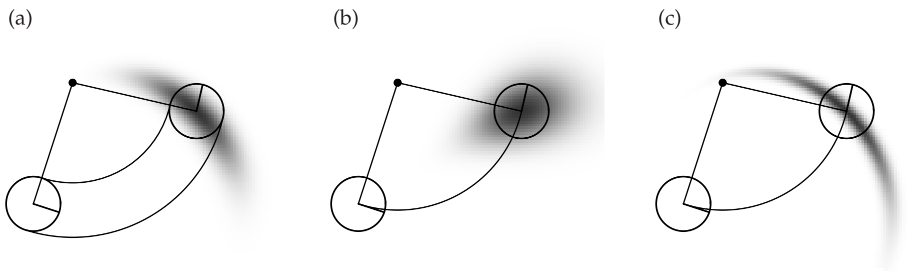

Figure 5.3 The velocity motion model, for different noise parameter settings.

The function **prob**(x, b^2) models the motion error. It computes the probability of its parameter x under a zero-centered random variable with variance b^2. Two possible implementations are shown in Table 5.2, for error variables with normal distribution and triangular distribution, respectively.

Figure 5.3 shows graphical examples of the 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 (c.f. Chapter 4.3), 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

```
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 prob($v - \hat{v}, \alpha_1 v^2 + \alpha_2 \omega^2$) · prob($\omega - \hat{\omega}, \alpha_3 v^2 + \alpha_4 \omega^2$)
                · prob($\hat{\gamma}, \alpha_5 v^2 + \alpha_6 \omega^2$)
```

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 $(x \quad y \quad \theta)^T$; x_t is represented by $(x' \quad y' \quad \theta')^T$; and u_t is represented by the velocity vector $(v \quad \omega)^T$. The function **prob**(a, b^2) computes the probability of its argument a under a zero-centered distribution with variance b^2. It may be implemented using any of the algorithms in Table 5.2.

```
1:      Algorithm prob_normal_distribution($a$, $b^2$):
2:          return $\frac{1}{\sqrt{2\pi\, b^2}} \exp\left\{-\frac{1}{2}\frac{a^2}{b^2}\right\}$

3:      Algorithm prob_triangular_distribution($a$, $b^2$):
4:          return $\max\left\{0, \frac{1}{\sqrt{6}\, b} - \frac{|a|}{6\, b^2}\right\}$
```

Table 5.2 Algorithms for computing densities of a zero-centered normal distribution and a triangular distribution with variance b^2.

```
1:  Algorithm sample_motion_model_velocity($u_t, x_{t-1}$):
2:      $\hat{v} = v + \textbf{sample}(\alpha_1 v^2 + \alpha_2 \omega^2)$
3:      $\hat{\omega} = \omega + \textbf{sample}(\alpha_3 v^2 + \alpha_4 \omega^2)$
4:      $\hat{\gamma} = \textbf{sample}(\alpha_5 v^2 + \alpha_6 \omega^2)$
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^2) generates a random sample from a zero-centered distribution with variance b^2. It may, for example, be implemented using the algorithms in Table 5.4.

```
1:  Algorithm sample_normal_distribution($b^2$):
2:      return $\frac{1}{2} \sum_{i=1}^{12} \textbf{rand}(-b, b)$

3:  Algorithm sample_triangular_distribution($b^2$):
4:      return $\frac{\sqrt{6}}{2} [\textbf{rand}(-b, b) + \textbf{rand}(-b, b)]$
```

Table 5.4 Algorithm for sampling from (approximate) normal and triangular distributions with zero mean and variance b^2; see Winkler (1995: p293). The function **rand**(x, y) is assumed to be a pseudo random number generator with uniform distribution in $[x, y]$.

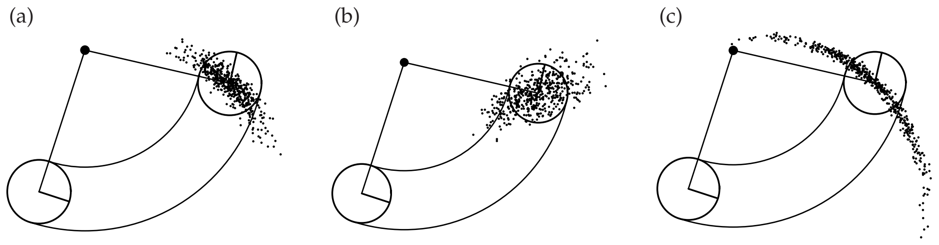

Figure 5.4 Sampling from the velocity motion model, using the same parameters as in Figure 5.3. Each diagram shows 500 samples.

new pose, in lines 5 through 7. Thus, the sampling procedure 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 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 us 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 of the Velocity Motion Model

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 Chapter 5.4 (page 132). 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

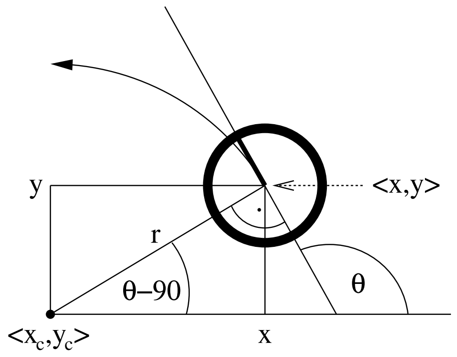

Figure 5.5 Motion carried out by a noise-free robot moving with constant velocities v and ω and starting at $(x\ \ y\ \ \theta)^T$.

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 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 its 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

$$\begin{pmatrix} \hat{v} \\ \hat{\omega} \end{pmatrix} = \begin{pmatrix} v \\ \omega \end{pmatrix} + \begin{pmatrix} \varepsilon_{\alpha_1 v^2 + \alpha_2 \omega^2} \\ \varepsilon_{\alpha_3 v^2 + \alpha_4 \omega^2} \end{pmatrix} \tag{5.10}$$

Here ε_{b^2} is a zero-mean error variable with variance b^2. Thus, the true velocity equals the commanded velocity plus some small, additive error (noise). In our model, the standard deviation 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.

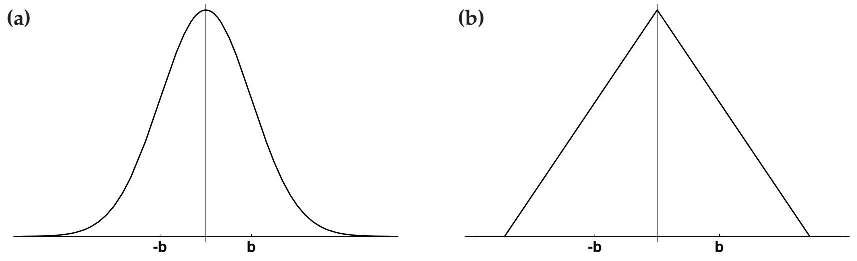

Figure 5.6 Probability density functions with variance b^2: (a) Normal distribution, (b) triangular distribution.

Two common choices for the error ε_{b^2} are the normal and the triangular distribution.

NORMAL DISTRIBUTION

The *normal distribution* with zero mean and variance b^2 is given by the density function

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

Figure 5.6a shows the density function of a normal distribution with variance b^2. Normal distributions are commonly used to model noise in continuous stochastic processes. Its support, which is the set of points a with $p(a) > 0$, is $\Re$.

TRIANGULAR DISTRIBUTION

The density of a *triangular distribution* with zero mean and variance b^2 is given by

$$\varepsilon_{b^2}(a) \quad = \quad \max\left\{0, \frac{1}{\sqrt{6}\, b} - \frac{|a|}{6\, b^2}\right\} \tag{5.12}$$

which is non-zero only in $(-\sqrt{6}b; \sqrt{6}b)$. 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

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

This equation is obtained by substituting the commanded velocity $u_t = (v \;\; \omega)^T$ with the noisy motion $(\hat{v} \;\; \hat{\omega})^T$ 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

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

with

(5.15) $$\hat{\gamma} = \varepsilon_{\alpha_5 v^2 + \alpha_6 \omega^2}$$

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:

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

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 almost all 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 travels with 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

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

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

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

and hence

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

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

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

Furthermore, we can now calculate the change of heading direction

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

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):

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

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

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

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

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

The rotational velocity $\hat{\gamma}$ needed to achieve the final heading θ' of the robot in $(x'y')$ within Δt can be determined according to (5.14) as:

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

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

(5.26)
$$v_{\mathrm{err}} \quad = \quad v - \hat{v}$$

(5.27)
$$\omega_{\mathrm{err}} \quad = \quad \omega - \hat{\omega}$$

(5.28)
$$\gamma_{\mathrm{err}} \quad = \quad \hat{\gamma}$$

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

(5.29)
$$\varepsilon_{\alpha_1 v^2 + \alpha_2 \omega^2}(v_{\mathrm{err}})$$

(5.30)
$$\varepsilon_{\alpha_3 v^2 + \alpha_4 \omega^2}(\omega_{\mathrm{err}})$$

(5.31)
$$\varepsilon_{\alpha_5 v^2 + \alpha_6 \omega^2}(\gamma_{\mathrm{err}})$$

where ε_{b^2} denotes a zero-mean error variable with variance b^2, 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:

(5.32)
$$p(x_t \mid u_t, x_{t-1}) \quad = \quad \varepsilon_{\alpha_1 v^2 + \alpha_2 \omega^2}(v_{\mathrm{err}}) \cdot \varepsilon_{\alpha_3 v^2 + \alpha_4 \omega^2}(\omega_{\mathrm{err}}) \cdot \varepsilon_{\alpha_5 v^2 + \alpha_6 \omega^2}(\gamma_{\mathrm{err}})$$

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 $p(x' \mid u, x)$

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 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 encoder information; most commercial robots make such integrated pose estimation available in periodic time intervals (e.g., every tenth of a second). This leads to a second motion model discussed in this chapter, the *odometry motion model*. The odometry motion model uses odometry measurements in lieu of controls.

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 fil-

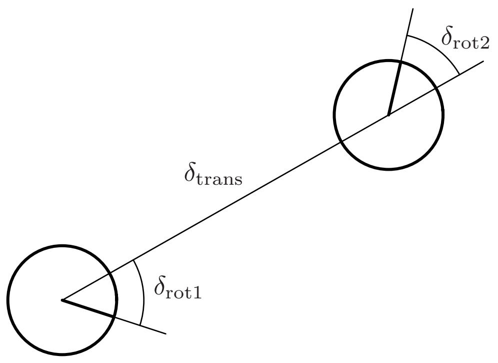

Figure 5.7 Odometry model: The robot motion in the time interval $(t-1, t]$ is approximated by a rotation δ_{rot1}, followed by a translation δ_{trans} and a second rotation δ_{rot2}. The turns and translations are noisy.

ter algorithms, such as the localization and mapping algorithms discussed in later chapters. But it makes this information unusable for accurate motion planning and control.

5.4.1 Closed Form Calculation

Technically, odometric information 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 consider odometry data as if it were control signals. In this section, we will treat odometry measurements just like 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!

The odometry model uses the *relative motion information*, as measured by 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 $\bar{x}_{t-1} = (\bar{x} \;\; \bar{y} \;\; \bar{\theta})^T$ to $\bar{x}_t = (\bar{x}' \;\; \bar{y}' \;\; \bar{\theta}')^T$. Here the

```
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:      $\delta_{\text{trans}} = \sqrt{(\bar{x} - \bar{x}')^2 + (\bar{y} - \bar{y}')^2}$
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:      $p_1 = \textbf{prob}(\delta_{\text{rot1}} - \hat{\delta}_{\text{rot1}}, \alpha_1 \hat{\delta}^2_{\text{rot1}} + \alpha_2 \hat{\delta}^2_{\text{trans}})$
9:      $p_2 = \textbf{prob}(\delta_{\text{trans}} - \hat{\delta}_{\text{trans}}, \alpha_3 \hat{\delta}^2_{\text{trans}} + \alpha_4 \hat{\delta}^2_{\text{rot1}} + \alpha_4 \hat{\delta}^2_{\text{rot2}})$
10:     $p_3 = \textbf{prob}(\delta_{\text{rot2}} - \hat{\delta}_{\text{rot2}}, \alpha_1 \hat{\delta}^2_{\text{rot2}} + \alpha_2 \hat{\delta}^2_{\text{trans}})$

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 given by $(\bar{x}_{t-1} \;\; \bar{x}_t)^T$, with $\bar{x}_{t-1} = (\bar{x} \;\; \bar{y} \;\; \bar{\theta})$ and $\bar{x}_t = (\bar{x}' \;\; \bar{y}' \;\; \bar{\theta}')$.

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

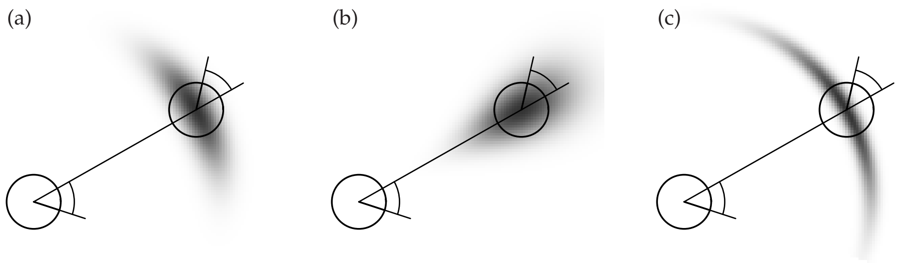

Figure 5.8 The odometry motion model, for different noise parameter settings.

$(\delta_{\text{rot1}} \;\; \delta_{\text{trans}} \;\; \delta_{\text{rot2}})^T$, and these parameters are sufficient to reconstruct the relative motion between $\bar{s}$ and $\bar{s}'$. Thus, $\delta_{\text{rot1}}, \delta_{\text{trans}}, \delta_{\text{rot2}}$ form together a sufficient statistics of the relative motion encoded by the odometry.

The probabilistic 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 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 $p(x_t \mid u_t, x_{t-1})$.

Lines 2 to 4 in Table 5.5 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^2) implements an error distribution over a with zero mean and variance b^2. 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 be difficult to debug. Finally, line 11 returns the combined error probability, obtained by multiplying the individual error probabilities p_1, p_2, and p_3. This last step

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

5:       $\hat{\delta}_{\text{rot1}} = \delta_{\text{rot1}} - \textbf{sample}(\alpha_1 \delta_{\text{rot1}}^2 + \alpha_2 \delta_{\text{trans}}^2)$
6:       $\hat{\delta}_{\text{trans}} = \delta_{\text{trans}} - \textbf{sample}(\alpha_3 \delta_{\text{trans}}^2 + \alpha_4 \delta_{\text{rot1}}^2 + \alpha_4 \delta_{\text{rot2}}^2)$
7:       $\hat{\delta}_{\text{rot2}} = \delta_{\text{rot2}} - \textbf{sample}(\alpha_1 \delta_{\text{rot2}}^2 + \alpha_2 \delta_{\text{trans}}^2)$

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

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 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 122. The smaller the time between two 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.

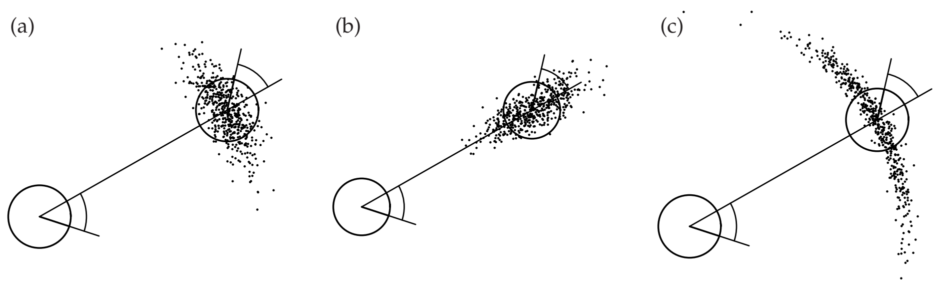

Figure 5.9 Sampling from the odometry motion model, using the same parameters as in Figure 5.8. Each diagram shows 500 samples.

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 (Chapter 4.3) 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 implement 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 large space.

5.4.3 Mathematical Derivation of the Odometry Motion Model

The derivation of the algorithms is relatively straightforward, and once again may be skipped at first reading. To derive a probabilistic motion model using

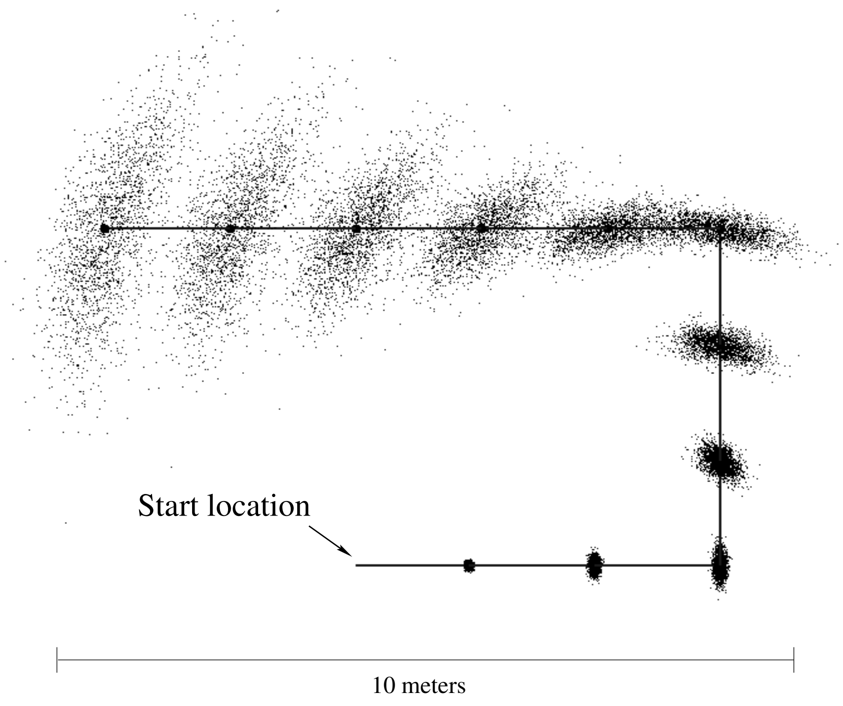

Figure 5.10 Sampling approximation of the position belief for a non-sensing robot. The solid line displays the actions, and the samples represent the robot's belief at different points in time.

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

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

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

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

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^2} with zero mean and variance b^2:

(5.37) $$\hat{\delta}_{\text{rot1}} = \delta_{\text{rot1}} - \varepsilon_{\alpha_1\delta^2_{\text{rot1}}+\alpha_2\delta^2_{\text{trans}}}$$

(5.38) $$\hat{\delta}_{\text{trans}} = \delta_{\text{trans}} - \varepsilon_{\alpha_3\ \delta^2_{\text{trans}}+\alpha_4\ \delta^2_{\text{rot1}}+\alpha_4\ \delta^2_{\text{rot2}}}$$

(5.39) $$\hat{\delta}_{\text{rot2}} = \delta_{\text{rot2}} - \varepsilon_{\alpha_1\delta^2_{\text{rot2}}+\alpha_2\delta^2_{\text{trans}}}$$

As in the previous section, ε_{b^2} is a zero-mean noise variable with variance b^2. 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,

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

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,

(5.41) $$\delta_{\text{rot1}} - \hat{\delta}_{\text{rot1}}$$

(5.42) $$\delta_{\text{trans}} - \hat{\delta}_{\text{trans}}$$

(5.43) $$\delta_{\text{rot2}} - \hat{\delta}_{\text{rot2}}$$

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

(5.44) $$p_1 = \varepsilon_{\alpha_1\delta^2_{\text{rot1}}+\alpha_2\delta^2_{\text{trans}}}(\delta_{\text{rot1}} - \hat{\delta}_{\text{rot1}})$$

(5.45) $$p_2 = \varepsilon_{\alpha_3\ \delta^2_{\text{trans}}+\alpha_4\ \delta^2_{\text{rot1}}+\alpha_4\ \delta^2_{\text{rot2}}}(\delta_{\text{trans}} - \hat{\delta}_{\text{trans}})$$

(5.46) $$p_3 = \varepsilon_{\alpha_1\delta^2_{\text{rot2}}+\alpha_2\delta^2_{\text{trans}}}(\delta_{\text{rot2}} - \hat{\delta}_{\text{rot2}})$$

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 $p_1 \cdot p_2 \cdot p_3$ (c.f., line 11).