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

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

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

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

## 4.2 THE PARTICLE FILTER

### 4.2.1 Basic Algorithm

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

that is initialized by 1 and updated multiplicatively:

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

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

### 4.2.2 Importance Sampling

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

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

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

$$w^{[m]} \quad = \quad \frac{f(x^{[m]})}{g(x^{[m]})} \tag{4.26}$$

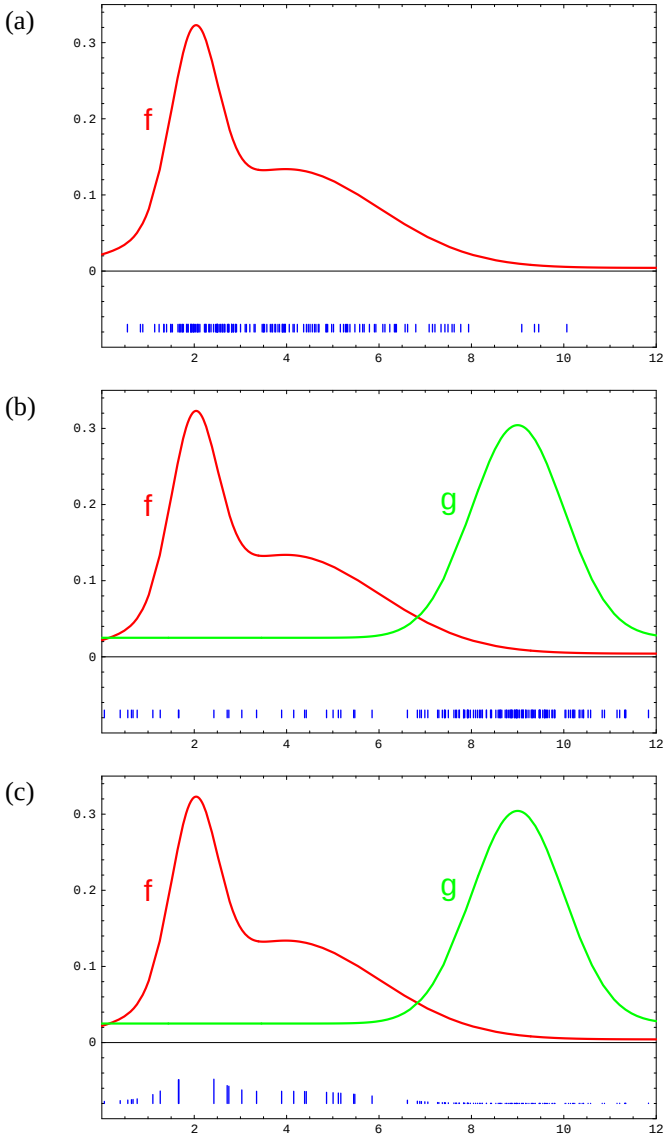


**Figure 4.2** Illustration of importance factors in particle filters: (a) We seek to approximate the target density $f$. (b) Instead of sampling from $f$ directly, we can only generate samples from a different density, $g$. Samples drawn from $g$ are shown at the bottom of this diagram. (c) A sample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

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

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

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

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

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

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

This distribution is called the *proposal distribution*.

### 4.2.3 Mathematical Derivation of the PF

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

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

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

calculates the posterior over all state sequences:

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

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

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

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

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

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

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

With

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

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

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

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

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

### 4.2.4 Properties of the Particle Filter

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

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

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

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

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

its own normalizer if $M = 1$:

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

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

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

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

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

```
1:   Algorithm Low_variance_sampler(X_t, W_t):
2:       X̄_t = ∅
3:       r = rand(0; M^{-1})
4:       c = w_t^[1]
5:       i = 1
6:       for m = 1 to M do
7:           u = r + (m − 1) · M^{-1}
8:           while u > c
9:               i = i + 1
10:              c = c + w_t^[i]
11:          endwhile
12:          add x_t^[i] to X̄_t
13:      endfor
14:      return X̄_t
```

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

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

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

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

The second strategy for reducing the sampling error is known as *low variance sampling*. Table 4.4 depicts an implementation of a low variance sampler. The basic idea is that instead of selecting samples independently of each other in the

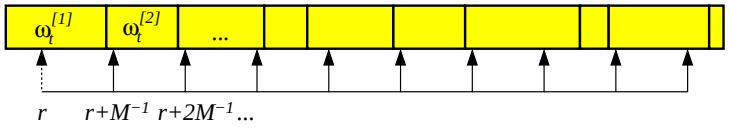


**Figure 4.3** Principle of the low variance resampling procedure. We choose a random number $r$ and then select those particles that correspond to $u = r + (m-1) \cdot M^{-1}$ where $m = 1, \ldots, M$.

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

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

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

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

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

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

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

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

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

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

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

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

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

## 4.3 SUMMARY

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