Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

b-it Bonn-Aachen International Center for Information Technology

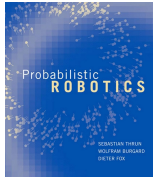
Institute for AI and Autonomous Systems

Monte Carlo Localisation

Localising Using a Particle Filter

Dr. Alex Mitrevski
Master of Autonomous Systems

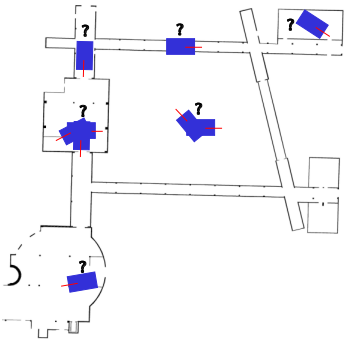
Structure



- Reminder: Robot localisation essentials
- Monte Carlo localisation
- Probabilistic motion models

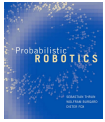
Reminder: Robot Localisation Essentials

Reminder: Localisation



- The process that determines a robot's pose in an environment is called **localisation**
- The localisation process is performed **with respect to a given map** — thus, **localisation requires a map to be given**
- Since a robot moves around, the localisation estimate needs to be continuously updated based on the **known motion commands** and **any relevant environment observations**

Reminder: Recursive State Estimation Using a Bayes Filter



- The process of continuously updating the belief based on performed motions and recorded measurements is called **recursive state estimation**; this is performed using a **Bayes filter**
- The Bayes filter continuously performs two steps:

Control update (prediction)

The belief is updated based on a performed motion $\boldsymbol{u}_t$:

$$\overline{\text{bel}}\,(\boldsymbol{x}_t) = \int p\,(\boldsymbol{x}_t|\boldsymbol{u}_t, \boldsymbol{x}_{t-1})\,\text{bel}\,(\boldsymbol{x}_{t-1})\,d\boldsymbol{x}_{t-1}$$

Measurement update (correction)

The belief is corrected based on a sensor measurement $\boldsymbol{z}_t$:

$$\text{bel}\,(\boldsymbol{x}_t) = \eta\,p\,(\boldsymbol{z}_t|\boldsymbol{x}_t)\,\overline{\text{bel}}\,(\boldsymbol{x}_t)$$

- The control update **increases the pose uncertainty**, while the measurement update **decreases it**
- Both update equations are based on the **Markov assumption**, according to which:
  - $\boldsymbol{x}_t$ only depends on $\boldsymbol{x}_{t-1}$ and $\boldsymbol{u}_t$
  - $\boldsymbol{z}_t$ only depends on $\boldsymbol{x}_t$

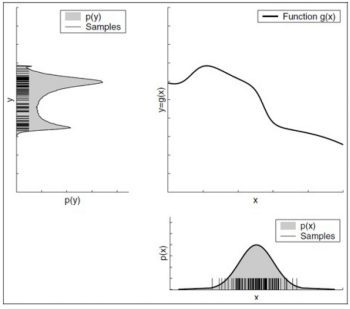
Recursive Bayes Filter Family

- The Bayes filter is not a single algorithm, but **a family of algorithms**, each of which is based on different assumptions
- We will particularly distinguish between the following variants of the recursive Bayes filter as applied to the robot localisation problem:
  - **Discrete Bayes filter**: Applicable when **the state is discretised** (discussed in MRC)
  - **Kalman filter**: Assumes that **the state is governed by a Gaussian distribution** (discussed last time)
  - **Particle filter**: Does not make an assumption about the underlying state distribution; **represents multiple hypotheses about the state by a set of particles** (discussed today)

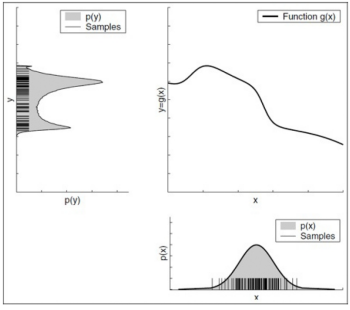
Monte Carlo Localisation

Particle Filter Preliminaries

- A particle filter is a **non-parametric representation of a Bayes filter**



Particle Filter Preliminaries

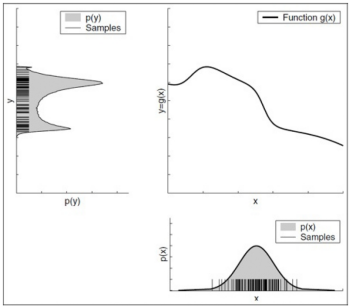


- A particle filter is a **non-parametric representation of a Bayes filter**
- The filter represents a distribution by n particles

$$P = \{\boldsymbol{p}^1, ..., \boldsymbol{p}^n\}$$

where **each particle is a weighted hypothesis** about the state, namely $\boldsymbol{p}^i = \left(\boldsymbol{x}^i, w^i\right)$

Particle Filter Preliminaries



- A particle filter is a **non-parametric representation of a Bayes filter**
- The filter represents a distribution by n particles

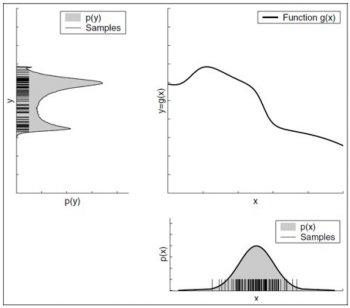
$$P = \{\boldsymbol{p}^1, ..., \boldsymbol{p}^n\}$$

where **each particle is a weighted hypothesis** about the state, namely $\boldsymbol{p}^i = \left(\boldsymbol{x}^i, w^i\right)$

- The objective is to include particles $\boldsymbol{p}^i, 1 \leq i \leq n$ in P with a **probability that is proportional to the estimated posterior distribution**:

$$\boldsymbol{x}_{t+1}^i \sim p(\boldsymbol{x}_{t+1} | \boldsymbol{x}_{0:t}, \boldsymbol{u}_{0:t}, \boldsymbol{z}_{1:t})$$

Particle Filter Preliminaries



- A particle filter is a **non-parametric representation of a Bayes filter**
- The filter represents a distribution by n particles

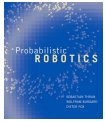
  $$P = \{\boldsymbol{p}^1, ..., \boldsymbol{p}^n\}$$

  where **each particle is a weighted hypothesis** about the state, namely $\boldsymbol{p}^i = \left(\boldsymbol{x}^i, w^i\right)$
- The objective is to include particles $\boldsymbol{p}^i, 1 \leq i \leq n$ in P with a **probability that is proportional to the estimated posterior distribution**:

  $$\boldsymbol{x}_{t+1}^i \sim p(\boldsymbol{x}_{t+1} | \boldsymbol{x}_{0:t}, \boldsymbol{u}_{0:t}, \boldsymbol{z}_{1:t})$$

- A particle filter is a general instantiation of the Bayes filter and has been used in a variety of problems, for instance localisation, object tracking, or fault diagnosis

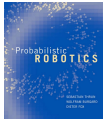
Particle Filter Algorithm



- In a particle, filter, **both the motion and the measurement update are operations on the particle set P_t at time t:**

1: **function** PARTICLEFILTER(P_t, $\boldsymbol{u}_t$, $\boldsymbol{z}_t$, $\mathcal{M}$)
2: $\quad\hat{P} \leftarrow \{\}$
3: $\quad$sampling_dist $\leftarrow$ []
4: $\quad$**for** $i \leftarrow 1$ **to** n **do**
5: $\quad\quad\boldsymbol{x}_t^i \leftarrow P^i.\boldsymbol{x}$
6: $\quad\quad\boldsymbol{x}_{t+1}^i \sim p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t^i, \boldsymbol{u}_t)$
7: $\quad\quad w^i \leftarrow p(\boldsymbol{z}_{t+1}|\boldsymbol{x}_{t+1}, \mathcal{M})$
8: $\quad\quad\hat{P} \leftarrow \hat{P} \cup \left(\boldsymbol{x}_{t+1}^i, w^i\right)$
9: $\quad\quad$sampling_dist.append(w^i)
10: $\quad$sampling_dist $\leftarrow \eta \cdot$ sampling_dist
11: $\quad P_{t+1} \leftarrow \{\}$
12: $\quad$**for** $i \leftarrow 1$ **to** n **do**
13: $\quad\quad p^i \leftarrow$ sample($\hat{P}$, sampling_dist)
14: $\quad\quad P_{t+1} \leftarrow P_{t+1} \cup p^i$
15: $\quad$**return** P_{t+1}

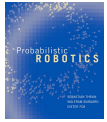
Particle Filter Algorithm



1: **function** PARTICLEFILTER(P_t, $\boldsymbol{u}_t$, $\boldsymbol{z}_t$, $\mathcal{M}$)
2: $\hat{P} \leftarrow \{\}$
3: sampling_dist $\leftarrow []$
4: **for** $i \leftarrow 1$ **to** n **do**
5: $\boldsymbol{x}_t^i \leftarrow P^i.\boldsymbol{x}$
6: $\boldsymbol{x}_{t+1}^i \sim p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t^i, \boldsymbol{u}_t)$
7: $w^i \leftarrow p(\boldsymbol{z}_{t+1}|\boldsymbol{x}_{t+1}, \mathcal{M})$
8: $\hat{P} \leftarrow \hat{P} \cup (\boldsymbol{x}_{t+1}^i, w^i)$
9: sampling_dist.append(w^i)
10: sampling_dist $\leftarrow \eta \cdot$ sampling_dist
11: $P_{t+1} \leftarrow \{\}$
12: **for** $i \leftarrow 1$ **to** n **do**
13: $p^i \leftarrow$ sample($\hat{P}$, sampling_dist)
14: $P_{t+1} \leftarrow P_{t+1} \cup p^i$
15: **return** P_{t+1}

- In a particle, filter, **both the motion and the measurement update are operations on the particle set P_t at time t**:
  - **Motion update**: The states of the particles are updated by sampling from a **motion model**

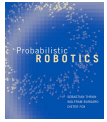
Particle Filter Algorithm



1: **function** PARTICLEFILTER(P_t, $\boldsymbol{u}_t$, $\boldsymbol{z}_t$, $\mathcal{M}$)
2: $\quad\hat{P} \leftarrow \{\}$
3: $\quad$sampling_dist $\leftarrow []$
4: $\quad$**for** $i \leftarrow 1$ **to** n **do**
5: $\quad\quad\boldsymbol{x}_t^i \leftarrow P^i.\boldsymbol{x}$
6: $\quad\quad\boldsymbol{x}_{t+1}^i \sim p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t^i, \boldsymbol{u}_t)$
7: $\quad\quad w^i \leftarrow p(\boldsymbol{z}_{t+1}|\boldsymbol{x}_{t+1}, \mathcal{M})$
8: $\quad\quad\hat{P} \leftarrow \hat{P} \cup (\boldsymbol{x}_{t+1}^i, w^i)$
9: $\quad\quad$sampling_dist.append(w^i)
10: $\quad$sampling_dist $\leftarrow \eta \cdot$ sampling_dist
11: $\quad P_{t+1} \leftarrow \{\}$
12: $\quad$**for** $i \leftarrow 1$ **to** n **do**
13: $\quad\quad p^i \leftarrow$ sample($\hat{P}$, sampling_dist)
14: $\quad\quad P_{t+1} \leftarrow P_{t+1} \cup p^i$
15: $\quad$**return** P_{t+1}

- In a particle, filter, **both the motion and the measurement update are operations on the particle set P_t at time t**:
  - **Motion update**: The states of the particles are updated by sampling from a **motion model**
  - **Measurement update**: The particles' weights are updated based on the **measurement model** with respect to a map $\mathcal{M}$

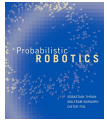
Particle Filter Algorithm



1: **function** PARTICLEFILTER(P_t, $\boldsymbol{u}_t$, $\boldsymbol{z}_t$, $\mathcal{M}$)
2: $\quad\hat{P} \leftarrow \{\}$
3: $\quad$sampling_dist $\leftarrow []$
4: $\quad$**for** $i \leftarrow 1$ **to** n **do**
5: $\quad\quad\boldsymbol{x}_t^i \leftarrow P^i.\boldsymbol{x}$
6: $\quad\quad\boldsymbol{x}_{t+1}^i \sim p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t^i, \boldsymbol{u}_t)$
7: $\quad\quad w^i \leftarrow p(\boldsymbol{z}_{t+1}|\boldsymbol{x}_{t+1}, \mathcal{M})$
8: $\quad\quad\hat{P} \leftarrow \hat{P} \cup (\boldsymbol{x}_{t+1}^i, w^i)$
9: $\quad\quad$sampling_dist.append(w^i)
10: $\quad$sampling_dist $\leftarrow \eta \cdot$ sampling_dist
11: $\quad P_{t+1} \leftarrow \{\}$
12: $\quad$**for** $i \leftarrow 1$ **to** n **do**
13: $\quad\quad p^i \leftarrow$ sample($\hat{P}$, sampling_dist)
14: $\quad\quad P_{t+1} \leftarrow P_{t+1} \cup p^i$
15: $\quad$**return** P_{t+1}

- In a particle, filter, **both the motion and the measurement update are operations on the particle set P_t at time t**:
  - **Motion update**: The states of the particles are updated by sampling from a **motion model**
  - **Measurement update**: The particles' weights are updated based on the **measurement model** with respect to a map $\mathcal{M}$
- After these two steps, the posterior distribution is updated — this is achieved by **resampling the particle set**

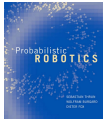
Particle Filter Algorithm



1: **function** PARTICLEFILTER(P_t, $\boldsymbol{u}_t$, $\boldsymbol{z}_t$, $\mathcal{M}$)
2: $\quad\hat{P} \leftarrow \{\}$
3: $\quad$sampling_dist $\leftarrow []$
4: $\quad$**for** $i \leftarrow 1$ **to** n **do**
5: $\quad\quad\boldsymbol{x}_t^i \leftarrow P^i.\boldsymbol{x}$
6: $\quad\quad\boldsymbol{x}_{t+1}^i \sim p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t^i, \boldsymbol{u}_t)$
7: $\quad\quad w^i \leftarrow p(\boldsymbol{z}_{t+1}|\boldsymbol{x}_{t+1}, \mathcal{M})$
8: $\quad\quad\hat{P} \leftarrow \hat{P} \cup (\boldsymbol{x}_{t+1}^i, w^i)$
9: $\quad\quad$sampling_dist.append(w^i)
10: $\quad$sampling_dist $\leftarrow \eta \cdot$ sampling_dist
11: $\quad P_{t+1} \leftarrow \{\}$
12: $\quad$**for** $i \leftarrow 1$ **to** n **do**
13: $\quad\quad p^i \leftarrow$ sample($\hat{P}$, sampling_dist)
14: $\quad\quad P_{t+1} \leftarrow P_{t+1} \cup p^i$
15: $\quad$**return** P_{t+1}

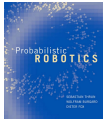
- In a particle, filter, **both the motion and the measurement update are operations on the particle set P_t at time t**:
  - **Motion update**: The states of the particles are updated by sampling from a **motion model**
  - **Measurement update**: The particles' weights are updated based on the **measurement model** with respect to a map $\mathcal{M}$
- After these two steps, the posterior distribution is updated — this is achieved by **resampling the particle set**
- Resampling is performed using a procedure called **importance sampling**, during which particles from P_t are **sampled with replacement proportional to their weights**; this results in a new particle set P_{t+1}

Importance Sampling



- Importance sampling is a technique based on which samples are generated from a **proposal distribution** $g(x)$ instead of a **target distribution** $f(x)$
  - g is defined so that, if $f(x) > 0$, then it should also be the case that $g(x) > 0$

Importance Sampling

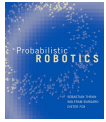


- Importance sampling is a technique based on which samples are generated from a **proposal distribution** $g(x)$ instead of a **target distribution** $f(x)$
  - g is defined so that, if $f(x) > 0$, then it should also be the case that $g(x) > 0$

- Given a random variable X with a density $f(x)$, its expected value can be calculated as

$$\begin{aligned} E_f[X] &= \int f(x)x dx \\ &= \int \frac{f(x)}{g(x)} g(x)x dx \\ &= \int w(x)g(x)x dx = E_g[W(X)X] \end{aligned}$$

Importance Sampling



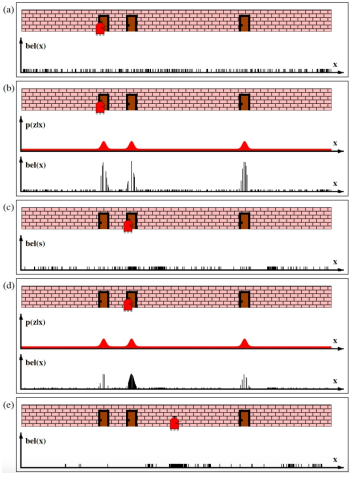
- Importance sampling is a technique based on which samples are generated from a **proposal distribution** $g(x)$ instead of a **target distribution** $f(x)$
  - g is defined so that, if $f(x) > 0$, then it should also be the case that $g(x) > 0$

- Given a random variable X with a density $f(x)$, its expected value can be calculated as

$$\begin{aligned} E_f[X] &= \int f(x)x dx \\ &= \int \frac{f(x)}{g(x)} g(x)x dx \\ &= \int w(x)g(x)x dx = E_g[W(X)X] \end{aligned}$$

- In a particle filter:
  - the proposal distribution $g(\boldsymbol{x}) = p(\boldsymbol{x}_{t+1}|\boldsymbol{u}_{t+1}, \boldsymbol{x}_t)\underbrace{p(\boldsymbol{x}_{0:t}|\boldsymbol{u}_{0:t}, \boldsymbol{z}_{1:t})}_{\text{bel}(\boldsymbol{x}_t)}$
  - the target distribution $f(\boldsymbol{x}) = p(\boldsymbol{z}_{t+1}|\boldsymbol{x}_{t+1})p(\boldsymbol{x}_{t+1}|\boldsymbol{u}_{t+1}, \boldsymbol{x}_t)\underbrace{p(\boldsymbol{x}_{0:t}|\boldsymbol{u}_{0:t}, \boldsymbol{z}_{1:t})}_{\text{bel}(\boldsymbol{x}_t)}$
  - the weights $w^i \propto p(\boldsymbol{z}_{t+1}|\boldsymbol{x}_{t+1})$

Monte Carlo Localisation

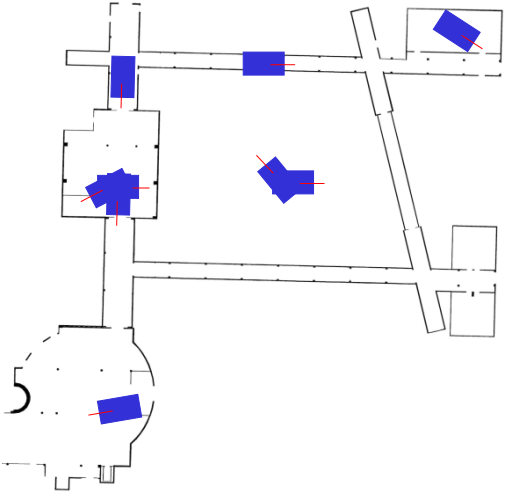


- Localisation performed using particle filters is called **Monte Carlo localisation**
  - Monte Carlo methods represent a general category of techniques based on which solutions to problems are discovered using random sampling
- In the most general case, the initial state of the robot is fully unknown; this can be expressed by **sampling the initial particle set P_0 from a uniform distribution**

Particle Filter Illustration: Motion Update

During the motion update step, each particle is updated according to the motion model

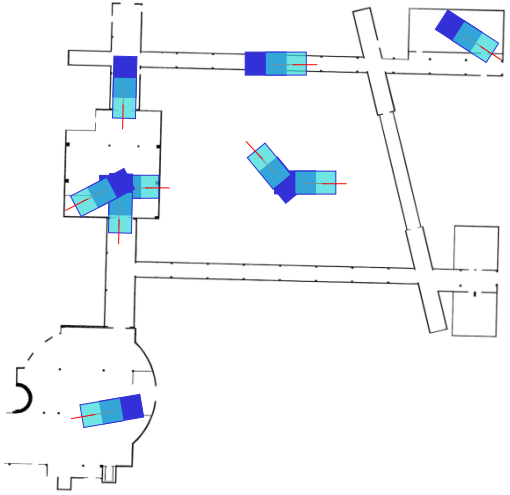
1: **function** PARTICLEFILTER(P_t, $\boldsymbol{u}_t$, $\boldsymbol{z}_t$, $\mathcal{M}$)
2: $\hat{P} \leftarrow \{\}$
3: sampling_dist $\leftarrow []$
4: **for** $i \leftarrow 1$ **to** n **do**
5: $\boldsymbol{x}_t^i \leftarrow P^i.\boldsymbol{x}$
6: $\boldsymbol{x}_{t+1}^i \sim p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t^i, \boldsymbol{u}_t)$
7: $w^i \leftarrow p(\boldsymbol{z}_{t+1}|\boldsymbol{x}_{t+1}, \mathcal{M})$
8: $\hat{P} \leftarrow \hat{P} \cup \left(\boldsymbol{x}_{t+1}^i, w^i\right)$
9: sampling_dist.append(w^i)
10: sampling_dist $\leftarrow \eta \cdot$ sampling_dist
11: $P_{t+1} \leftarrow \{\}$
12: **for** $i \leftarrow 1$ **to** n **do**
13: $p^i \leftarrow$ sample($\hat{P}$, sampling_dist)
14: $P_{t+1} \leftarrow P_{t+1} \cup p^i$
15: **return** P_{t+1}



Particle Filter Illustration: Motion Update

During the motion update step, each particle is updated according to the motion model

1: **function** PARTICLEFILTER(P_t, $\boldsymbol{u}_t$, $\boldsymbol{z}_t$, $\mathcal{M}$)
2: $\quad\hat{P} \leftarrow \{\}$
3: $\quad$sampling_dist $\leftarrow []$
4: $\quad$**for** $i \leftarrow 1$ **to** n **do**
5: $\quad\quad\boldsymbol{x}_t^i \leftarrow P^i.\boldsymbol{x}$
6: $\quad\quad\boldsymbol{x}_{t+1}^i \sim p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t^i, \boldsymbol{u}_t)$
7: $\quad\quad w^i \leftarrow p(\boldsymbol{z}_{t+1}|\boldsymbol{x}_{t+1}, \mathcal{M})$
8: $\quad\quad\hat{P} \leftarrow \hat{P} \cup (\boldsymbol{x}_{t+1}^i, w^i)$
9: $\quad\quad$sampling_dist.append(w^i)
10: $\quad$sampling_dist $\leftarrow \eta \cdot$ sampling_dist
11: $\quad P_{t+1} \leftarrow \{\}$
12: $\quad$**for** $i \leftarrow 1$ **to** n **do**
13: $\quad\quad p^i \leftarrow$ sample($\hat{P}$, sampling_dist)
14: $\quad\quad P_{t+1} \leftarrow P_{t+1} \cup p^i$
15: $\quad$**return** P_{t+1}



Particle Filter Illustration: Measurement Update

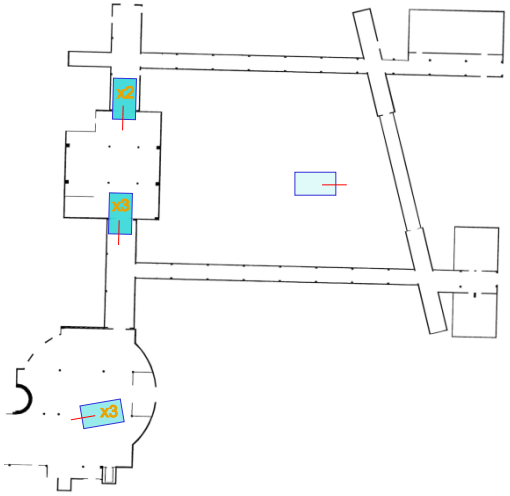
During the measurement update, each particle is reweighed based on a given observation — particles that result in likelier measurements are assigned higher weights

1: **function** PARTICLEFILTER(P_t, $\boldsymbol{u}_t$, $\boldsymbol{z}_t$, $\mathcal{M}$)
2: $\quad\hat{P} \leftarrow \{\}$
3: $\quad$sampling_dist $\leftarrow []$
4: $\quad$**for** $i \leftarrow 1$ **to** n **do**
5: $\quad\quad\boldsymbol{x}_t^i \leftarrow P^i.\boldsymbol{x}$
6: $\quad\quad\boldsymbol{x}_{t+1}^i \sim p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t^i, \boldsymbol{u}_t)$
7: $\quad\quad w^i \leftarrow p(\boldsymbol{z}_{t+1}|\boldsymbol{x}_{t+1}, \mathcal{M})$
8: $\quad\quad\hat{P} \leftarrow \hat{P} \cup (\boldsymbol{x}_{t+1}^i, w^i)$
9: $\quad\quad$sampling_dist.append(w^i)
10: $\quad$sampling_dist $\leftarrow \eta \cdot$ sampling_dist
11: $\quad P_{t+1} \leftarrow \{\}$
12: $\quad$**for** $i \leftarrow 1$ **to** n **do**
13: $\quad\quad p^i \leftarrow$ sample($\hat{P}$, sampling_dist)
14: $\quad\quad P_{t+1} \leftarrow P_{t+1} \cup p^i$
15: $\quad$**return** P_{t+1}

Particle Filter Illustration: Particle Resampling

The particles are finally resampled (with replacement) proportional to their weights

1: **function** PARTICLEFILTER(P_t, $\boldsymbol{u}_t$, $\boldsymbol{z}_t$, $\mathcal{M}$)
2: $\quad\hat{P} \leftarrow \{\}$
3: $\quad$sampling_dist $\leftarrow []$
4: $\quad$**for** $i \leftarrow 1$ **to** n **do**
5: $\quad\quad\boldsymbol{x}_t^i \leftarrow P^i.\boldsymbol{x}$
6: $\quad\quad\boldsymbol{x}_{t+1}^i \sim p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t^i, \boldsymbol{u}_t)$
7: $\quad\quad w^i \leftarrow p(\boldsymbol{z}_{t+1}|\boldsymbol{x}_{t+1}, \mathcal{M})$
8: $\quad\quad\hat{P} \leftarrow \hat{P} \cup (\boldsymbol{x}_{t+1}^i, w^i)$
9: $\quad\quad$sampling_dist.append(w^i)
10: $\quad$sampling_dist $\leftarrow \eta \cdot$ sampling_dist
11: $\quad P_{t+1} \leftarrow \{\}$
12: $\quad$**for** $i \leftarrow 1$ **to** n **do**
13: $\quad\quad p^i \leftarrow$ sample($\hat{P}$, sampling_dist)
14: $\quad\quad P_{t+1} \leftarrow P_{t+1} \cup p^i$
15: $\quad$**return** P_{t+1}



Extracting the Robot's State

- In localisation, we are typically **interested in a single estimate of a robot's state** — the states of all particles are not of direct interest

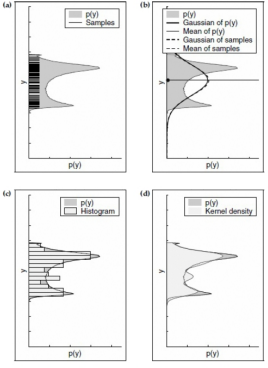


Figure 4.5 Different ways of extracting densities from particles. (a) Density and sample set approximation, (b) Gaussian approximation (mean and variance), (c) histogram approximation, (d) kernel density estimate. The choice of approximation strongly depends on the specific application and the computational resources.

Extracting the Robot's State

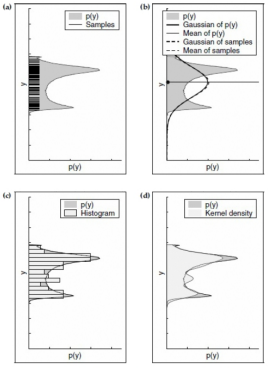


Figure 4.5 Different ways of extracting densities from particles. (a) Density and sample set approximation, (b) Gaussian approximation (mean and variance), (c) histogram approximation, (d) kernel density estimate. The choice of approximation strongly depends on the specific application and the computational resources.

- In localisation, we are typically **interested in a single estimate of a robot's state** — the states of all particles are not of direct interest
- One way to achieve this is to **represent the state as the weighted mean of the particles:** $\tilde{\boldsymbol{x}}_t = \sum_{i=1}^{n} w^i \boldsymbol{x}_t^i$
  - This corresponds to **approximating the unknown distribution $f(x)$ by a Gaussian distribution**
  - The variance of this distribution provides **information about how certain we can be in the state approximation**
  - The variance information can be used as a heuristic to **reinitialise the filter**

Extracting the Robot's State

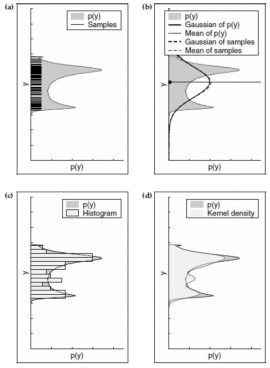
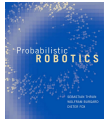


Figure 4.5 Different ways of extracting densities from particles. (a) Density and sample set approximation, (b) Gaussian approximation (mean and variance), (c) histogram approximation, (d) kernel density estimate. The choice of approximation strongly depends on the specific application and the computational resources.

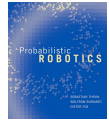
- In localisation, we are typically **interested in a single estimate of a robot's state** — the states of all particles are not of direct interest
- One way to achieve this is to **represent the state as the weighted mean of the particles:** $\tilde{\boldsymbol{x}}_t = \sum_{i=1}^{n} w^i \boldsymbol{x}_t^i$
  - This corresponds to **approximating the unknown distribution $f(x)$ by a Gaussian distribution**
  - The variance of this distribution provides **information about how certain we can be in the state approximation**
  - The variance information can be used as a heuristic to **reinitialise the filter**
- The particles can also be used to approximate the density more accurately, for instance using a **histogram** or with **kernel density estimation**

Particle Deprivation and Kidnapped Robot Problem Avoidance



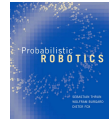
- Unlike the Kalman filter, the particle filter can, in principle, be used to solve the kidnapped robot problem and thus **recover from localisation failures**
  - This is due to the fact that the particle filter can model multimodal distributions, while the Kalman filter represents the underlying state distribution by a unimodal Gaussian distribution

Particle Deprivation and Kidnapped Robot Problem Avoidance



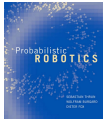
- Unlike the Kalman filter, the particle filter can, in principle, be used to solve the kidnapped robot problem and thus **recover from localisation failures**
  - This is due to the fact that the particle filter can model multimodal distributions, while the Kalman filter represents the underlying state distribution by a unimodal Gaussian distribution
- The most common way to avoid localisation failures is to **include random particles in P_t during the resampling step**, typically sampled from a uniform distribution

Particle Deprivation and Kidnapped Robot Problem Avoidance



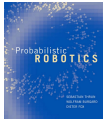
- Unlike the Kalman filter, the particle filter can, in principle, be used to solve the kidnapped robot problem and thus **recover from localisation failures**
  - This is due to the fact that the particle filter can model multimodal distributions, while the Kalman filter represents the underlying state distribution by a unimodal Gaussian distribution
- The most common way to avoid localisation failures is to **include random particles in P_t during the resampling step**, typically sampled from a uniform distribution
- While random particles can be included continuously and with the same frequency, a better strategy for this is to **couple the inclusion of random particles to the evolution of the measurement likelihood**
  - This is an intuitive principle based on which more particles are included only if the particles' measurement likelihood reduces over time

Adaptive Monte Carlo Localisation (AMCL)



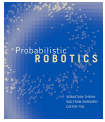
- The basic particle filter algorithm uses **a fixed number of samples n**
  - The number of particles required for accurately approximating the state is typically high (in the order of thousands or even tens of thousands) — although this depends on the map size

Adaptive Monte Carlo Localisation (AMCL)



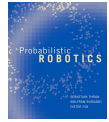
- The basic particle filter algorithm uses **a fixed number of samples n**
  - The number of particles required for accurately approximating the state is typically high (in the order of thousands or even tens of thousands) — although this depends on the map size
- In practice, using a fixed number of particles is computationally suboptimal, **particularly once the particles converge to a state estimate**

Adaptive Monte Carlo Localisation (AMCL)



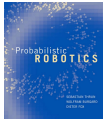
- The basic particle filter algorithm uses **a fixed number of samples n**
  - The number of particles required for accurately approximating the state is typically high (in the order of thousands or even tens of thousands) — although this depends on the map size
- In practice, using a fixed number of particles is computationally suboptimal, **particularly once the particles converge to a state estimate**
- A practically more useful approach is to **vary the number of particles depending on the distribution approximation quality** — this is the approach followed by commonly used localisation frameworks, such as AMCL in ROS

Adaptive Monte Carlo Localisation (AMCL)



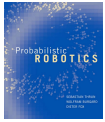
- The basic particle filter algorithm uses **a fixed number of samples n**
  - The number of particles required for accurately approximating the state is typically high (in the order of thousands or even tens of thousands) — although this depends on the map size
- In practice, using a fixed number of particles is computationally suboptimal, **particularly once the particles converge to a state estimate**
- A practically more useful approach is to **vary the number of particles depending on the distribution approximation quality** — this is the approach followed by commonly used localisation frameworks, such as AMCL in ROS
- AMCL modifies the particle resampling step so that **particles are sampled until a desired distribution quality is reached**
  - The distribution quality is measured by **overlaying a discrete grid over the map** and **calculating an approximation bound based on the number of grid cells with non-zero particles**
  - The concrete sampling technique used by AMCL is called **KLD-sampling**

Handling Dynamic Environments



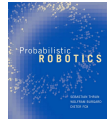
- The Monte Carlo localisation algorithm discussed above is actually only applicable to static environments — **dynamic obstacles will have a direct effect on the measurements, but this effect is not considered by the measurement model**

Handling Dynamic Environments



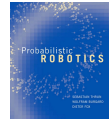
- The Monte Carlo localisation algorithm discussed above is actually only applicable to static environments — **dynamic obstacles will have a direct effect on the measurements, but this effect is not considered by the measurement model**
- Two main strategies can be used to handle dynamic obstacles during localisation:

Handling Dynamic Environments



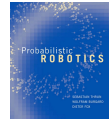
- The Monte Carlo localisation algorithm discussed above is actually only applicable to static environments — **dynamic obstacles will have a direct effect on the measurements, but this effect is not considered by the measurement model**
- Two main strategies can be used to handle dynamic obstacles during localisation:
  - **Modelling dynamic obstacles in the measurement model**: This is the most general way of handling dynamic obstacles, but is typically inefficient — a variable state dimensionality needs to be considered

Handling Dynamic Environments



- The Monte Carlo localisation algorithm discussed above is actually only applicable to static environments — **dynamic obstacles will have a direct effect on the measurements, but this effect is not considered by the measurement model**
- Two main strategies can be used to handle dynamic obstacles during localisation:
  - **Modelling dynamic obstacles in the measurement model**: This is the most general way of handling dynamic obstacles, but is typically inefficient — a variable state dimensionality needs to be considered
  - **Outlier rejection**: Using this strategy, measurements from dynamic obstacles are rejected before the particle filter performs a measurement update

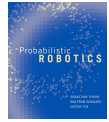
Handling Dynamic Environments



- The Monte Carlo localisation algorithm discussed above is actually only applicable to static environments — **dynamic obstacles will have a direct effect on the measurements, but this effect is not considered by the measurement model**
- Two main strategies can be used to handle dynamic obstacles during localisation:
  - **Modelling dynamic obstacles in the measurement model**: This is the most general way of handling dynamic obstacles, but is typically inefficient — a variable state dimensionality needs to be considered
  - **Outlier rejection**: Using this strategy, measurements from dynamic obstacles are rejected before the particle filter performs a measurement update
- Particularly outlier rejection requires a **model of dynamic obstacles**; such a model may be learned

Probabilistic Motion Models

Probabilistic Motion Models



- The particle filter performs a motion update of each particle based on a motion command $\boldsymbol{u}_{t+1}$; this update is **not deterministic, but probabilistic** — by sampling from a motion model $p(\boldsymbol{x}_{t+1}|\boldsymbol{u}_{t+1}, \boldsymbol{x}_t)$ — so that it takes the motion uncertainty into account
- The used motion model depends on the **type of motion command that is sent to a robot**
- We will briefly look at two types of **motion models for planar motion**:
  - **Velocity motion model**: The motion command is specified by linear and angular velocities: $\boldsymbol{u} = (v, \omega)^T$
  - **Odometry motion model**: The motion command is given by relative odometry $\boldsymbol{u} = (\overline{\boldsymbol{x}}_t, \overline{\boldsymbol{x}}_{t+1})^T$, where $\overline{\boldsymbol{x}}_t = (\overline{x}, \overline{y}, \overline{\theta})^T$ and $\overline{\boldsymbol{x}}_{t+1} = (\overline{x}', \overline{y}', \overline{\theta}')$

Velocity Motion Model

- Given a velocity command $(v, \omega)^T$, a robot moves along a circle of radius $r = \left|\frac{v}{\omega}\right|$ with center $\left(x - \frac{v}{\omega}\sin\theta, y + \frac{v}{\omega}\cos\theta\right)^T$

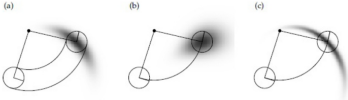


Figure 5.3 The velocity motion model, for different noise parameter settings.

- Based on the velocity motion model, a robot is **controlled by a velocity command under the influence of noise**:

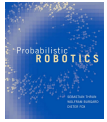
$$\begin{pmatrix}\hat{v}\\ \hat{\omega}\end{pmatrix} = \begin{pmatrix}v\\ \omega\end{pmatrix} + \begin{pmatrix}\epsilon_{\alpha_1 v^2 + \alpha_2 \omega^2}\\ \epsilon_{\alpha_3 v^2 + \alpha_4 \omega^2}\end{pmatrix}$$

- The robot is also **assumed to undergo a final orientation once the goal location is reached**; the final orientation is thus given by $\theta' = \theta + \hat{\omega}\Delta t + \hat{\gamma}\Delta t$ with $\hat{\gamma} = \epsilon_{\alpha_5 v^2 + \alpha_6 \omega^2}$

- Under the velocity motion model, the pose update is given as

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

Velocity Motion Model Sampling



- In the particle filter, we are interested in sampling from the velocity motion model; this sampling can be done using the algorithm below

1: **function** SAMPLEVELOCITYMOTIONMODEL($\boldsymbol{x}_t = (x, y, \theta)$, $\boldsymbol{u}_t = (v, \omega)$)
2: $\quad \hat{v} \leftarrow v + \texttt{sample}(\alpha_1 v^2 + \alpha_2 \omega^2)$
3: $\quad \hat{\omega} \leftarrow \omega + \texttt{sample}(\alpha_3 v^2 + \alpha_4 \omega^2)$
4: $\quad \hat{\gamma} \leftarrow \texttt{sample}(\alpha_5 v^2 + \alpha_6 \omega^2)$

5: $\quad x' \leftarrow x - \frac{\hat{v}}{\hat{\omega}} \sin\theta + \frac{\hat{v}}{\hat{\omega}} \sin(\theta + \hat{\omega}\Delta t)$
6: $\quad y' \leftarrow y + \frac{\hat{v}}{\hat{\omega}} \cos\theta - \frac{\hat{v}}{\hat{\omega}} \cos(\theta + \hat{\omega}\Delta t)$
7: $\quad \theta' \leftarrow \theta + \hat{\omega}\Delta t + \hat{\gamma}\Delta t$
8: $\quad$ **return** $(x', y', \theta')^T$

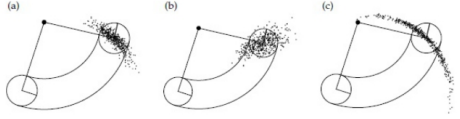


Figure 5.4 Sampling from the velocity motion model, using the same parameters as in Figure 5.3. Each diagram shows 500 samples.

Odometry Motion Model

- Given an initial pose $\boldsymbol{x}_t$ and a final pose $\boldsymbol{x}_{t+1}$, the motion can be modelled by an **initial rotation** δ_{rot_1} towards the goal, a **translational motion** δ_{trans}, and a **final rotation** δ_{rot_2}

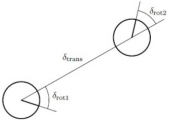


Figure 5.7 Odometry model: The robot motion in the time interval $(t-1, t]$ is approximated by a rotation δ_{rot1}, followed by a translation δ_{trans} and a second rotation δ_{rot2}. The turns and translations are noisy.

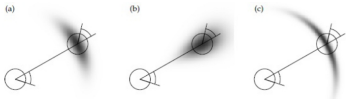


Figure 5.8 The odometry motion model, for different noise parameter settings.

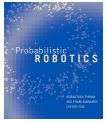
- Under the odometry motion model, **the components of the motion are all under the influence of noise**:

$$\begin{pmatrix} \hat{\delta}_{\text{rot}_1} \\ \hat{\delta}_{\text{rot}_2} \\ \hat{\delta}_{\text{trans}} \end{pmatrix} = \begin{pmatrix} \delta_{\text{rot}_1} \\ \delta_{\text{rot}_2} \\ \delta_{\text{trans}} \end{pmatrix} + \begin{pmatrix} \epsilon_{\alpha_1 \delta^2_{\text{rot}_1} + \alpha_2 \delta^2_{\text{trans}}} \\ \epsilon_{\alpha_1 \delta^2_{\text{rot}_1} + \alpha_2 \delta^2_{\text{trans}}} \\ \epsilon_{\alpha_3 \delta^2_{\text{trans}} + \alpha_4 \delta^2_{\text{rot}_1} + \alpha_4 \delta^2_{\text{rot}_2}} \end{pmatrix}$$

- The pose update under the odometry motion model is thus given as

$$\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} \hat{\delta}_{\text{trans}} \cos(\theta + \hat{\delta}_{\text{rot}_1}) \\ \hat{\delta}_{\text{trans}} \sin(\theta + \hat{\delta}_{\text{rot}_1}) \\ \hat{\delta}_{\text{rot}_1} + \hat{\delta}_{\text{rot}_2} \end{pmatrix}$$

Odometry Motion Model Sampling



- The algorithm below can be used for sampling from the odometry motion model

1: **function** SAMPLEODOMETRYMOTIONMODEL($\boldsymbol{x}_t = (x, y, \theta)$, $\boldsymbol{u}_t = (\overline{\boldsymbol{x}}_{t-1}, \overline{\boldsymbol{x}}_t)$)

2: $\quad \delta_{\text{rot}_1} \leftarrow \text{atan2}(\overline{y}' - \overline{y}, \overline{x}' - \overline{x}) - \theta$

3: $\quad \delta_{\text{trans}} \leftarrow \sqrt{(\overline{x}' - \overline{x})^2 + (\overline{y}' - \overline{y})^2}$

4: $\quad \delta_{\text{rot}_2} \leftarrow \overline{\theta}' - \overline{\theta} - \delta_{\text{rot}_1}$

5: $\quad \hat{\delta}_{\text{rot}_1} \leftarrow \delta_{\text{rot}_1} - \texttt{sample}\left(\alpha_1 \delta^2_{\text{rot}_1} + \alpha_2 \delta^2_{\text{trans}}\right)$

6: $\quad \hat{\delta}_{\text{trans}} \leftarrow \delta_{\text{trans}} - \texttt{sample}\left(\alpha_3 \delta^2_{\text{trans}} + \alpha_4 \delta^2_{\text{rot}_1} + \alpha_4 \delta^2_{\text{rot}_2}\right)$

7: $\quad \hat{\delta}_{\text{rot}_2} \leftarrow \delta_{\text{rot}_2} - \texttt{sample}\left(\alpha_1 \delta^2_{\text{rot}_2} + \alpha_2 \delta^2_{\text{trans}}\right)$

8: $\quad x' \leftarrow x + \hat{\delta}_{\text{trans}} \cos(\theta + \hat{\delta}_{\text{rot}_1})$

9: $\quad y' \leftarrow y + \hat{\delta}_{\text{trans}} \sin(\theta + \hat{\delta}_{\text{rot}_1})$

10: $\quad \theta' \leftarrow \theta + \hat{\delta}_{\text{rot}_1} + \hat{\delta}_{\text{rot}_2}$

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

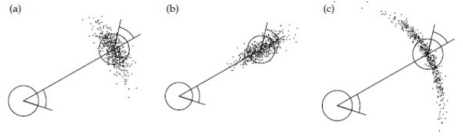


Figure 5.9 Sampling from the odometry motion model, using the same parameters as in Figure 5.8. Each diagram shows

Determining the Noise Parameters

- Both the velocity motion model and the odometry motion model represent **noise by a set of parameters $\alpha_{1:6}$ and $\alpha_{1:4}$, respectively**
- **The noise parameters are robot-specific** — the noise induced by a given motion varies between robots, as it depends on the concrete kinematic structure
- These parameters need to be **determined experimentally**, using a procedure as follows:
  1. **Collect a dataset of robot motions**
  2. **Calculate the probability $p(\boldsymbol{x}_{t+1}|\boldsymbol{u}_{t+1}, \boldsymbol{x}_t)$ under the chosen motion model** — algorithms for calculating the probability are given in the reference book
  3. **Perform parameter estimation**, for instance using maximum likelihood estimation (MLE)

Summary

- A particle filter approximates an unknown state distribution by a discrete set of particles
  - Each particle represents a weighted hypothesis about the state, where the weight is determined by the measurement model
- Particle filters update the approximation of the state distribution by performing importance sampling
  - At each iteration, particles are sampled with replacement; the probability that a particle is sampled is proportional to its weight
- Unlike a Kalman filter, a particle filter can, in principle, be used to recover from the kidnapped robot problem
  - For this, randomly distributed particles can be added during the particle resampling process
- Probabilistic motion models, such as the velocity motion model and the odometry motion model, can be used for recursive state estimation using particle filters