o'))) + \int c(s') \; B(b, a, o')(s') \; ds' \right]$$
$$\int P(o'|s') \int P(s'|a, s) \; b(s) \; ds \; ds' \; do' \tag{16.84}$$

This suggests an algorithm similar to the one in Table 16.2, replacing the value function $\hat{C}$ by the concatenated function $\hat{C}f$. This algorithm would still loop over all belief states b, but the estimation of the value function would be considerably faster due to the fact that many belief b share the same statistic $f(b)$.

Here we will formulate this algorithm similar to the algorithm **MDP_value_iteration** in Table 15.1.

1:        **Algorithm Augmented_MDP_value_iteration():**

2:            *for all* $\bar{b}$
3:                $\hat{C}(\bar{b}) = 0$

4:            *repeat until convergence*
5:                *for all* $\bar{b}$
6:                    $\hat{C}(\bar{b}) = \max_a \int [c(\bar{b}') + \hat{C}(\bar{b}')] \ P(\bar{b}'|a, \bar{b}); d\bar{b}'$

7:            *return* $\hat{C}$

Table 16.4 The value iteration algorithm for augmented MDPs with finite state and action spaces.

In analogy to the derivation in the Section on POMDPs, Section 16.3.1, we can calculate $P(\bar{b}'|a, \bar{b})$ as follows:

$$
\begin{aligned}
P(\bar{b}'|a, \bar{b}) &= \int P(\bar{b}'|a, b) \ P(b|\bar{b}) \ db & (16.85) \\
&= \int \int I_{f(b)=\bar{b}} \ P(\bar{b}'|o', a, b) \ P(o'|a, b) \ do \ db \\
&= \int \int \int I_{f(b)=\bar{b}} \ P(\bar{b}'|o', a, b) \ P(o'|s') \ P(s'|a, b) \ ds' \ do \ db \\
&= \int \int \int I_{f(b)=\bar{b}} \ P(\bar{b}'|o', a, b) \ P(o'|s') \ P(s'|a, s) \ P(s|b) \ ds \ ds' \ do \ db \\
&= \int \int \int I_{f(b)=\bar{b}} \ I_{f(B(o',a,b))=\bar{b}'} \ P(o'|s') \ P(s'|a, s) \ P(s|b) \ ds \ ds' \ do \ db
\end{aligned}
$$

The function I is the indicator function whose value is 1 iff its condition is true. The resulting conditional probability $P(\bar{b}'|a, \bar{b})$ is the motion model, formulated in the space defined by $\bar{b}$.

Similarly, the expected costs can be expressed as a function of $\bar{b}$ as follows:

$$
c(\bar{b}) = \int I_{f(b)=\bar{b}} \ c(b) \ db
$$

$$= \int \int I_{f(b)=\bar{b}} \, c(s); \underline{(s)} \, ds \, db \qquad (16.86)$$

The resulting value iteration algorithm for the augmented MDP is shown in Table 16.4. Notice that this algorithm is essentially the same as the one derived for MDPs. It is also highly related, though less obvious, to the general POMDP value iteration algorithm in Table 16.2. Lines 2 and 3 in Table 16.4 initialize the value function, defined over the space of all belief statistics $\bar{b}$. Lines 4 through 6 calculate the value function, using the probability $P(\bar{b}'|a, \bar{b})$ defined above.

The resulting algorithm still appears inefficient, due to the difficulty of computing $P(\bar{b}'|a, \bar{b})$.

[reword the following]

We have employ two strategies for speeding up the calculation. Both are approximate. First, we have replaced the exhaustive integration in Equations (16.85) and (16.86) by a stochastic integration, in which the belief b, the states s and s' and the observation o' are drawn at random from the corresponding distributions. Second, the model $P(\bar{b}'|a, \bar{b})$ was cached using a look-up table, avoiding computing the same probability twice. This leads to an explicit preprocessing phase where the familiar probabilistic models $P(o|s)$ and $P(s'|s, a)$ are 'translated' into the augmented state model $P(\bar{b}'|a, \bar{b})$, for which the subsequent application of value iteration is then straightforward.

16.5.3   Illustration

The augmented state model considers the robot's uncertainty when executing actions. In particular, it characterizes the increase and decrease of certainty upon moving and sensing. This enables the robot to anticipate and avoid situations with increased danger of loosing critical state information.

In the context of mobile robot navigation, the augmented MDP algorithm anticipates the increase and decrease of uncertainty in a robot's pose estimate. For example, a robot traversing a large featureless area is likely to gradually loose information as to where it is. This is reflected in the conditional probability $P(\bar{b}'|a, \bar{b})$, which with high likelihood increases the entropy of the belief in such areas. In areas populated with localization features, e.g., near the walls, the uncertainty is more likely to decrease. Planners using the augmented MDP can anticipate such situations, and generate policies that minimize the time of arrival while simultaneously maximize the certainty at
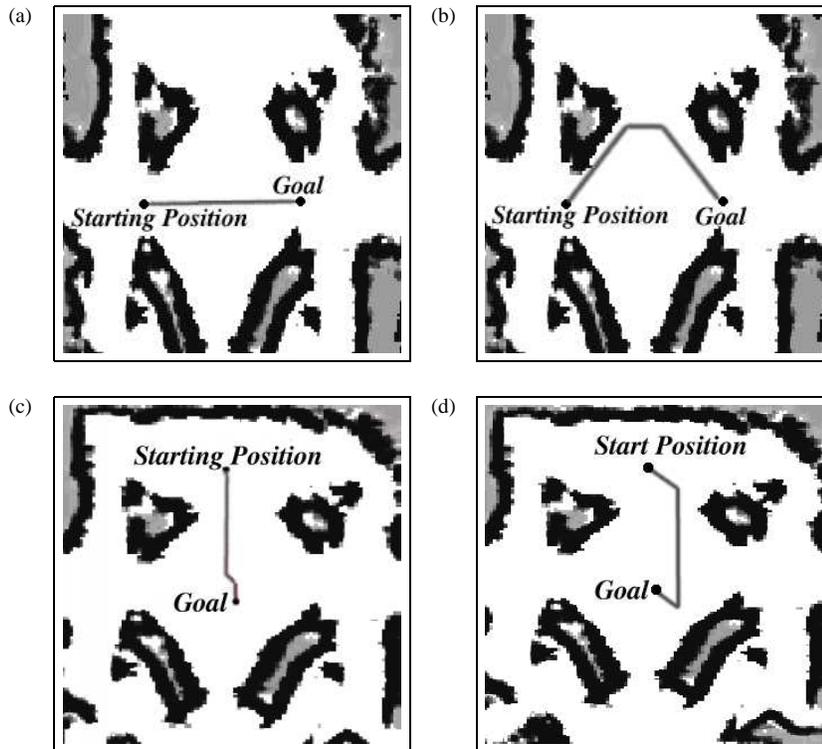
Figure 16.5 Examples of robot paths in a large, open environment, for two different configurations (top row and bottom row). The diagrams (a) and (c) are paths generates by a conventional dynamic programming path planner that ignores the robot's perceptual uncertainty. The diagrams (b) and (d) are obtained using the augmented MDP planner, which anticipates uncertainty and avoids regions where the robot is more likely to get lost. In analogy to ships who often stay close to the shore for orientation, the augmented MDP planner is known as 'coastal planner.'

the time of arrival at a goal location. Since the uncertainty is an estimate of the true positioning error, it is a good measure for the chances of actually arriving at the desired location.

In the context of mobile robot navigation, the augmented MDP algorithm is also known as *coastal navigation.* This name indicates the resemblance to ships, which often stay close to the coastline so as to not loose track of their location (unless, of course, the ship is equipped with a global navigation system).
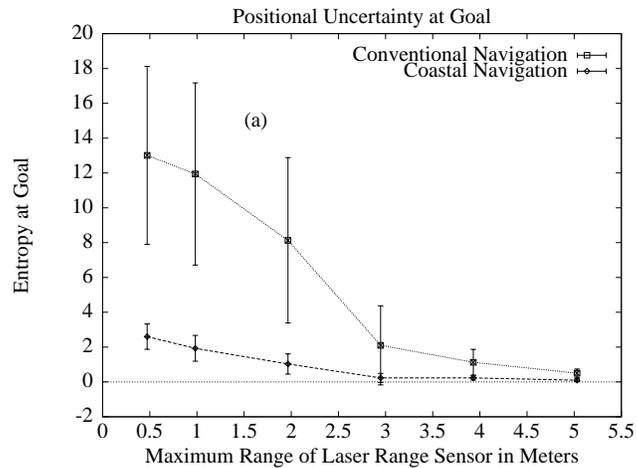
Figure 16.6   Performance comparison of MDP planning and Augmented MDP planning. Shown here is the uncertainty (entropy) at the goal location as a function of the sensor range.

Figure 16.5 shows example trajectories for two constellations (two different start and goal locations). The diagrams on the left correspond to a MDP planner, which does not consider the robot's uncertainty. The augmented MDP planner generates trajectories like the ones shown on the right. In Figures 16.5a&b, the robot is requested to move through a large open area in a museum (approximately 40 meters wide). he MDP algorithm, not aware of the increased risk of getting list in the open area, generates a policy that corresponds to the shortest path form the start to the goal location. The coastal planner, in contrast, generates a policy that stays close to the obstacles, where the robot has an increased chance of receiving informative sensor measurements at the expense of an increased travel time. Similarly, Figure 16.5c&d considers a situation where the goal location is close to the center of the featureless, open area. Here the coastal planner recognizes that passing by known objects reduces the pose uncertainty, increasing the chances of successfully arriving at the goal location.

Figure 16.6 shows a performance comparison between the coastal navigation strategy and the MDP approach. In particular, it depicts the entropy of the robot's belief b at the goal location, as a function of the sensor characteristics. In this graph, the maximum perceptual range is varied, to study the effect of impoverished sensors. As the graph suggests, the coastal approach has significantly higher chances of success. The difference is largest if the sensors are very poor. For sensors that have a long range, the difference ultimately disappears. The latter does not come as a surprise, since with good range sensors the amount of information that can be perceived is less dependent on the specific pose of the robot.

16.6   SUMMARY

In this section, we introduced three major algorithms for action selection under uncertainty.

- The Markov Decision Process (MDP) framework addresses the problem of action selection in situations where the outcomes of actions are stochastic, but the robot knows its state with absolute certainty.

- Value iteration solves discrete MDPs by computing a value function over states. Selecting actions by greedily maximizing value leads to optimal action choices. A key characteristic of value functions is that they induce policies for action selection that are defined over the entire state space. No matter what the outcome of an action is, the robot knows what to do.

- The more general framework of Partially Observable Markov Decision Processes (POMDPs) addresses the question of action selection with imperfect sensors. POMDP algorithms have to base decisions on belief states. Solutions trade off information gathering (exploration) and exploitation.

- Value iteration can also be applied to solve POMDPs. In worlds with finitely many states, actions, and observations, the value function is piecewise linear in the parameters of the belief space. However, in the worst case calculating the value function is doubly exponential in the planning horizon.

- Finally, we introduced the Augmented Markov Decision Process (AMDP) framework, which blends MDPs and POMDPs. AMDPs represent the robot's uncertainty by a low-dimensional statistic (e.g., the entropy), opening the door to efficient planning algorithms that consider some of the uncertainty in state estimation.

- Value iteration was also applied to AMDPS. Within mobile robot navigation, the resulting algorithm is known as coastal navigation, since it navigates robots in a way that minimizes the anticipated uncertainty in localization.

16.7   BIBLIOGRAPHICAL REMARKS

16.8   PROJECTS

1. ???

REFERENCES

[1] I. Asimov. *Runaround*. Faucett Crest, New York, 1942.

[2] I. Asimov. *I, Robot*. Doubleday, 1950.

[3] R. E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

[4] J. Borenstein, B. Everett, and L. Feng. *Navigating Mobile Robots: Systems and Techniques*. A. K. Peters, Ltd., Wellesley, MA, 1996.

[5] H. Choset. *Sensor Based Motion Planning: The Hierarchical Generalized Voronoi Graph*. PhD thesis, California Institute of Technology, 1996.

[6] E. Chown, S. Kaplan, and D. Kortenkamp. Prototypes, location, and associative networks (plan): Towards a unified theory of cognitive mapping. *Cognitive Science*, 19:1–51, 1995.

[7] M. Csorba. *Simultaneous Localization and Map Building*. PhD thesis, Department of Engineering Science, University of Oxford, Oxford, UK, 1997.

[8] P. Dayan and T. J. Sejnowski. TD(λ) converges with probability 1. February 1993.

[9] M. Deans and M. Hebert. Invariant filtering for simultaneous localization and mapping. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1042–1047, San Francisco, CA, 2000. IEEE.

[10] M. Garland and P. Heckbert. Surface simplification using quadric error metrics. In *Proceedings of SIGGRAPH*, 1997.

[11] J. Guivant and E. Nebot. Optimization of the simultaneous localization and map building algorithm for real time implementation. *IEEE Transactions of Robotics and Automation*, May 2001. In press.

[12] J.-S. Gutmann and K. Konolige. Incremental mapping of large cyclic environments. In *Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA)*, 2000.

[13] D. Hähnel, D. Fox, W. Burgard, and S. Thrun. A highly efficient FastSLAM algorithm for generating cyclic maps of large-scale environments from raw laser range measurements. In *Proceedings of the Conference on Intelligent Robots and Systems (IROS)*, 2003.

[14] T. Hofmann, J. Puzicha, and M. Jordan. Learning from dyadic data. In *Advances in Neural Information Processing Systems 11 (NIPS)*, Cambridge, MA, 1999. MIT Press.

[15] S.J. Julier and J. K. Uhlmann. Building a million beacon map. In *Proceedings of the SPIE Sensor Fusion and Decentralized Control in Robotic Systems IV, Vol. #4571*, 2000.

[16] L.P. Kaelbling, M.L. Littman, and A.W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4, 1996.

[17] B. Kuipers and Y.-T. Byun. A robust qualitative method for spatial learning in unknown environments. In *Proceeding of Eighth National Conference on Artificial Intelligence AAAI-88*, Menlo Park, Cambridge, 1988. AAAI Press / The MIT Press.

[18] B. Kuipers and Y.-T. Byun. A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. *Journal of Robotics and Autonomous Systems*, 8:47–63, 1991.

[19] J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, MA, 1991.

[20] J.J. Leonard and H.J.S. Feder. A computationally efficient method for large-scale concurrent mapping and localization. In J. Hollerbach and D. Koditschek, editors, *Proceedings of the Ninth International Symposium on Robotics Research*, Salt Lake City, Utah, 1999.

[21] L.-J. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8, 1992.

[22] M. J. Matarić. A distributed model for mobile robot environment-learning and navigation. Master's thesis, MIT, Cambridge, MA, January 1990. also available as MIT Artificial Intelligence Laboratory Tech Report AITR-1228.

[23] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI)*, Acapulco, Mexico, 2003. IJCAI.

[24] A.W. Moore, C.G. Atkeson, and S.A. Schaal. Locally weighted learning for control. *AI Review*, 11:75–113, 1997.

[25] H. P. Moravec. Sensor fusion in certainty grids for mobile robots. *AI Magazine*, 9(2):61–74, 1988.

[26] K. Murphy. Bayesian map learning in dynamic environments. In *Advances in Neural Information Processing Systems (NIPS)*. MIT Press, 2000.

[27] J. Neira, J.D. Tardós, and J.A. Castellanos. Linear time vehicle relocation in SLAM. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2003.

[28] E. Nettleton, S. Thrun, and H. Durrant-Whyte. A constant time communications algorithm for decentralised SLAM. Submitted for publication, 2002.

[29] E. Nettleton, S. Thrun, and H. Durrant-Whyte. Decentralised slam with low-bandwidth communication for teams of airborne vehicles. In *Proceedings of the International Conference on Field and Service Robotics*, Lake Yamanaka, Japan, 2003.

[30] E.W. Nettleton, P.W. Gibbens, and H.F. Durrant-Whyte. Closed form solutions to the multiple platform simultaneous localisation and map building (slam) problem. In Bulur V. Dasarathy, editor, *Sensor Fusion: Architectures, Algorithms, and Applications IV*, volume 4051, pages 428–437, Bellingham, 2000.

[31] P. Newman. *On the Structure and Solution of the Simultaneous Localisation and Map Building Problem*. PhD thesis, Australian Centre for Field Robotics, University of Sydney, Sydney, Australia, 2000.

[32] N. J. Nilsson. *Principles of Artificial Intelligence*. Springer Publisher, Berlin, New York, 1982.

[33] D. Ormoneit and S. Sen. Kernel-based reinforcement learning. Technical Report 1999-8, Department of Statistics, Stanford University, 1999.

[34] K. Rose. Deterministic annealing for clustering, compression, classification, regression, and related optimization problems. *Proceedings of IEEE*, D, November 1998.

[35] H Shatkay and L. Kaelbling. Learning topological maps with weak local odometric information. In *Proceedings of IJCAI-97*. IJCAI, Inc., 1997.

[36] R.C. Smith and P. Cheeseman. On the representation and estimation of spatial uncertainty. *International Journal of Robotics Research*, 5(4):56–68, 1986.

[37] B. Stewart, J. Ko, D. Fox, and K. Konolige. A hierarchical bayesian approach to mobile robot map structure estimation. In *Proceedings of the Conference on Uncertainty in AI (UAI)*, Acapulco, Mexico, 2003.

[38] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[39] J.D. Tardós, J. Neira, P.M. Newman, and J.J. Leonard. Robust mapping and localization in indoor environments using sonar data. *Int. J. Robotics Research*, 21(4):311–330, April 2002.

[40] S. Thrun, D. Hähnel, D. Ferguson, M. Montemerlo, R. Triebel, W. Burgard, C. Baker, Z. Omohundro, S. Thayer, and W. Whittaker. A system for volumetric robotic mapping of abandoned mines. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2003.

[41] S. Thrun and Y. Liu. Multi-robot SLAM with sparse extended information filers. In *Proceedings of the 11th International Symposium of Robotics Research (ISRR'03)*, Sienna, Italy, 2003. Springer.

[42] K. Čapek. *R.U.R. (Rossum's Universal Robots)*. (out of print), 1921.

[43] C.-C. Wang, C. Thorpe, and S. Thrun. Online simultaneous localization and mapping with detection and tracking of moving objects: Theory and results from a ground vehicle in crowded urban areas. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2003.

[44] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, England, 1989.