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:

$$\begin{pmatrix} x \\ y \\ \theta \end{pmatrix} \tag{5.1}$$

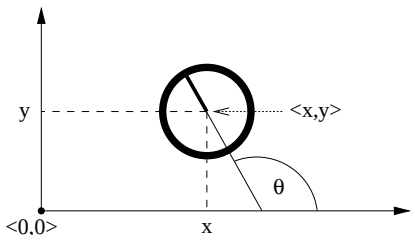


Figure 5.1 Robot pose, shown in a global coordinate system.

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

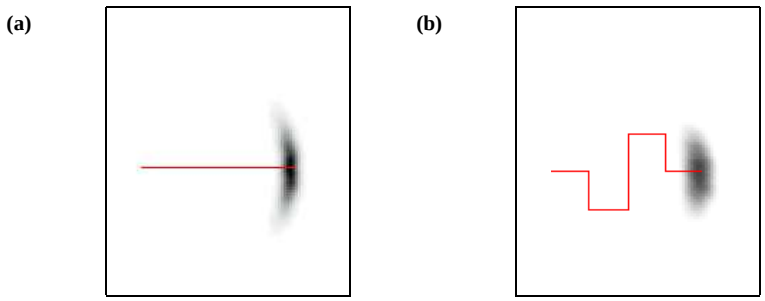


Figure 5.2 The motion model: Posterior distributions of the robot's pose upon executing the motion command illustrated by the solid line. The darker a location, the more likely it is. This plot has been projected into 2D. The original density is three-dimensional, taking the robot's heading direction 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 synchro-drive, 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 $\hat{\omega}$.

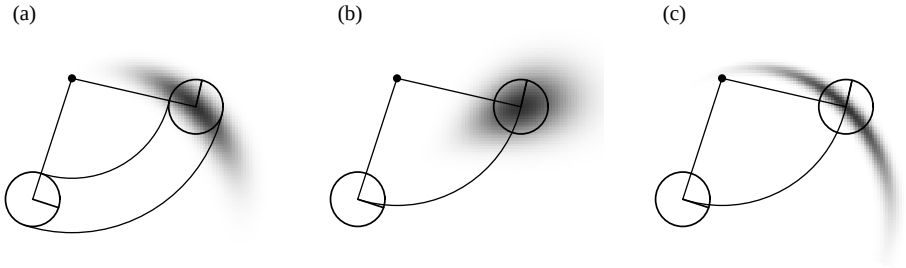


Figure 5.3 The velocity motion model, for different noise parameter settings.

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

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

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

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

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

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

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

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

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

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 **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: $\hat{v} = v + \mathbf{sample}(\alpha_1|v| + \alpha_2|\omega|)$
3: $\hat{\omega} = \omega + \mathbf{sample}(\alpha_3|v| + \alpha_4|\omega|)$
4: $\hat{\gamma} = \mathbf{sample}(\alpha_5|v| + \alpha_6|\omega|)$
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* $\frac{b}{6} \sum_{i=1}^{12} \mathbf{rand}(-1, 1)$

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

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

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 $[x, y]$.

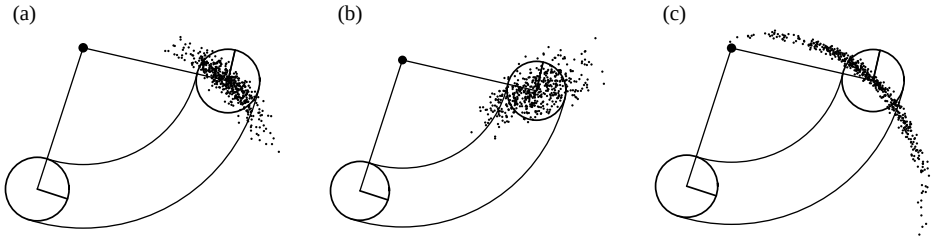


Figure 5.4 Sampling from the velocity motion model, using the same parameters as in Figure 5.3. Each diagram shows 500 samples.

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 retro-guessing 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

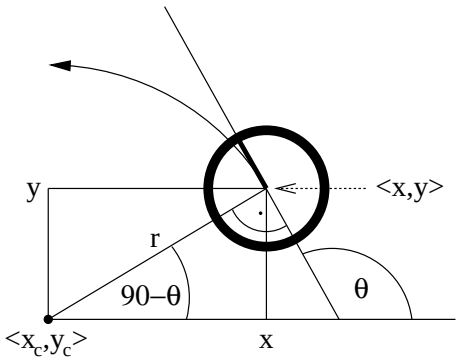


Figure 5.5 Motion carried out by a noise-free robot moving with constant velocities v and ω and starting at $(x \;\; y \;\; \theta)^T$.

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.

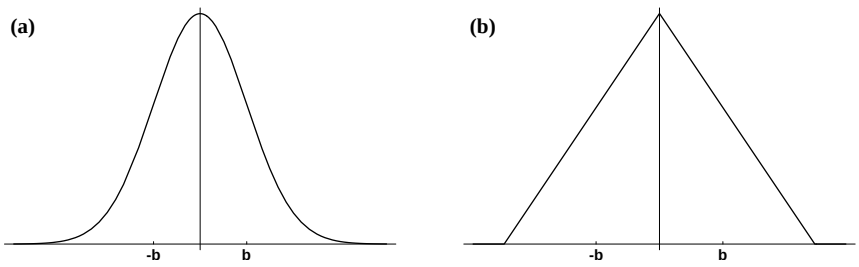


Figure 5.6 Probability density functions with variance b: (a) Normal distribution, (b) triangular distribution.

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

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

  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

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

  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

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

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

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

with

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

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:

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

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

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

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

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

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

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

Here $\mathrm{atan2}$ 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):

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

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

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

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

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

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).

$$v_{\text{err}} = v - \hat{v} \tag{5.26}$$
$$\omega_{\text{err}} = \omega - \hat{\omega} \tag{5.27}$$
$$\gamma_{\text{err}} = \hat{\gamma} \tag{5.28}$$

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

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

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:

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

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

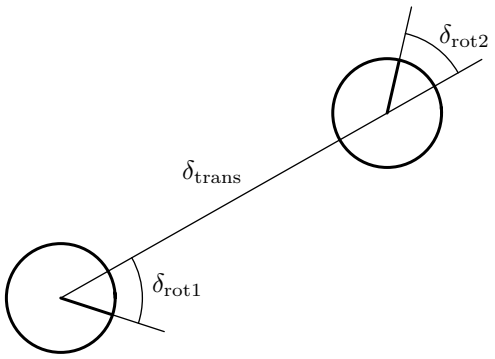


Figure 5.7 Odometry model: The robot motion in the time interval $(t-1, t]$ is approximated by a rotation δ_{rot1}, followed by a translation δ_{trans} and a second rotation δ_{rot2}. The turns and translation are noisy.

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!

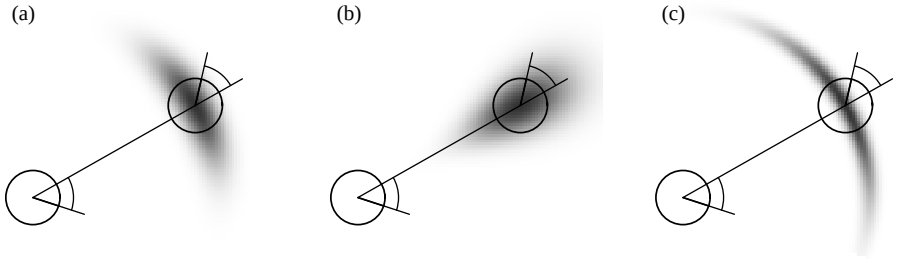


Figure 5.8 The odometry motion model, for different noise parameter settings.

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

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

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 $(\delta_{\text{rot1}} \;\; \delta_{\text{trans}} \;\; \delta_{\text{rot2}})^T$, and these parameters are sufficient to reconstruct the relative motion between $\bar{s}$ and $\bar{s}'$. Thus, $\delta_{\text{rot1}}, \delta_{\text{trans}}, \delta_{\text{rot2}}$ 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 $(\delta_{\text{rot1}} \ \ \delta_{\text{trans}} \ \ \delta_{\text{rot2}})^T$ 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 **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

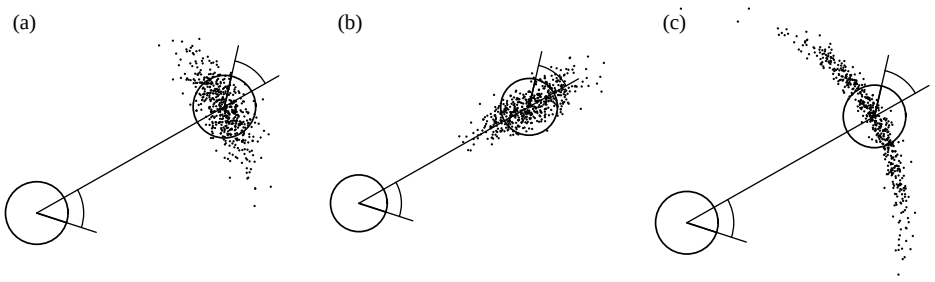


Figure 5.9 Sampling from the odometry motion model, using the same parameters as in Figure 5.8. Each diagram shows 500 samples.

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-

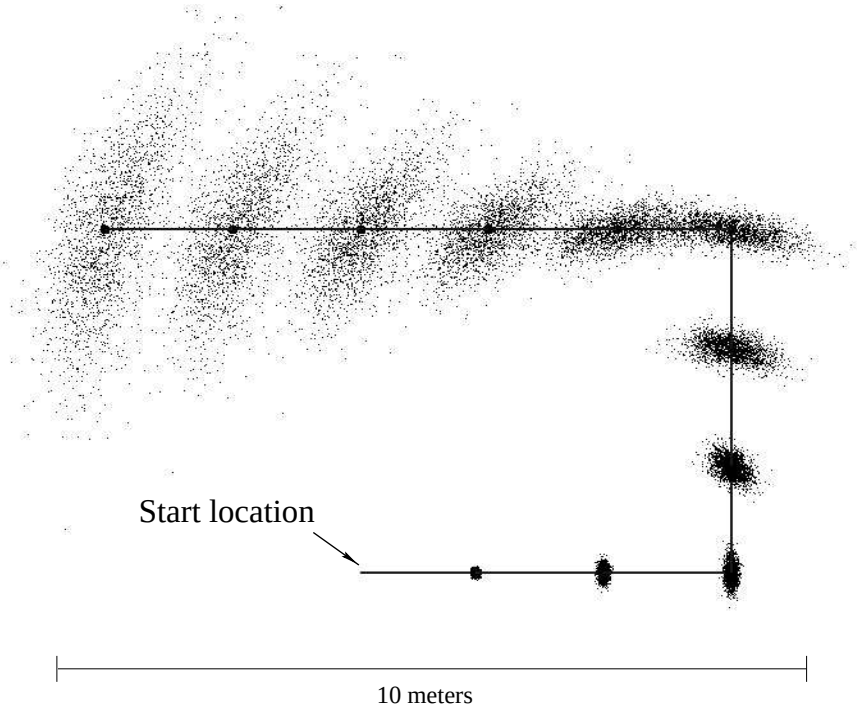


Figure 5.10 Sampling approximation of the position belief for a non-sensing robot. The solid line displays the actions, and the samples represent the robot's belief at different points in time.

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 $\bar{x}_t = (\bar{x}'\ \ \bar{y}'\ \ \bar{\theta}')$:

$$\delta_{\text{rot1}} = \text{atan2}(\bar{y}' - \bar{y}, \bar{x}' - \bar{x}) - \bar{\theta} \tag{5.34}$$

$$\delta_{\text{trans}} = \sqrt{(\bar{x} - \bar{x}')^2 + (\bar{y} - \bar{y}')^s} \tag{5.35}$$

$$\delta_{\text{rot2}} = \bar{\theta}' - \bar{\theta} - \delta_{\text{rot1}} \tag{5.36}$$

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:

$$\hat{\delta}_{\text{rot1}} = \delta_{\text{rot1}} - \varepsilon_{\alpha_1|\delta_{\text{rot1}}|+\alpha_2|\delta_{\text{trans}}|} \tag{5.37}$$

$$\hat{\delta}_{\text{trans}} = \delta_{\text{trans}} - \varepsilon_{\alpha_3\ |\delta_{\text{trans}}|+\alpha_4|\delta_{\text{rot1}}+\delta_{\text{rot2}}|} \tag{5.38}$$

$$\hat{\delta}_{\text{rot2}} = \delta_{\text{rot2}} - \varepsilon_{\alpha_1|\delta_{\text{rot2}}|+\alpha_2|\delta_{\text{trans}}|} \tag{5.39}$$

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,

$$\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} \hat{\delta}_{\text{trans}}\ \cos(\theta + \hat{\delta}_{\text{rot1}}) \\ \hat{\delta}_{\text{trans}}\ \sin(\theta + \hat{\delta}_{\text{rot1}}) \\ \theta + \hat{\delta}_{\text{rot1}} + \hat{\delta}_{\text{rot2}} \end{pmatrix} \tag{5.40}$$

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:

$$p(x_t \mid u_t, x_{t-1}, m) \quad = \quad \eta \, p(m \mid x_t) \, p(x_t \mid u_t, x_{t-1})$$

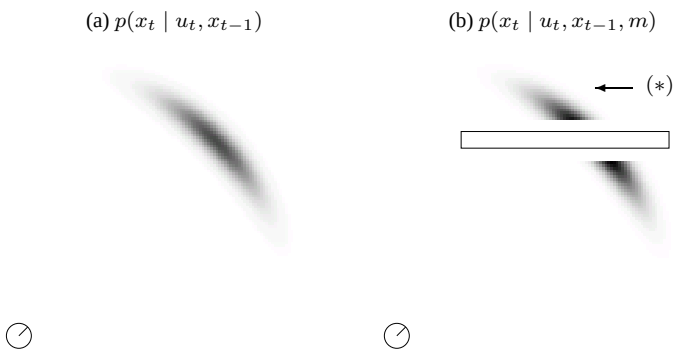


Figure 5.11 Velocity motion model (a) without a map and (b) conditioned on a map m.

$$= \quad \eta \, p(x_t \mid m) \, p(x_t \mid u_t, x_{t-1}) \tag{5.50}$$

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

$$p(m \mid x_t, u_t, x_{t-1}) \quad = \quad p(m \mid x_t) \tag{5.51}$$

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

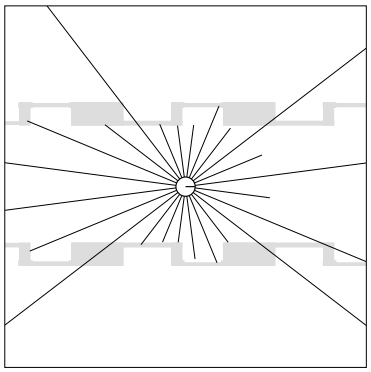


Figure 6.1 Typical ultrasound scan of a robot in its environment.

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:

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

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

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

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:

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

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

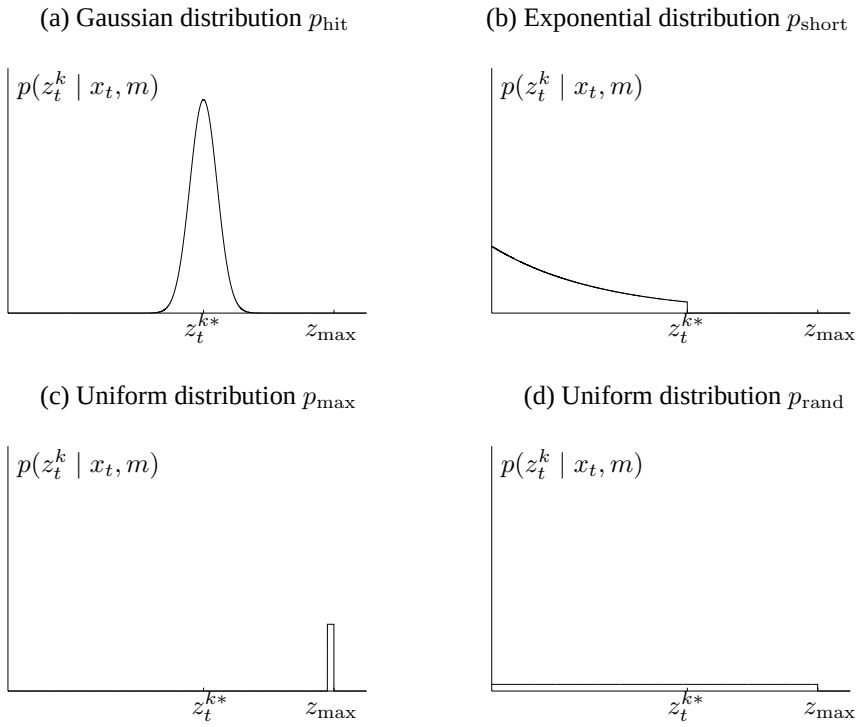


Figure 6.2 Components of the range finder sensor model. In each diagram the horizontal axis corresponds to the measurement z_t^k, the vertical to the likelihood.

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

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

where z_t^{k*} is calculated from x_t and m via ray 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:

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

The normalizer η evaluates to

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

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

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

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

$$\int_0^{z_t^{k*}} \lambda_{\text{short}} \, e^{-\lambda_{\text{short}} z_t^k} \, dz_t^k \quad = \quad -e^{-\lambda_{\text{short}} z_t^{k*}} + e^{-\lambda_{\text{short}} 0} \tag{6.8}$$

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

the value of η can be derived as:

$$\eta \quad = \quad \frac{1}{1 - e^{-\lambda_{\text{short}} z_t^{k*}}} \tag{6.10}$$

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 $z_{\max}$:

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

   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).

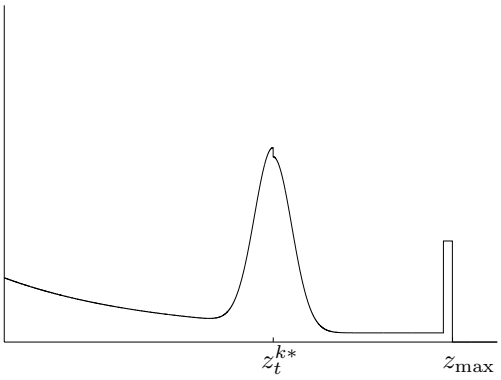


Figure 6.3 "Pseudo-density" of a typical mixture distribution $p(z_t^k \mid x_t, m)$.

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 $[0; z_{\max}]$:

$$p_{\text{rand}}(z_t^k \mid x_t, m) = \begin{cases} \frac{1}{z_{\max}} & \text{if } 0 \leq z_t^k < z_{\max} \\ 0 & \text{otherwise} \end{cases} \tag{6.12}$$

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 $z_{\text{hit}} + z_{\text{short}} + z_{\max} + z_{\text{rand}} = 1$.

$$p(z_t^k \mid x_t, m) = \begin{pmatrix} z_{\text{hit}} \\ z_{\text{short}} \\ z_{\max} \\ z_{\text{rand}} \end{pmatrix}^T \cdot \begin{pmatrix} p_{\text{hit}}(z_t^k \mid x_t, m) \\ p_{\text{short}}(z_t^k \mid x_t, m) \\ p_{\max}(z_t^k \mid x_t, m) \\ p_{\text{rand}}(z_t^k \mid x_t, m) \end{pmatrix} \tag{6.13}$$

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

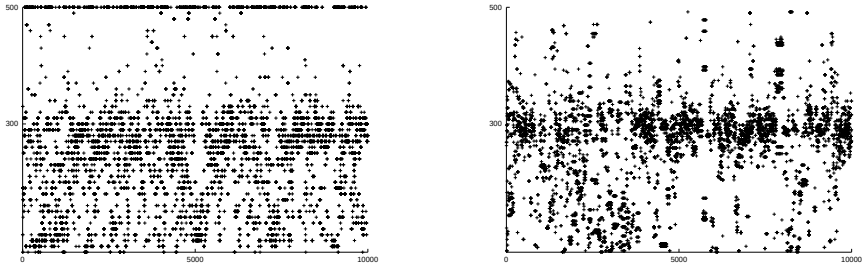


Figure 6.4 Typical data obtained with (a) a sonar sensor and (b) a laser-range sensor in an office environment for a "true" range of 300 cm and a maximum range of 500 cm.

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. Algorithms that maximize the likelihood of data are known as *maximum likelihood* estimators, 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* algorithm, 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: $\lambda_{\text{short}} = \frac{\sum_i e_{i,\text{short}}}{\sum_i e_{i,\text{short}}\, z_i}$
16: return $\Theta = \{z_{\text{hit}}, z_{\text{short}}, z_{\text{max}}, z_{\text{rand}}, \sigma_{\text{hit}}, \lambda_{\text{short}}\}$

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

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.

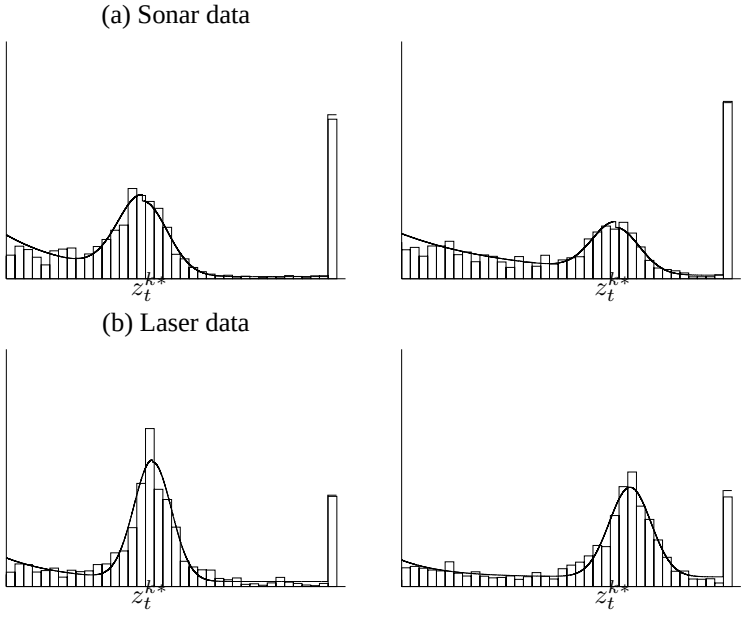
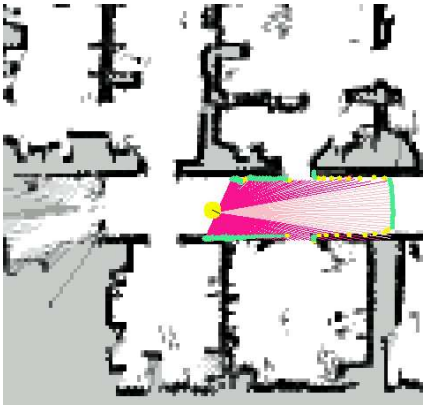


Figure 6.5 Approximation of the beam model based on (a) sonar data and (b) laser range data. The sensor models depicted on the left were obtained by a maximum likelihood approximation to the data sets depicted in Figure 6.4.

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

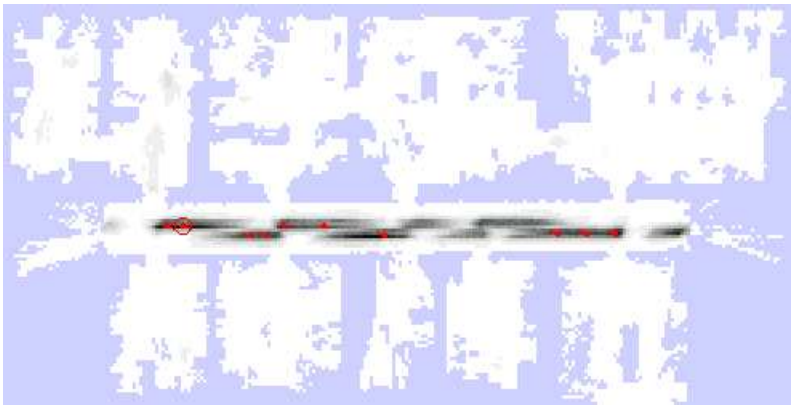


Figure 6.6 Probabilistic model of perception: (a) Laser range scan, projected into a previously acquired map m. (b) The likelihood $p(z_t \mid x_t, m)$, evaluated for all positions x_t and projected into the map (shown in gray). The darker a position, the larger $p(z_t \mid x_t, m)$.

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 classical 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:

$$\frac{\partial \log p(Z_{\text{hit}} \mid X, m, \Theta)}{\partial \sigma_{\text{hit}}} = -\frac{|Z_{\text{hit}}|}{\sigma_{\text{hit}}} + \frac{1}{\sigma_{\text{hit}}^3} \sum_{z_i \in Z_{\text{hit}}} (z_i - z_i^*)^2 \quad (6.19)$$

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

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

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

$$\frac{\partial \log p(Z_{\text{short}} \mid X, m, \Theta)}{\partial \lambda_{\text{short}}} = \frac{|Z_{\text{short}}|}{\lambda_{\text{short}}} - \sum_{z_i \in Z_{\text{short}}} z_i \tag{6.23}$$

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

$$\lambda_{\text{short}} = \frac{|Z_{\text{short}}|}{\sum_{z_i \in Z_{\text{short}}} z_i} \tag{6.24}$$

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:

$$\begin{aligned}&\log p(Z \mid X, m, \Theta) \\ &\quad = \sum_{z_i \in Z} \log p(z_i \mid x_i, m, \Theta) \\ &\quad = \sum_{z_i \in Z_{\text{hit}}} \log p_{\text{hit}}(z_i \mid x_i, m) + \sum_{z_i \in Z_{\text{short}}} \log p_{\text{short}}(z_i \mid x_i, m) \\ &\qquad + \sum_{z_i \in Z_{\text{max}}} \log p_{\text{max}}(z_i \mid x_i, m) + \sum_{z_i \in Z_{\text{rand}}} \log p_{\text{rand}}(z_i \mid x_i, m)\end{aligned} \tag{6.25}$$

This expression can be rewritten using the variables c_i:

$$\begin{aligned}\log p(Z \mid X, m, \Theta) &= \sum_{z_i \in Z} I(c_i = \text{hit})\ \log p_{\text{hit}}(z_i \mid x_i, m) \\ &\quad + I(c_i = \text{short})\ \log p_{\text{short}}(z_i \mid x_i, m) \\ &\quad + I(c_i = \text{max})\ \log p_{\text{max}}(z_i \mid x_i, m) \\ &\quad + I(c_i = \text{rand})\ \log p_{\text{rand}}(z_i \mid x_i, m) \end{aligned} \tag{6.26}$$

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:

$$\begin{aligned} &E[\log p(Z \mid X, m, \Theta)] \\ &= \sum_i p(c_i = \text{hit})\ \log p_{\text{hit}}(z_i \mid x_i, m) + p(c_i = \text{short})\ \log p_{\text{short}}(z_i \mid x_i, m) \\ &\quad + p(c_i = \text{max})\ \log p_{\text{max}}(z_i \mid x_i, m) + p(c_i = \text{rand})\ \log p_{\text{rand}}(z_i \mid x_i, m) \\ &=: \sum_i e_{i,\text{hit}}\ \log p_{\text{hit}}(z_i \mid x_i, m) + e_{i,\text{short}}\ \log p_{\text{short}}(z_i \mid x_i, m) \\ &\quad + e_{i,\text{max}}\ \log p_{\text{max}}(z_i \mid x_i, m) + e_{i,\text{rand}}\ \log p_{\text{rand}}(z_i \mid x_i, m) \end{aligned} \tag{6.27}$$

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

$$\begin{aligned}\eta &= [\, p_{\text{hit}}(z_i \mid x_i, m) + p_{\text{short}}(z_i \mid x_i, m) \\ &\quad + p_{\text{max}}(z_i \mid x_i, m) + p_{\text{rand}}(z_i \mid x_i, m)\,]^{-1} \end{aligned} \tag{6.29}$$

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.

$$\begin{pmatrix} x_{z_t^k} \\ y_{z_t^k} \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x_{k,\text{sens}} \\ y_{k,\text{sens}} \end{pmatrix} + z_t^k \begin{pmatrix} \cos(\theta + \theta_{k,\text{sens}}) \\ \sin(\theta + \theta_{k,\text{sens}}) \end{pmatrix} \tag{6.33}$$

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.

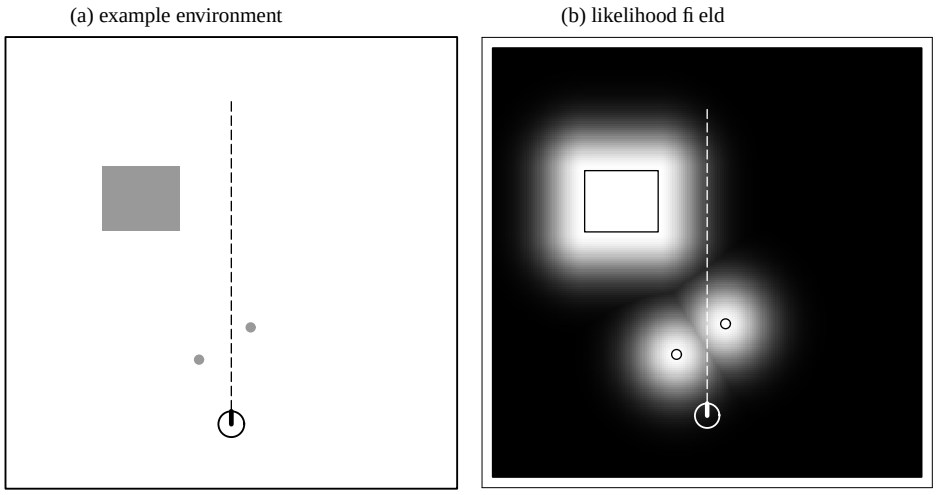


Figure 6.7 (a) Example environment with three obstacles (gray). The robot is located towards the bottom of the figure, and takes a measurement z_t^k as indicated by the dashed line. (b) Likelihood field for this obstacle configuration: the darker a location, the less likely it is to perceive an obstacle 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:

$$p_{\text{hit}}(z_t^k \mid x_t, m) \quad = \quad \varepsilon_{\sigma_{\text{hit}}^2}(dist^2) \tag{6.34}$$

   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.

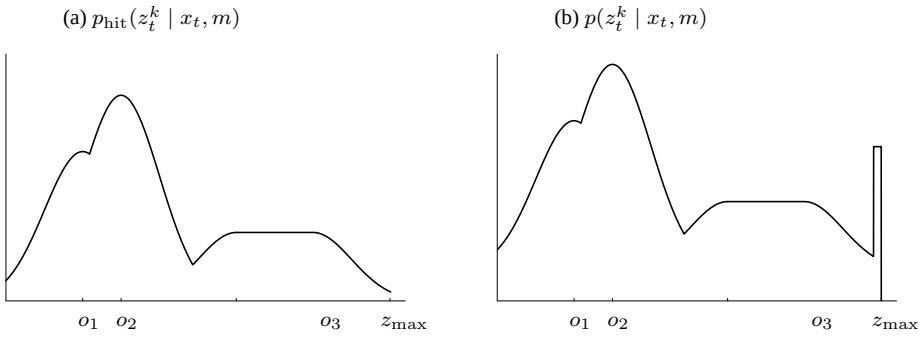


Figure 6.8 (a) Probability $p_{\text{hit}}(z_t^k)$ as a function of the measurement z_t^k, for the situation depicted in Figure 6.7. Here the sensor beam passes by three obstacles, with respective nearest points o_1, o_2, and o_3. (b) Sensor probability $p(z_t^k \mid x_t, m)$, obtained for the situation 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:

$$z_{\text{hit}} \cdot p_{\text{hit}} + z_{\text{rand}} \cdot p_{\text{rand}} + z_{\max} \cdot p_{\max} \tag{6.35}$$

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: $dist^2 = \min_{x',y'} \left\{ (x_{z_t^k} - x')^2 + (y_{z_t^k} - y')^2 \;\middle|\; \langle x', y' \rangle \text{ occupied in } m \right\}$

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 **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 **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 $p(z_t^k \mid$

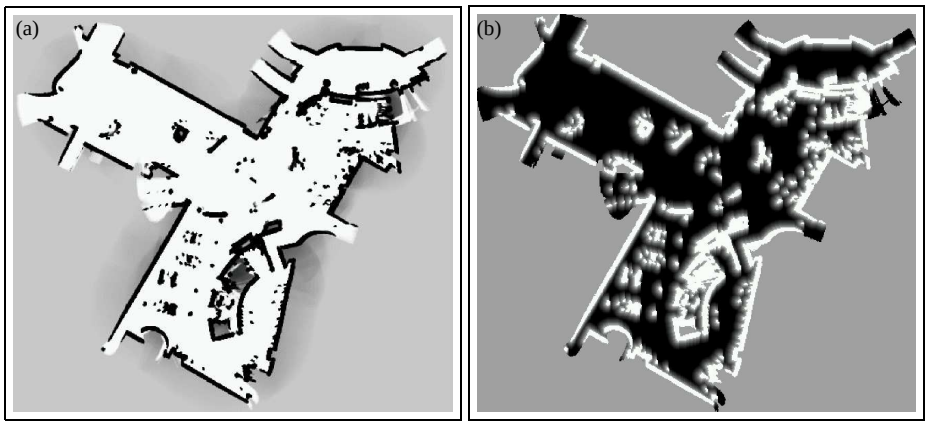


Figure 6.9 (a) Occupancy grid map of the San Jose Tech Museum, (b) pre-processed likelihood field.

$x_t, m)$. 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.

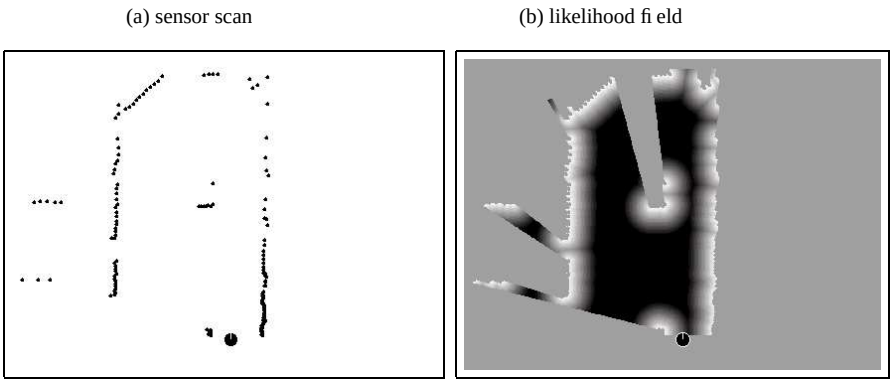


Figure 6.10 (a) Sensor scan, from a bird's eye perspective. The robot is placed at the bottom of this figure, generating a proximity scan that consists of the 180 dots in front of the robot. (b) Likelihood function generated from this sensor scan. The darker a region, the smaller the likelihood for sensing 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

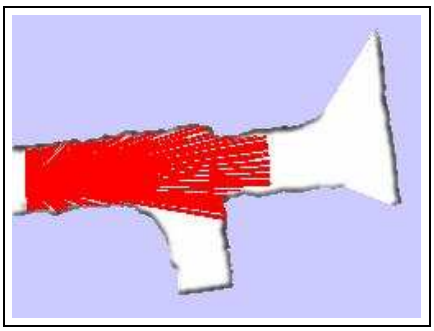


Figure 6.11 Example of a local map generated from 10 range scans, one of which is shown.

that the more similar m and m_{local}, the larger $p(m_{\text{local}} \mid x_t, m)$. Since the local map is represented relative to the robot location, this comparison requires that the cells of the local map are transformed into the coordinate framework of the global map. Such a transformation can be done similar to the coordinate transform (6.33) of sensor measurements used in the likelihood field model. If the robot is at location x_t, we denote by $m_{x,y,\text{local}}(x_t)$ the grid cell in the local map that corresponds to $(x \;\; y)^T$ in global coordinates. Once both maps are in the same reference frame, they can be compared using the map correlation function, which is defined as follows:

$$\rho_{m,m_{\text{local}},x_t} = \frac{\sum_{x,y}(m_{x,y} - \bar{m}) \cdot (m_{x,y,\text{local}}(x_t) - \bar{m})}{\sqrt{\sum_{x,y}(m_{x,y} - \bar{m})^2 \sum_{x,y}(m_{x,y,\text{local}}(x_t) - \bar{m})^2}} \tag{6.36}$$

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

$$\bar{m} = \frac{1}{2N} \sum_{x,y}(m_{x,y} + m_{x,y,\text{local}}), \tag{6.37}$$

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

$$p(m_{\text{local}} \mid x_t, m) = \max\{\rho_{m,m_{\text{local}},x_t}, 0\} \tag{6.38}$$

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 $x_t = (x \;\; y \;\; \theta)^T$.

$$\begin{pmatrix} r_t^i \\ \phi_t^i \\ s_t^i \end{pmatrix} = \begin{pmatrix} \sqrt{(m_{j,x} - x)^2 + (m_{j,y} - y)^2} \\ \operatorname{atan2}(m_{j,y} - y, m_{j,x} - x) - \theta \\ s_j \end{pmatrix} + \begin{pmatrix} \varepsilon_{\sigma_r^2} \\ \varepsilon_{\sigma_\phi^2} \\ \varepsilon_{\sigma_s^2} \end{pmatrix} \tag{6.41}$$

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

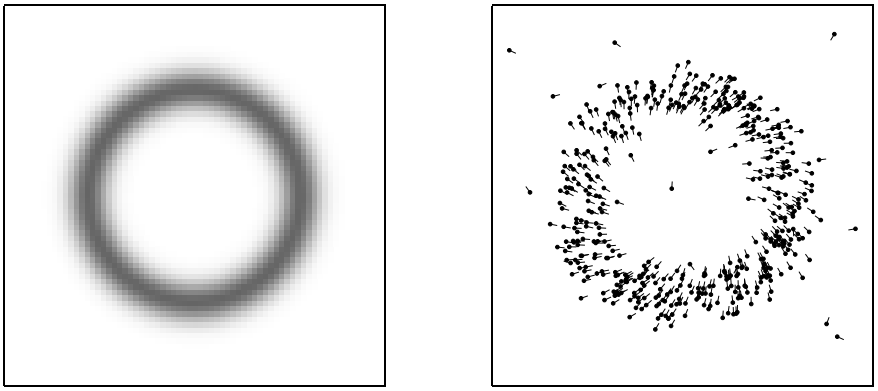


Figure 6.12 Landmark detection model: (a) Posterior distribution of the robot's pose given that it detected a landmark in 5m distance and 30deg relative bearing (projected onto 2D). (b) Sample robot poses generated from such a detection. The lines indicate the orientation of the poses.

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.